Chapter 8

Name Resolution and Service Discovery on the Internet and in Ad Hoc Networks

Paal E. Engelstad and Geir Egeland

CONTENTS

Name Resolution ..................................................................................................... 172
   An Architecture for Naming Services .......................................................... 173
   A Generic Model .......................................................................................... 173
   The Domain Name System ......................................................................... 176
Resolving Names Without the Use of DNS Servers ........................................ 177
   Multicast DNS Name Resolution .............................................................. 177
   Link Local Multicast Name Resolution ..................................................... 178
Name Resolution in Ad Hoc Networks ......................................................... 178
   Characteristics of Ad Hoc Networks ...................................................... 178
Reactive and Proactive Routing Protocols ..................................................... 179
The Importance of Name Resolution in MANETs ......................................... 181
Architectures for Resolving Host and Service Names in Ad Hoc Networks ... 181

171
As human beings, we prefer to remember the name of a computer. Computers on the other hand, prefer to address each other by numbers, which on the Internet are 32 bits or 128 bits long, depending on the Internet Protocol (IP) used (IPv4 or IPv6). This is one reason why we need a naming service that handles mapping between computer names that we humans find convenient to remember and the network addresses (i.e., numbers) that computers deal with. Another reason is that, according to the Internet model, an IP address does not identify a host, such as a Web server, but a network interface. Although the host makes changes to its network interface or network attachment, it is convenient for the users and applications to allow the name of the host to remain unchanged. As such, keeping different identifiers at different layers helps keep the
protocol layers more independent and also reduces problems associated with layering violations. From a middleware perspective, naming services are a question of keeping higher layer names of entities independent of their lower layer identifiers and their actual locations. Here, the naming service not only helps the users of applications but is also just as much an aid for the software developer. This section familiarizes the reader with the most common naming services used on the Internet and how names can be resolved in mobile wireless ad hoc networks.

**An Architecture for Naming Services**

A fundamental facility in any computer network is the naming service, which is the means by which names are associated with network addresses. Network addresses are found based on their names; for example, to use an electronic mail system the user must provide the name of the recipient to whom the mail is being sent. To access a Web site, the user must provide the universal resource locator (URL) of the site, which again is translated into the network address of the computer hosting the Web site. To give an example, the Domain Name System (DNS) [1,2] maps the host name of the University of Oslo’s public Web server, which is www.uio.no, to the IP address 129.240.4.44. Another example is a Voice-over-IP (VoIP) system that maps a Session Initiation Protocol (SIP) identifier to an E.164 number; for example, the URL sip:dave@my_telecom.com:5060 is translated to +47 904 30 495. The association between a name and the lower layer identifier of a network entity is called a binding. Some naming services, such as DNS, can do reverse mapping (i.e., map an IP address to a corresponding higher layer name) and mapping from one higher layer name to another. In the following section, we will describe a generic model for naming services, which will serve as a reference model for the remaining part of the chapter.

**A Generic Model**

The process of looking up a name in a computer network normally consists of the following steps: First, the binding between higher layer names and lower layer addresses must be registered in the network. This procedure can be referred to as registration. The registration is normally done only once, and the binding is provided through some administrator authority. The bindings are normally registered with a server that holds a binding cache. An example of such an entity is a DNS server. With a strict authentication regime, the registration can be done automatically, as illustrated in message 1 of Figure 8.1. Second, the network entities that desire to resolve a name must be informed of the corresponding binding
that is registered in the network. This procedure can be referred to as *name resolution*. The two main approaches to name resolution are:

- **Push approach** — The bindings are proactively broadcast to all network entities that might have to use the bindings for name resolution some time in the future.
- **Pull approach** — A network entity that desires to resolve a name to a network entity issues a request on demand at the time the binding is needed.

Due to scalability issues, the *pull* model has been chosen for use on the Internet, and this model is the focus of the following sections.

The generic model involves four networking entities for the registration and name resolution procedures:

- A *user entity* (UE) represents the network entity issuing a request to resolve a name.
- A *named entity* (NE) represents the network entity that a name points to.
- A *naming authority* (NA) is authorized to create a binding between names and addresses of named entities (NEs).
- A *Caching Coordinator Entity* (CCE) does intermediate storage of bindings.

**User Entity**

The role of the user entity is to resolve the mapping between a name and the lower layer identifiers of an NE by retrieving a binding from the naming service. The UE may be software components or actual end users who want to look up a specific name. In most cases, the UE will offer a low-level functionality directed toward the system components. In the DNS naming system, requests are issued by the *resolver* in the computer’s operating system. The initiative to activate the resolver can come from an end user typing a Web address in a browser or from an application requiring access to a binding. The request will end up at the entity caching the binding for the name requested, and the binding containing the resolved identifiers will be returned to the resolver. This is illustrated with messages 2 and 3 in Figure 8.1.

**Named Entity**

The named entity is the network entity (e.g. host or computer) identified by a name. For communication, it uses the network interfaces referred to by the lower layer identifiers of a binding.
Naming Authority

The naming authority is the authority or system of authorities permitted to assign names to named entities and bindings between the names and the lower layer identifiers of the NEs. This is normally only configured once but might have to be updated if parameters of the network configuration changes. For example, an Internet Service Provider (ISP) can perform the administrative task of configuring their DNS server to map the network address of a customer’s public Web server to the network address assigned to the customer, or the network administrator of an enterprise network can configure the company’s local DNS server to map a computer’s name to a fixed network address.

Solutions exist that enable a named entity to update its own binding directly with a Caching Coordinator Entity. This requires the CCE to be able to authenticate the NE. With no authentication, it would be possible for someone to insert false information into the caching entity and, in a worst-case scenario, impersonate another network entity by hijacking its binding. The NE must also be authorized to automatically update its binding, thus the node also plays a role in the naming authority.

