

IP-based resource discovery in fixed and mobile networks

PAAL E. ENGELSTAD AND GEIR EGELAND



Paal E. Engelstad
is Research
Scientist in
Telenor R&D



Geir Egeland
is Research
Scientist in
Telenor R&D

This article introduces the most common IP-based naming services and service discovery mechanisms. The reader is made familiar with the most common naming services used on the Internet and how names can be resolved in mobile wireless ad hoc networks. 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 since the availability of services is dependent on the network dynamics.

1 Name resolution

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 is 32 bits or 128 bits long, depending on whether IPv4 or IPv6 is used. This is one reason why we need a naming service that can handle mapping between computer names, which we humans find convenient to remember, and between network addresses (i.e. numbers), which 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 that the name of the host remains unchanged. As such, keeping different identifiers at different layers helps keep the protocol layers more independent and also reduces problems associated with layering violations.

In a middleware perspective, naming services is also a question of keeping higher layer names of entities independent of their lower layer identifiers and their actual locations. Here, the naming service is not only a way of helping the users of applications; it is just as much a help for the software developer.

In this section, the reader is made familiar with the most common naming services used on the Internet and how names can be resolved in mobile wireless ad hoc networks.

1.1 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, and network addresses are found based on their names. For example, when you use an electronic mail system, you must provide the name of the recipient to whom you want to send mail. If you want to access a web site you must provide its URL, which again is translated into the network address of the computer hosting the web site.

To give some examples, 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 can be a VoIP system that maps a SIP identifier to an E.164 number, e.g. URI `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, also have the possibility to do reverse mapping, e.g. to map an IP address to a corresponding higher layer name, and to do mapping from one higher layer name to another. Some naming services, such as DNS, also have the possibility to do reverse mapping, e.g. to map an IP address to a corresponding higher layer name, and to do 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 this article.

1.1.1 A generic model

The process of looking up a name in a computer network consists normally 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 1-1.

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*. There are two different principle approaches to name resolution, namely *push* and *pull*:

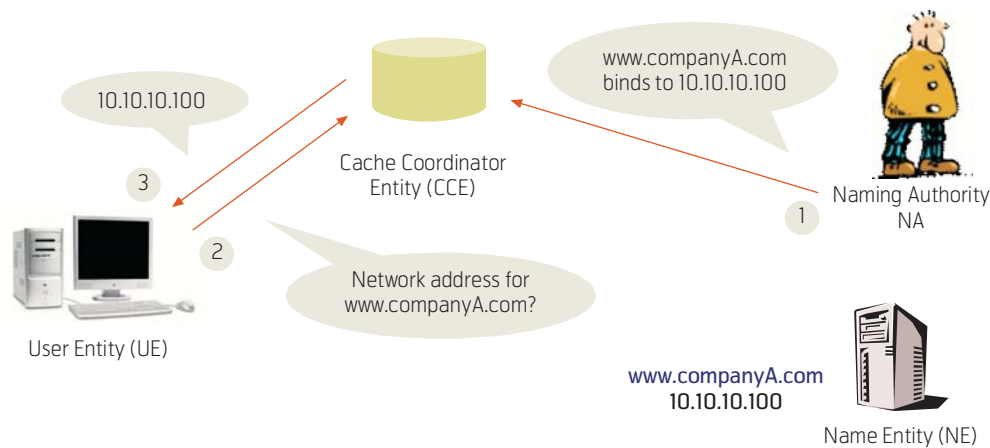


Figure 1-1 A Cache Coordinator Entity (e.g. the enterprise DNS server) is updated with a new binding for its Named Entity (e.g. a public web server)

- *Push approach:* The bindings are proactively broadcast to all network entities that might need 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 the Internet, and this model will be our focus in the following sections.