Caching Coordinator Entity

The primary task of the Caching Coordinator Entity is to act as a cache for name bindings. The CCEs are important for the efficiency of the naming service in the presence of a large number of user entities and named entities. Normally, the NE knows the location of the CCE with which it
Mobile Middleware

is supposed to register its bindings. The UE, which somehow has to retrieve this information, normally does not know the location of the CCE where the binding is located. For the DNS system, the location of the local CCE (i.e., the local DNS server) is normally provided dynamically to the UE by, for example, mechanisms such as the Dynamic Host Control Protocol (DHCP). In this case, the UE normally uses the local CCE to locate and retrieve a binding stored at another CCE on the Internet.

The Domain Name System

The distributed database of the DNS is indexed by domain names. Each domain is basically just a path in an inverted tree referred to as the domain name space. Figure 8.2 illustrates the structure of the domain name space. The practical operation of the DNS system consists of three modules:

- The DNS resolver, which generates DNS requests on behalf of software programs
- The recursive DNS server, which searches through the DNS in response to queries from resolvers and returns answers to those resolvers
- The authoritative DNS server, which responds to queries from recursive DNS servers

The DNS resolver acts as the user entity described earlier, and the DNS servers act as Caching Coordinator Entities. The registration is normally a manual process, and the named entities normally do not take part in the name resolution process (unless they use DNS Secure Dynamic Updates [5]).
Resolving Names Without the Use of DNS Servers

In some situations, it is not feasible to make use of the DNS system to resolve name bindings, and in some cases the DNS system might not even be available. For example, consider a setting where people who happen to meet in an airport lounge want to make use of the wireless local area network (WLAN) feature of their laptops to connect to each other to exchange music or other information they may find interesting (Figure 8.3). Without any DNS service, they would have to identify themselves using the network address, which for human beings is not very appealing. If it was somehow possible to define a separate name space in addition to the DNS system, and if some mechanism could advertise and resolve these names in such a spontaneous setting, users could search and identify names in a more human-friendly way. Multicast DNS [6] and Link Local Multicast Name Resolution [7] are two competing solutions addressing this scenario.

Multicast DNS Name Resolution

Multicast DNS (mDNS) utilizes familiar DNS programming interfaces, packet formats, and operating semantics in a small network where no conventional DNS server has been installed [6]. In short, it enables a node to search for the network address of a computer with named X by sending a multicast DNS message asking: “Does anyone know the network address of node X?” If a node with the name X is present on the network, it will respond by sending back a DNS response containing information about its network address. Multicast DNS is a part of the Mac OS® X operating system, where its implementation is called Rendezvous.
Link Local Multicast Name Resolution

Link Local Multicast Name Resolution (LLMNR) is a peer-to-peer name resolution protocol focused on enabling resolution of names on the local link [7]. LLMNR utilizes the DNS packet format and supports all DNS formats, types, and classes. LLMNR is not intended as a replacement for DNS, and as a result it is only used when a DNS server is either not available or is not providing an answer to a query. LLMNR differs from mDNS in many ways. First, LLMNR is an Internet Engineering Task Force (IETF) standards track specification, while the mDNS that is used in Apple Rendezvous is not. LLMNR is designed for use only on the local link, but mDNS also offers sitewide usage. Furthermore, mDNS sends multicast responses as well as multicast queries.

Name Resolution in Ad Hoc Networks

Mobile ad hoc networking was developed from military research on packet radio networks. In the late 1990s, however, the topic was included as a working group item of the IETF. The goal of the IETF was “to develop a peer-to-peer mobile routing capability in a purely mobile, wireless domain. This capability will exist beyond the fixed network (as supported by traditional IP networking) and beyond the one-hop fringe of the fixed network” [8].

Characteristics of Ad Hoc Networks

A mobile ad hoc network consists of mobile routers,- often simply referred to as nodes. They are free to move about arbitrarily, and wireless technology is used for direct communication between the nodes. Due to the dynamic nature of the wireless media and the arbitrary mobility of the nodes, the network forms a random, multi-hop graph that changes with time. The network is an autonomous system that may operate in isolation, or it may optionally have gateways that connect it as a “stub” network to a fixed network infrastructure. Because a node is not necessarily in direct radio range with any other node in the network, the nodes must participate in the routing process and be willing to forward packets on behalf of other nodes in the network.

An ad hoc network is a network that is created spontaneously, without support from the existing fixed Internet infrastructure. The network might be formed when people equipped with portable PCs come together at conferences, or it can be used to network a user’s personal wireless devices into a personal area network (PAN). Ad hoc networks may also be formed during emergency situations when legacy network infrastructures are
unavailable or damaged. Yet another application of *ad hoc* networking is on a military battlefield where fixed network infrastructures are unavailable or not feasible to use.

The salient characteristics of *ad hoc* networks do not include simply the dynamics of the network topology; in addition, the links are bandwidth constrained and of ever-changing capacity. Furthermore, the nodes often rely on energy-constrained batteries to move about freely, so energy conservation is an important design goal for many *ad hoc* networking technologies. Due to the dynamic networking topology and the fact that nodes might enter and leave the network frequently, it is often also assumed that an *ad hoc* network is without any preexisting infrastructure and that it is difficult to maintain an infrastructure in such a dynamic environment. Because of the lack of a preexisting infrastructure, it is anticipated that direct peer-to-peer communication between nodes will be popular on *ad hoc* networks. This means that any node may in principle operate as a server (e.g., Web server or SIP server) and be contacted directly by other MANET nodes. Any node may also operate as a client and contact other servers available in the network.

A mobile *ad hoc* network that is equipped with IP-based routing is normally referred to as a MANET. The two primary approaches to routing in MANETs are reactive routing and proactive routing. These two approaches are detailed below, with an emphasis on reactive routing.

### Reactive and Proactive Routing Protocols

The routing protocols in mobile *ad hoc* networks can be reactive or proactive. A reactive routing protocol has no prior knowledge of the network topology but finds a route to a given destination on demand. A proactive routing protocol, on the other hand, tries to always have a complete updated picture of the network topology. Reactive routing protocols are normally preferred when nodes are highly mobile, when only a subset of nodes are communicating at any one time, and when communication sessions last for relatively long times. Proactive routing protocols, on the other hand, are preferred for lower levels of mobility and when communication is random and sporadic. Reactive routing is not well known to most people, probably because routing on the fixed Internet is proactive in nature.

A number of reactive routing protocols have been proposed over the years. The most widely studied and popular proposals include the *Ad Hoc* On-Demand Distance Vector (AODV) routing protocol [9] and the Dynamic Source Routing (DSR) routing protocol [10]. Reactive protocols allow source nodes to discover routes to an IP address on demand. Most proposals, including AODV and DSR, work as follows: When a source router requires a route to a destination IP address for which it does not
already have a route, it issues a route request (RREQ) packet. The packet is broadcast by controlled flooding throughout the network, and it sets up a return route to the source (Figure 8.4). If a router receiving the RREQ is either the destination or has a valid route to the destination IP address, it unicasts a route reply (RREP) back to the source along the reverse route. The RREP normally sets up a forward route from source to destination; thus, the pair of RREQ and RREP messages sets up a bidirectional unicast route between source and destination. When the source router receives the RREP, it may begin to forward data packets to the destination. (The acronyms RREQ and RREP are borrowed from AODV.)

Most protocols let routes that are inactive eventually time out. If a link becomes unavailable while the route is active, the routing protocol normally implements an algorithm to repair the route. Often the router upstream of the link breakage would send an error message upstream toward the source. The Ad Hoc On-Demand Distance Vector is a protocol that stores state information in the network. Routers that receive RREQs set up the return routes in the route tables as backward pointers to the source router, and RREPs that are propagated back to the source along the reverse route leave forward pointers to the destination in the route tables. The Dynamic Source Routing protocol, on the other hand, does not rely on routing state in the network; instead, DSR uses source routing. The RREQ collects the IP addresses of all the nodes that it has passed on the way to the destination. The destination subsequently sends a route reply by source routing back to the source of the request, providing it with the source route to the destination.
The Importance of Name Resolution in MANETs

Name resolution is an important feature in an ad hoc network, as addresses may change relatively frequently due to the network dynamics (nodes entering and leaving the network). Furthermore, it is often not possible to use the address as a well-known identifier, because IP addresses for nodes on the MANET will normally be autoconfigured at random, and nodes may also have to change addresses due to addressing conflicts. Devices and resources should instead be identified by stable and unique higher layer names (e.g., fully qualified domain names).

If a MANET is connected to the Internet, a MANET node may use the existing mechanism for name resolution on the Internet (namely, DNS) to look up the IP address of another MANET node; however, in most scenarios the MANET will not always be permanently connected to a fixed infrastructure, and the DNS infrastructure on the Internet might be unavailable. Relying entirely on the DNS on the Internet would not be a robust solution to name resolution in the MANET.

One option would be to introduce a DNS infrastructure into the MANET; however, DNS is designed with a fixed network in mind and has a relatively static, centralized, and hierarchical architecture that does not fit well with MANETs. Without a name resolution method in place, MANET users cannot easily use the applications that are developed for fixed networks for local communication on the MANET. The following text explores name resolutions in ad hoc networks. The focus is primarily on name resolution in reactive MANETs, because name resolution in proactive MANETs might be a less challenging task.

Architectures for Resolving Host and Service Names in Ad Hoc Networks

For name resolution in ad hoc networks, a MANET node may play the same role as discussed earlier for fixed networks. A node may act as a user entity that wants to resolve a name, as a named entity that wants to make its services available to other MANET nodes, or as a Caching Coordinator Entity that holds a central repository for cached bindings and assists other UEs and NEs with name resolution. A binding maps a name to an IP address and possibly a port number that the UE may subsequently use to contact the NE. Three name resolution architectures must be considered for ad hoc networks:

- Distributed architecture
- Coordinator-based architecture
- Hybrid architecture
Distributed Architecture

As shown in Figure 8.5, this architecture contains no CCE. Instead, a user entity floods the name resolution request (NREQ) throughout its surroundings in the network. The flooding can be limited by a flooding scope parameter. Each named entity responds to an NREQ for its own name (i.e., no name caching is allowed) with a unicast name resolution reply (NREP).

Coordinator-Based Architecture

Certain nodes in the MANET are chosen to be Caching Coordinator Entities, a role quite similar to a DNS server. The interactions among user entities, named entities, and Caching Coordinators are illustrated in Figure 8.6. The CCEs announce their presences to the network by periodically flooding CCE announcement messages. The flooding can be limited to a certain number of hops, as determined by the Coordinator announcement scope parameter. Due to the dynamics of ad hoc networks, the NEs must be allowed to register their bindings automatically with the CCE; hence, an NE that receives CCE announcements unicasts name registration messages to register its bindings (i.e., names and associated IP addresses) with CCEs in its surroundings. A UE that has received CCE announcement messages may unicast an NREQ to a selected CCE to discover desired services. The CCE finally responds with a unicast NREP. The selected CCE is often referred to as an affiliated CCE.
Hybrid Architecture

This architecture combines the two architectures described previously. User entities within the Coordinator announcement scope of one or more Caching Coordinator Entities will register their bindings with them; however, they must also be ready to respond to flooded NREQs. When a UE unicasts a NREQ to its affiliated CCE in line with the Coordinator-based architecture (Figure 8.6), the CCE responds with a positive or negative NREP; however, if there is no CCE in the surroundings of the UE or if the affiliated CCE returned a negative NREP, the UE will simply fall back to the distributed architecture (Figure 8.5). Both CCEs and NEs may respond to a flooded NREQ with a positive NREP that matches the requested service.