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

These are:

- 1 A *User Entity* (UE) represents the network entity issuing a request to resolve a name.
- 2 A *Named Entity* (NE) represents the network entity that a name points to.
- 3 A *Naming Authority* (NA) is authorized to create a binding between names and addresses of Named Entities (NE).
- 4 A *Caching Coordinator Entity* (CCE) does intermediate storage of bindings.

User Entity (UE)

The role of the *User Entity* (UE) 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 that want to look up a specific name. In most cases the UE will offer a low level functionality directed towards 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 needing to access 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 by messages 2 and 3 in Figure 1-1.

Named Entity (NE)

The *Named Entity* (NE) 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 (NA)

The *Naming Authority* (NA) is the authority or system of authorities permitted to assign names to NEs and bindings between the names and the lower layer identifiers of the NE. This is normally only configured once, but might require to be updated if parameters of the network configuration changes. For example, an 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 will configure the company's local DNS-server to map a computer's name to a fixed network address.

There exist solutions that enable an NE to update its own binding directly with a Caching Coordinator Entity (described below). This requires that the Caching Coordinator Entity can authenticate the NE. With no authentication, it would be possible for someone to insert false information into the caching entity, and in the worst case impersonate another network entity by hijacking its binding. The NE must

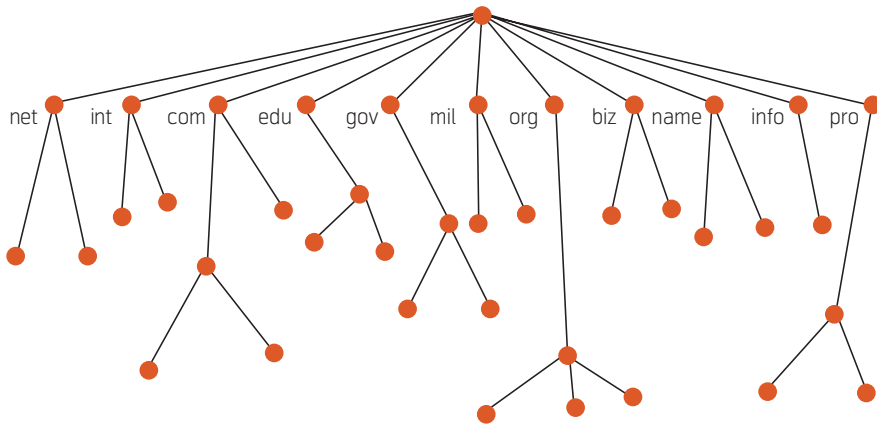


Figure 1-2 The structure of the DNS name space

also be authorized to automatic updates to its binding, and the node thus also has the role of an NA.

Caching Coordinator Entity (CCE)

The primary task of the *Caching Coordinator Entity* (CCE) 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 UEs and NEs. Normally the NE knows the location of the CCE it is supposed to register its bindings with. The UE, which somehow has to retrieve this information, does not normally 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 DHCP. Here, the UE normally uses the local CCE to be able to locate and retrieve a binding stored at another CCE on the Internet.

1.1.2 The Domain Name System

The DNS' distributed database is indexed by domain names. Each domain is basically just a path in an inverted tree, called the *domain name space*. Figure 1-2 illustrates the structure of the domain name space.

The practical operation of the DNS system consists of three modules:

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

The DNS resolver acts as the UE described above, while the DNS servers act as CCEs. 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]).

1.2 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 spontaneous setting where people in some airport lounge want to make use of their laptop's WLAN feature and connect to each other to exchange music or other information they may find interesting (Figure 1-3). Without any DNS service, they will have to identify themselves using the network address, which for human beings is not very attractive.

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*

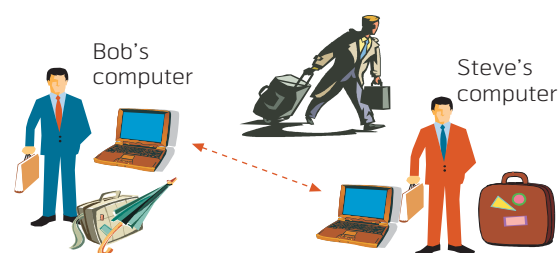


Figure 1-3 Computers communicating without a DNS server

tion [7] are two competing solutions addressing this scenario. These will be described in the following.