Intermediate Node Caching

An additional alternative (or a supplement) to the three architectures is to use intermediate node caching. The name resolution may, for example, follow the distributed architecture, but intermediate nodes are allowed to cache bindings found in NREPs that they are forwarding. Later, when receiving a NREP for a cached binding, the intermediate node resolves the name on behalf of the named entity according to the cached binding. The bindings should contain a lifetime value that controls for how long a binding should be kept valid in a cache.
Emerging Principles for Name Resolution in Reactive Ad Hoc Networks

Many ad hoc routing protocols are designed to conserve the scarce networking resources by reducing the need for and negative impact of systemwide flooded broadcasts. Flooded broadcasts exhaust the available bandwidth on the network and reduce the scalability in terms of number of nodes accommodated on the network. Broadcasts also consume the battery power of all networked devices. Reactive routing protocols, such as AODV, are designed to reduce to the greatest extent possible the need for systemwide flooded broadcasts associated with route discovery. Although route discovery is efficient in terms of reducing the number of flooded broadcasts from two to one, name resolution that is not optimized with respect to the route discovery would not work efficiently with reactive routing. The process of contacting a node on the MANET would require two or three broadcasts, as illustrated in the left side of Figure 8.7. Two broadcasts are necessary for the initial name resolution, because the User Entities first have to flood a name resolution request. The reply returned by a node that can resolve the name also requires flooding, because the node does not have a route to the node that issued the request. Finally, the UE will have to flood a regular RREQ to find a route to the resolved IP address.

Alternatively, if the node resolving the name to an IP address is the named entity of the name (and not a node that has cached the name binding), the specification might mandate that the reply be returned by unicast to the user entity. Then, before replying, the node must first flood a RREQ to discover and set up a unicast route to the UE and send the name resolution reply by unicast along this route. It would then be possible to reduce the number of flooded broadcasts from three to two, because the UE already has a route to the resolved IP address as a result of the name resolution process when it contacts the NE. Further details are provided in Engelstad et al. [11,12].

It is possible to reduce the number of flooded broadcast to one, as illustrated in the right side of Figure 8.7. The solution is to use routing messages as carriers for name resolution. First, the user entity floods the name resolution request. By piggybacking the NREQ on a route request packet, a return route to the UE is formed as part of this flooding. By also piggybacking a name resolution reply on a route request packet, the NREP is sent by unicast along the return route to the UE. The RREP also ensures that a forward route is formed as part of this transmission. When the UE finally contacts the service at the IP address of the resolved name, the service request is unicast along the forward route that was put in place by the RREP. In summary, only one flooded broadcast is required in total. The idea of using routing messages as carriers has been proposed for name resolution in Engelstad et al. [11]. In fact, the same mechanism can also
be used for service resolution, as we will see in the discussion on service discovery below.

Needless to say, this broadcast issue is a smaller a problem in proactive \textit{ad hoc} networks, because all unicast communication (including the NREP) can be sent along unicast routes established beforehand by the routing protocol. The broadcasting of the NREQ might benefit from reusing the

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{name_resolution}
\caption{Name resolution without optimization (left) and optimization (right).}
\label{fig:8.7}
\end{figure}
efficient flooding capabilities (e.g., using multi-point relays) that are built-in features of many proactive protocols, including the Optimized Link State Routing (OLSR) protocol [13].

A Proposal for Name Resolution in Reactive Ad Hoc Networks

Overview

A mechanism for name resolution in MANET is proposed in Engelstad et al. [14]. It is mainly targeted at users that can supply their MANET node with a fully qualified domain name (FQDN) from the globally unique DNS name space. The user may have control over some part of the DNS name space or may have received the FQDN from an organization to which the user belongs or subscribes. The proposed name resolution scheme shares characteristics of the Link Local Multicast Name Resolution protocol and the multicast DNS protocol for local-link name resolution, presented earlier. The mechanism proposed for ad hoc networks specifies compressed message formats that allow for bandwidth-efficient name lookups. As an option, it also specifies message formats that reuse the format of DNS messages, which allows for name lookups that are fully compatible with DNS.

Name Resolution Requests and Replies

The proposed scheme uses the distributed architecture presented earlier, with no intermediate node caching. No Caching Coordinators are allowed; instead, only user entities and named entities are present on the MANET. When a name resolution request is broadcast by flooding throughout the MANET, each node with an NE processes the request. By carrying the NREQ as an extension to a route request (Figure 8.8), the number of broadcasts required for name resolution is reduced, as explained earlier in this chapter; thus, a return unicast route to the UE of the request is already in place for a node that wants to respond to the NREQ.

The destination IP address contained in the RREQ, which indicates the address to which a route is sought, is set to a predefined value. This can be a zero address, a broadcast address, or a preassigned multicast address to which no node can cache a route. Intermediate nodes without a valid address mapping for the requested name will not respond to the RREQ part of the message. The NREP is carried as an extension to an RREP message (Figure 8.8). The user entity sending the NREP will normally include its own IP address as the destination IP address in the RREP message to ensure that a forward route is formed. By carrying the response in an RREP message, a responder that is identified by the name that is sought can supply the UE with the resolved IP address in addition to a
unicast route to that IP address. The UE does not have to issue an additional broadcast to discover a route to the resolved address when it subsequently tries to contact that address.

Interaction with External Networks
Mobile ad hoc networks might be connected to external networks through Internet gateways (IGWs). An IGW is a MANET router that also is a host or a router on an external network (with Internet connectivity). The IGW may have access to a conventional DNS server over the external network, and it may also provide other MANET nodes with access to the external network. The scheme proposes to use each IGW as a DNS proxy, as shown in Figure 8.9. The main advantage of using the IGW as a DNS proxy is that there is only one name resolution scheme used on the MANET and that nodes can resolve names in one single process.

Response Selection
A flooded name resolution request might result in reception of multiple name resolution replies. If the named entity present in the MANET has registered its name in the DNS, both the NE and each Internet gateway present in the network may return an NREP. Furthermore, if the NREP
contains a DNS SRV resource record to resolve a non-unique service name (as explained later in this chapter), many NEs present in the MANET may respond to the same NREP. To deal with the possibility of multiple responses, the user entity should wait for some milliseconds to collect responses that might arrive. The proposal includes response selection rules that ensure that a response from an NE present locally on the MANET will always have preference over responses arriving from other nodes, such as IGWs, because a local response might be more reliable and up to date. Furthermore, a direct route through the MANET should normally have preference compared to a route that goes through external networks. If the UE has multiple addresses from which to select after applying this selection rule, it should select (as a secondary selection rule) an IP address to which it has a route that is preferred by the routing protocol. That means that it will normally select an IP address to which it has valid routes and select the IP address that is the fewest hops away from the UE.

Service Discovery

The only information two endpoints communicating over the Internet must know, besides their own configuration, is the network address of the end point with which they are communicating. Any of the two end points might offer a wide range of services (e.g., e-mail, FTP, http), but no defined way exists for the other end point to find out which of these services are being offered. Instead, the users themselves have to remember the name of the computer offering the services and to know in advance the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers associated with the set of services desired or to rely on the well-known port numbers. Would it not be easier if services and the associated IP addresses and port numbers could be searched for and discovered dynamically? This was indeed the case for Macintosh computers.
that used AppleTalk® networking. The Mac user did not require any assistance from a network administrator or even a complicated manual to locate services. Service discovery worked automatically and was operated by using a simple interface.

This section introduces the reader to various service discovery mechanisms that enable the same type of functionality that the AppleTalk protocol offered [15]. Further, the reader is introduced to service discovery mechanisms for an **ad hoc** network where traditional service discovery mechanisms might be deficient and where service discovery is particularly important because the availability of services is dependent on the network dynamics.

**A Generic Model**

Users typically want to accomplish a certain task, not query a list of devices to find out what services are running. It makes far more sense for a client to ask a single question (“What print services are available?”) than to query each available device with a question (“What services are you running?”) and sift through the results looking for printers. This latter approach, which is referred to as **device-centric**, not only is time consuming but also generates a tremendous amount of network traffic, most of it useless. On the other hand, a **service-centric** approach sends a single query that generates only relevant replies. This is what service discovery is all about. The process of discovering services on a computer network is similar to the process of looking up a name described in the previous chapter, but instead of asking “Who has this name?” service discovery asks the question: “Who offers these services?” Normally three entities, or **agents**, are necessary in a service discovery architecture for a computer network:

- A user agent (UA) that represents the network entity issuing a request to find a service
- A service agent (SA) that represents the service offered
- A directory agent (DA) for intermediate storage of information about available services

**User Agent**

The user agent represents the client part in the process of discovering services. The UA may be a software component or an end user who wants to locate a specific service. In most cases, the UA will offer a low-level functionality directed toward system components.
Mobile Middleware

Service Agent

The service agent represents the service in the architecture. This can be the actual service or some entity representing it. Such an entity is called a proxy-SA. An SA will advertise the services it offers by either broadcast or multicast service messages. If a directory agent exists, the SA will try to register with it.

Directory Agent

A directory agent acts as a cache and will merely collect information from the service agents and forward it on demand to user agents. The UAs and SAs can use either passive or active DA discovery to find a DA (Figure 8.10). The service model can be divided into three architectures: distributed, centralized, and hybrid. In the distributed architecture, no DAs are present. The UAs will issue a multicast message to find the SAs. The reply from the SAs can be unicast, or multicast if the UAs have the ability to use and manage a local cache of available SAs. In the centralized architecture, one or more DAs are present.

A variation of the centralized architecture is shown in Figure 8.11, where the UA is retrieving service discovery bindings from the DA according to the pull model. In this example, UAs are also proactively caching announced bindings according to the push model, as illustrated in the left part of the figure. (The push and pull approaches were described in the previous section on name resolution.) In the hybrid approach, a UA will first try to contact a DA according to the centralized architecture and fall back to the distributed approach if no DA can be located.
Current Practice for Service Discovery

When an application on computer A wants to connect with an application on computer B, computer A requires the network address of computer B and the port number of the service. The network address is necessary to route A's request to B. Because B may offer a multitude of services, a port number is used to distinguish between the different services offered at the same network address. Existing practice in IP networks is to go through a three-step process to obtain the network address and the port number of the services. The three steps consists of:

- Mapping the service name to the name of the computer offering the service
- Mapping the port number of the service to a service name
- Resolving the computer name to a network address using DNS or a local name service
Today's IP network has no widely used mechanism to undertake the first step of this process. A common method of advertising services is to map the network address of the computer offering services and the service name with the DNS service. Examples of such services can be a public Web server, which is given the name “www,” (e.g., www.some_domain.com) or a public file transfer server, which is given the name “ftp” (ftp.some_domain.com).

The procedure for mapping port numbers to a service name is pretty simple, and a one-to-one relationship exists between the service name and the port number. The port number is assigned by the Internet Assigned Number Authority (IANA) and is normally maintained in a database on the local computer. The mapping between a service name and a port is typically:

\[ \text{name} \quad \text{<port number>}/\text{<transport protocol>} \quad \text{<aliases>} \]

An example of mapping between a port and a service name is:

\[ \text{http} \rightarrow 80/\text{tcp} \rightarrow \text{www} \rightarrow \text{www-http} \]

The mapping of a network address to the computer offering the service is normally done using address resolution through DNS or through a cache at the local computer, if no access to any name server is available. As can be seen, the procedure for discovering services on the Internet is cumbersome and not very efficient.

**Service Location Protocol**

The Service Location Protocol (SLP) is an emerging Internet standard provided by the IETF for automatic service discovery on the Internet [16]. SLP provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks. Traditionally, in order to locate services on the network, users of network applications have been required to supply the host name or network address of the machine that provides a desired service. Ensuring that users and applications are supplied with the correct information has, in many cases, become an administrative nightmare. SLP was inspired by the AppleTalk® protocol [15], a mechanism created by Apple that proved to be a huge success due to the simplicity and benefits of the solution. The only drawback was that it did not scale very well. The main focus of SLP is to be a mechanism that acts as an enabler for plug-and-play functionality in IP networks with automatic and dynamic bindings between services and service users. The SLP protocol introduces three major components into the network:
Name Resolution and Service Discovery

- **User agent (UA)** — The SLP user agent is a software entity that is looking for the location of one or more services. This search is usually implemented (at least partially) as a library to which client applications link, and it provides client applications with a simple interface for accessing SLP registered service information.