1.2.1 Multicast DNS name resolution

Multicast DNS (mDNS) is a way of using familiar DNS programming interfaces, packet formats and operating semantics in a small network where no conventional DNS server has been installed [6]. In short, the mechanism works by enabling a node to search for the network address of the computer with the name *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 returning 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*.

1.2.2 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 IETF standards track specification, while mDNS, which is used in Apple Rendezvous, is not. LLMNR is designed for use only on the local link, while mDNS also offers site-wide usage. Furthermore, mDNS sends multicast responses as well as multicast queries.

1.3 Name resolution in ad hoc networks

Mobile ad hoc networking started out from military research on Packet Radio Networks. In the late 1990s, however, the issue was included as a working group item of the Internet Engineering Task Force (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].

1.3.1 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. Since 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 their 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 in places where fixed network infrastructures are unavailable or not feasible to use.

The salient characteristics of ad hoc networks include not only 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, and 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 pre-existing infrastructure and that it is difficult to maintain an infrastructure in such a dynamic environment.

Due to the lack of pre-existing 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”. There are two principal different approaches to routing in MANETs, namely reactive and proactive routing. These two approaches will be detailed below, however, with most emphasize on reactive routing.

1.3.2 Reactive and proactive routing protocols

The routing protocols in mobile ad hoc networks can be divided into two different approaches:

- Reactive
- 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 a relatively long time. Proactive routing protocols, on the contrary, 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 by nature. As a consequence, we will summarize the features of reactive routing in the following:

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 [9]) routing protocol and the Dynamic Source Routing (DSR [10]) routing protocol.

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 sets up a return route to the source (Figure 1-4).

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 set up a bi-directional unicast route between source and destination. Once 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 to the link breakage would send an error message upstream towards the source.

AODV is a protocol that stores state information in the network. Routers that receive RREQs set up the return routes in the route tables as backwards pointers to the source router, while RREPs that are propagated back to the source along the reverse route leave for-

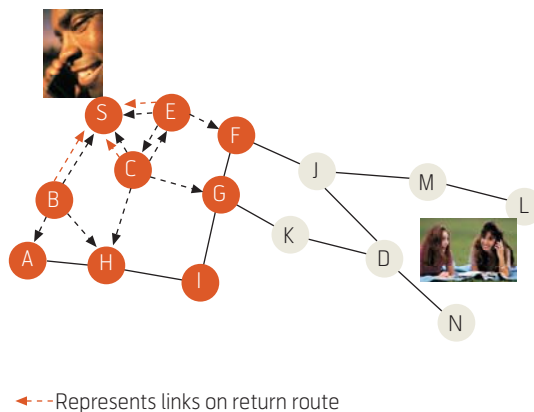


Figure 1-4 Route Requests (RREQs) in AODV

ward pointers to the destination in the route tables. The Dynamic Source Routing (DSR) 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 by source-routing a Route Reply back to the source of the request, providing it with the source route to the destination.

1.3.3 The importance of name resolution in MANETs

Name resolution is an important feature in an ad hoc network, since addresses may change relatively frequently due to the network dynamics, e.g. when nodes enter and leave the network. Furthermore, it is often impossible to use the address as a well-known identifier, since IP addresses for nodes on the MANET will normally be auto-configured at random, and nodes may also need 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 to 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. In the following, solutions to name resolutions in ad hoc networks will be explored. The focus is mainly on name resolution in reactive MANETs, because name resolution in proactive MANETs might be a less challenging task.

1.3.4 Architectures for resolving host and service names in ad hoc networks

For name resolution in ad hoc networks, a MANET node may hold the same role as presented above for fixed networks. A node may act as a User Entity (UE) that wants to resolve a name, a Named Entity (NE) that wants to make its services available to other MANET nodes, and/or a Caching Coordinator Entity (CCE) 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(es) and possibly a port number(s) that the UE may subsequently use to contact the NE.