- **Service agent (SA)** — The SLP service agent is a software entity that advertises the location of one or more services. SLP advertisement is designed to be both scalable and effective, minimizing the use of network bandwidth through the use of targeted multicast messages and unicast responses to queries.

- **Directory agent (DA)** — The SLP directory agent is a software entity that acts as a centralized repository for service location information. In a large network with many UAs and SAs, the amount of multicast traffic involved in service discovery can become so large that network performance degrades. By deploying one or more DAs, both SAs and UAs make it a priority to discover available DAs, as the use of a DA minimizes the amount of multicast messages sent by the protocol on the network. The only SLP-registered multicast in a network with DAs is for active and passive DA discovery.

The Service Location Protocol introduces dynamic naming services without the need for any centralized name server or other agents. Because SLP uses IP multicast for this purpose, it requires the cooperation of IP routers that implement IP multicast. IP multicast is used for such features as IP-based audio and video broadcasting and video conferencing, but IP multicasting may not be completely implemented across some intranets. In the absence of IP multicasting, SLP name lookups will only work within the subnet on which they are performed or within the groups of subnets over which IP multicast is supported.

The Service Location Protocol suffers from a lack of implementation support, because individual companies, such as Apple and Microsoft, are pushing different service discovery technologies. Generally, standardization is a good thing but not very useful if no one provides implementation support for the standards. The SLP has proved to be useful, especially for UNIX® variants, and an open-source project exists to support service discovery on operating systems such as Linux and BSD. (An implementation of SLP can be downloaded from www.openslp.org.)

**DNS Service Resource Records**

An alternative to building up a new SLP infrastructure on the Internet is to reuse the existing DNS infrastructure and allow for service discovery as an extension to DNS. Extensions to DNS are enabled through the use
of DNS resource records (DNS RRs). The most common resource record for service location is the DNS SRV resource record [17]. DNS SRV was originally designed to locate services on the global Internet. As an example, let us assume that company_A has implemented the use of SRV in its DNS server. Entries for the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP) would look like this in the zone file for company_A:

\[
\text{\_http\_.tcp\_.www\_.company\_A\_.com.}
\]

For example, to locate an HTTP server that supports TCP and provides Web service, it does a lookup for:

\[
\text{\_http\_.tcp\_.www\_.company\_A\_.com.}
\]

If the use of SRV had been widely deployed, a DNS server would have answered with a list of Web servers that satisfied the searching criteria.

With no existing directory agent, this mechanism depends solely on the DNS system. Critics claim that the deployment of it puts an extra burden on an already overloaded DNS system. Furthermore, DNS SRV only allows for simple service name resolution and has little support for the type of service attribute negotiation that is accommodated by SLP.

**XML Web Services/ UDDI**

An eXtensible Markup Language (XML) Web service is a service that accommodates direct interaction using XML-based messaging (such as Simple Object Access Protocol, or SOAP [18]) over Internet-based protocols, such as HTTP. The interfaces and bindings of the Web service are defined, described, and discovered by XML [19]. In addition to being able to describe and invoke a Web service, publication of and discovery of Web services should also be accommodated. The Universal Description, Discovery, and Integration (UDDI) specification [20] is commonly accepted to be the standard mechanism to handle this. UDDI registries provide a publishing interface to allow for creation and deletion of entries in the
registry and an inquiring interface to search for entries in the registry by different search criteria. The interfaces are invoked by SOAP messages, and as such UDDI itself can be thought of as an XML Web service. Each entry in the UDDI registry contains three parts:

- The white pages contain business information.
- The yellow pages contain the service a business provides.
- The green pages contain the specific services provided and technical information sufficient for a programmer to write an application that makes use of the service.

Web services are described by XML using the Web Services Description Language (WSDL) specification [21].

**Other Service Discovery Protocols**

The Salutation protocol [22] released by the Salutation Consortium in 1996 predates SLP. It introduces the same concept as a directory agent, referred to as the Salutation Manager (SLM). In the Salutation protocol, user agents and service agents are referred to as clients and servers, respectively. The Salutation Manager Protocol (SMP) and Cisco’s Transport Manager™ are used for the actual communication, utilizing remote procedure calls (RPCs). SLM will also assist in establishing a session pipe over which a service access between the client and the server can occur.

Jini™ [23] is another technology for service discovery that runs on top of Java. It allows clients to join a Jini lookup service, which maintains dynamic information about services in the network. The client can use it for simple service discovery by requesting information about a particular device. An attractive feature of Jini is that it also allows clients to download Java code from the lookup service which is used to access the service; however, it requires that the server has already uploaded the Java proxy that the client downloads from the lookup service. Jini also supports the concepts of federations, where groups of devices may register with each other to make their services available within the group.

**Service Discovery on Link Local Networks**

**Simple Service Discovery Protocol**

The Simple Service Discovery Protocol (SSDP) [24] is a part of Microsoft’s Universal Plug and Play (UpnP™) [25] and provides a mechanism that network clients can use to discover network services. UPnP supports self-configuration networks by enabling the ability to automatically acquire an
IP address, announce a name, learn about the existence and capabilities of other elements in the network, and inform others about their own capabilities.

The UPnP protocols are based on open Internet-based communications standards. UPnP is based on IP, TCP/UDP, http, and XML. The SSDP protocol specifies the use of multicast of UDP/HTTP for announcements of services. The content of the service announcements are described using XML. Hypertext Transfer Protocol Unicast (HTTPU) and Hypertext Transfer Protocol Multicast (HTTMU) are used by SSDP to generate requests over unicast and multicast. The SSDP architecture introduces three entities into the network:

- **SSDP service** — The SSDP service is a service agent and represents the individual resources in an SSDP-enabled network. The agent is defined in two versions, depending on whether or not an SSDP proxy is available in the network. A SSDP service without proxy support is a simple service where all messages are sent on an SSDP reserved multicast group.

- **SSDP client** — The SSDP client is a user agent. Initially, the SSDP client will search for a proxy, followed by a search for other relevant resources. If an SSDP proxy is available, all requests are done using unicast. If it is not, the SSDP searches for services using multicast. The SSDP client will cache all information about services and uses a time stamp to manage the accuracy of the cache.

- **SSDP proxy** — The SSDP proxy is a directory agent that gathers information about available resources in the network. The proxy can be viewed as a regular resource with responsibility to cache and manage all service information. An SSDP proxy is not a mandatory element in an SSDP-enabled network, but it does improve the scalability when deployed in large networks.

**Multicast DNS**

Domain Name System service discovery is a way of using standard DNS programming interfaces, servers, and packet formats to browse the network for services. As shown earlier, multicast DNS (mDNS) [26] can be used to resolve names without the use of any DNS server. The same multicast mechanism can be used to search for services, by requesting a binding for the type of service wanted instead of requesting a binding for a name. This is illustrated in Figure 8.12.

When a mDNS query is sent out for a given service type and domain, any matching services reply with their names. The result is a list of available services from which to choose. For example, an application that is searching
for a printer that supports TCP and is located in the company_A domain would issue a query for:

_lpr._tcp.company_A.com

Then, every printer attached to the LAN will answer with information about its services, such as color and pages per minute. Using a mechanism for automatic service discovery greatly simplifies the job of connecting PCs, terminals, wireless units, and consumer electronics.

The caching of multicast packets can prevent hosts from requesting information that has already been requested; for example, when one host requests, say, a list of LPR print spoolers, the list of printers comes back via multicast so all local hosts can see it. The next time a host requires a list of print spoolers, it already has the list in its cache and does not have to reissue the query.

**Service Discovery in Ad Hoc Networks**

**Service Discovery Mechanism for MANETs**

Discovery of services and other named resources is an important feature for the usability of mobile ad hoc networks. The characteristics of ad hoc networks were described earlier. We recall that a MANET is anticipated to be without any preexisting infrastructure and that nodes may enter or leave the network at any time. This makes efficient and timely service discovery a challenging task. In a MANET, any node may in principle operate as a server and provide its services to other MANET nodes. Any node may also operate as a client and use the service discovery protocol to detect available services in the network. The service attributes include service characteristics and service binding information, such as IP addresses, port numbers and protocols, that allow the client to initiate the selected service on the network.
appropriate server. Existing service discovery mechanisms, described in previous sections, are designed with a fixed network in mind and might not fit well with MANETs. Before a service discovery mechanism for ad hoc networks can be designed, we must determine the necessary principles for service discovery in ad hoc networks and evaluate which service discovery architecture that is most suitable. The reader will observe that name resolution and service discovery have many similar features in terms of both discovery principles and possible architectures.

**Service Location Architectures for Service Discovery on MANETs**

The architectures available for name resolution, as described for name resolution above, are also available for service discovery. In the context of service discovery, the Caching Coordinator Entity is normally referred to as a service coordinator (SC), the user entity is referred to as a user agent (UA), and the named entity (NE) is referred to as a service agent (SA). Furthermore, the architectures are often referred to as distributed, service-coordinator-based, and hybrid service location architectures. The distributed architecture for MANETs is illustrated in Figure 8.13, and the service-coordinator-based architecture is shown in Figure 8.14. The hybrid architecture is a combination of the two: The UA tries to discover services according to the service-coordinator-based architecture but falls back to the distributed approach if the selected SC does not have the desired binding or if no SC can be found.
Emerging Principles for Service Discovery on Reactively Routed MANETs

The emerging principles for name resolution in reactive MANETs, in which resolution requests and replies are carried by routing messages (outlined earlier), are also useful for service discovery. In the context of service discovery, service discovery requests (SREQs) and service discovery replies (SREPs) are piggybacked on RREQ and RREP packets, respectively (Figure 8.15). The advantages of piggybacking service discovery on routing messages include:

- Reverse routes to the user agent (i.e., client) are established along with the SREQ so no additional route discovery is necessary to relay the SREP back to the requestor.
- Forward routes to the SC are established along with the SC announcements so SREQs and service registrations can be unicast to the SC.
- A forward route is established along with the SREP so no additional route discovery is necessary for further communication with the node issuing the reply.

Figure 8.15 shows how service discovery can be streamlined with the reactive routing protocol. Service Discovery Requests are piggybacked on routing request packets, and service discovery replies are piggybacked on routing reply packets. In addition, for the hybrid architecture, the SC
announcements are piggybacked on RREQ packets, and service registrations are piggybacked on RREP packets. Thus, both the SC-based, hybrid and distributed architectures can take advantage of this procedure.

**Proposed Solution for Service Discovery in Reactive Ad Hoc Networks**

A solution for service discovery in reactive *ad hoc* networks has been proposed in Koodli and Perkins [27]. It uses basically the same mechanism as was presented for name resolution earlier in the chapter. It is based on the distributed architecture without the use of any service coordinators, but here the intermediate nodes are allowed to cache service bindings and respond immediately if a valid binding is found. It also uses the same technique to carry the discovery messages by the routing packets to allow both services and the routes to the nodes providing these services to be discovered in one round-trip. Koodli and Perkins [27] defined a *service binding* as a mapping of a service name to an IP address. Different encoding schemes, such as a service port request or service URL, can be used to request a binding for an IP address. A service discovery request for a service URL contains a service-type string and a service request predicate of formats that are defined by SLP. The format of the URL and the authentication block contained in the corresponding service discovery reply are also defined by SLP; hence, not only are formats of SLP reused but the authorization block also ensures that the service authorization features of SLP are maintained. The use of SREQs for service ports assumes that the user agents know in advance the well-defined (TCP or UDP) port number associated with the requested service; hence, the SREQ only has to contain the port number associated with the service application requested. The proposed service discovery protocol considers the case where the UA has neither a service binding nor an active route to a node providing the desired service. It also considers the case where the route is active but the service binding has expired (or is absent) and the case where the service binding is active but the route has expired.

![Figure 8.15 Routing packets carry service discovery messages.](image-url)
Evaluation of Service Location Architectures in Ad Hoc Networks

As a slight simplification, one could say that all of the service discovery protocols presented earlier are based on two baseline mechanisms for the management of service discovery information:

- Information about services offered on the network is stored on one or a few centralized nodes, referred to as service coordinators (SCs) in this chapter.
- Information about each service is stored on each node that is offering the service.

In previous sections we defined the service discovery architectures according to the two mechanisms above. A solution that is based only on the first mechanism is a service-coordinator-based architecture, and a solution based only on the second mechanism is a distributed architecture. Finally, a solution based on a mixture of both the first and the second mechanisms is a hybrid architecture. In the next section, we evaluate the performance of the hybrid and distributed architectures in a reactively routed MANET. The architectures have been presented in detail earlier in the chapter.

Architecture Evaluation

The evaluation of the distributed and hybrid architectures is based on results from Engelstad et al. [28]. The architectures can be configured by different settings of the following two parameters:

- **Flooding scope** — This parameter determines the maximum number of hops a flooded service discovery request is allowed to traverse in the network (e.g., the flooding scope is four hops in Figure 8.13 and two hops in Figure 8.14).
- **SC announcement scope** — This parameter determines the maximum number of hops a flooded SC announcement is allowed to traverse in the network (e.g., Figure 8.14 illustrates a situation with an SC announcement scope of two hops). This parameter is used only in the hybrid architecture. Alternatively, the distributed architecture can be considered as a special case of a hybrid architecture where the SC announcement scope is set to zero.

The objective is to optimize the benefits of additional service availability provided by the use of service coordinators against the cost of additional overhead and possibly higher delay. We will consider the performance measured by delay, by the service availability, and by the message overhead.
Engelstad et al. [28] observed that the differences in delays between the two architectures are only on the order of a few milliseconds. Because service discovery is normally part of the service initiation, users would normally accept an initial delay (e.g., when retrieving a Web page on the Internet or for setting up an IP telephony call); hence, it was concluded that the small observed differences in delay between the two architectures should be considered negligible in this context. With delay out of the picture, the key question is reduced to whether the increased service availability is worth the increase in message overhead. The service availability can be defined as [29]:

\[
\text{Service availability} = \frac{\text{Number of positive service replies}}{\text{Total number of service requests generated}}
\]

A positive service discovery reply indicates successful contact with this server via the given access information (i.e., a route to the resolved server can be found).

It is observed in Figure 8.16 that the service availability is indeed higher with the Hybrid approach. Figure 8.16 also shows how the presence of service coordinators (i.e., for the hybrid architecture) influences the service availability. When comparing architectures that use the same flooding scope, we find that the hybrid architecture improves the service availability as compared to the distributed query-based architecture. The main reason why SCs improve the service availability is that in some cases the SC will be positioned in between the user agent and service agent. We may return to Figure 8.14 for an example of such a situation. Here, the UA and SA are four hops apart, but both are only two hops away from the SC. The SC announcement and flooding scopes are both of two hops; hence, the SA is able to register its services with the SC, and the UA is able to discover the server by means of the SC. However, because the UA is four hops away from the SA, the UA would not be able to discover the SA if the distributed architecture with a flooding scope of two hops had been used.

From Figure 8.16 we observe that, with SC announcement scopes of one or two hops, the service availability is improved by 8.7% or 20.9%, respectively, at a server density of 5%. Because the introduction of SCs improves the service availability, it comes as no surprise that the service availability increases with an increasing SC announcement scope.

Although the service coordinators introduced in the hybrid architecture yield higher service availability, they also result in extra message overhead, as observed in Figure 8.17. The SCs introduce two proactive elements to the network — namely, SC announcements and service registrations. These messages will take up a fixed bandwidth regardless of whether or not there are clients doing service discoveries. In addition, these two types of
messages will also trigger pure route discovery messages when a reactive routing protocol is being used. By comparing Figure 8.16 and Figure 8.17, we observe that the additional cost of using SC in terms of percentage increase in message overhead is much higher than the additional benefits provided in terms of percentage increase in service availability.

A more rigorous analysis that compares the two architectures are undertaken in Engelstad et al. [28]. It takes into consideration a large range of control parameters, such as server density, service coordinator density, flooding scopes, SC announcement scopes, reasonable request frequencies, number of different types of services, level of mobility, and so forth. It is also argued that the conclusion is valid independent of the lengths of the service discovery messages.

In Engelstad et al. [28], it is generally observed that for any hybrid configuration with a given service coordinator announcement scope and flooding scope it is always possible to find a distributed configuration (with some flooding scope) that outperforms the hybrid configuration in terms of both higher service availability and lower messaging overhead. As the opposite is not the case, it is concluded that the distributed architecture outperforms the hybrid architecture. Service discovery protocols that are using SCs (or functionality similar to directory agents) do not
work well in *ad hoc* networks with reactive routing. The main reason is that the increase in service availability by adding SCs is negligible compared to the extra message overhead it causes.

In addition to the analyses presented in Engelstad et al. [28], several other arguments would suggest against introducing service coordinators in MANETs at large. First, the distributed architecture is considerably less complex than the hybrid architecture. Furthermore, in a dynamic topology with network entries and departures, the service coordinators of the hybrid architecture have the disadvantage of sometimes providing the user agents with “false positives” (i.e., with outdated bindings of servers that have already left the network). Moreover, the hybrid approach may call for a separate complicated mechanism for electing SCs, which might require a substantial amount of network resources.

**References**