There are three name resolution architectures that need to be considered for ad hoc networks:

- 1 *Distributed architecture*
- 2 *Coordinator based architecture*
- 3 *Hybrid architecture*

These name resolution architectures will be described in the following.

Distributed architecture

As shown in Figure 1-5, this architecture contains no CCE. Instead, a UE floods the *Name Resolution Request (NREQ)* throughout its surroundings in the network (1). The flooding can be limited by a *flood-*

ing scope parameter. Each NE responds to a *NREQ* for its own names (i.e. no name caching is allowed) with a unicast *Name Resolution Reply (NREP)* (2).

Coordinator based architecture

Certain nodes in the MANET are chosen to be Caching Coordinators (CCEs), a role quite similar to a DNS server. The interaction between User Entities, Named Entities and Caching Coordinators are illustrated in Figure 1-6. The CCEs announce their presences to the network by periodically flooding CCE Announcement messages (1). The flooding can be limited to a certain number of hops, determined by the *Coordinator announcement scope* parameter. Due to the dynamics of ad hoc networks, the Named Entities must be allowed to register their bindings automatically with the CCE. Hence, a Named Entity that receives CCE Announcements unicasts *Name Registration* messages to register its bindings (i.e. names and associated IP addresses) with CCEs in its surroundings (2). A User Entity that has received CCE Announcement messages may unicast a *Name Resolution Request (NREQ)* to a selected CCE to discover desired services (3). The CCE finally responds with a unicast *Name Resolution Reply (NREP)* (4). The selected CCE is often referred to as an affiliated CCE.

Hybrid architecture

This architecture combines the two architectures described in previous sections. UEs within the Coordinator announcement scope of one or more CCEs will register their bindings with them. They must however also be ready to respond to flooded NREQs. When a UE unicasts an NREQ to its affiliated CCE in line with the *Coordinator based architecture* (Figure 1-6), the CCE responds with a positive or negative NREP. However, if there is no CCE in the UE's sur-

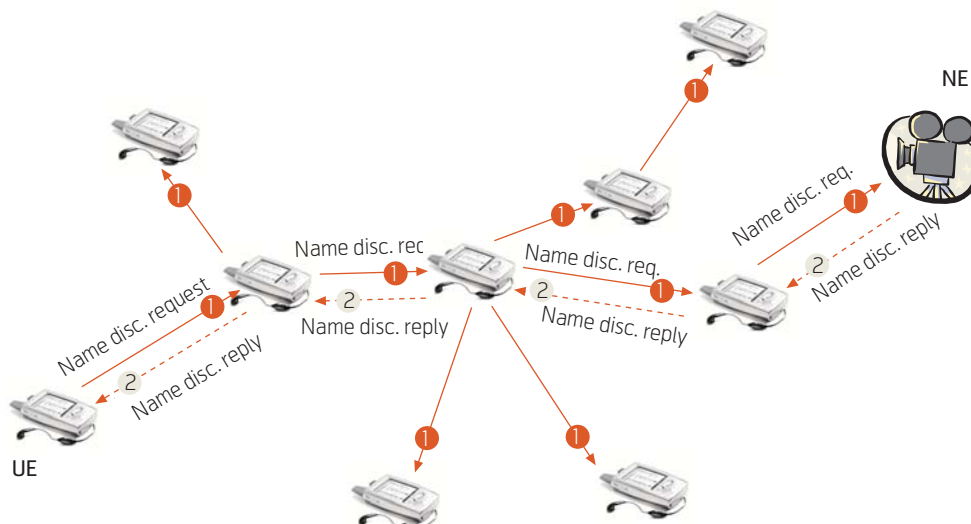


Figure 1-5 The Distributed architecture with User Entities (UE) and Named Entities (NE)

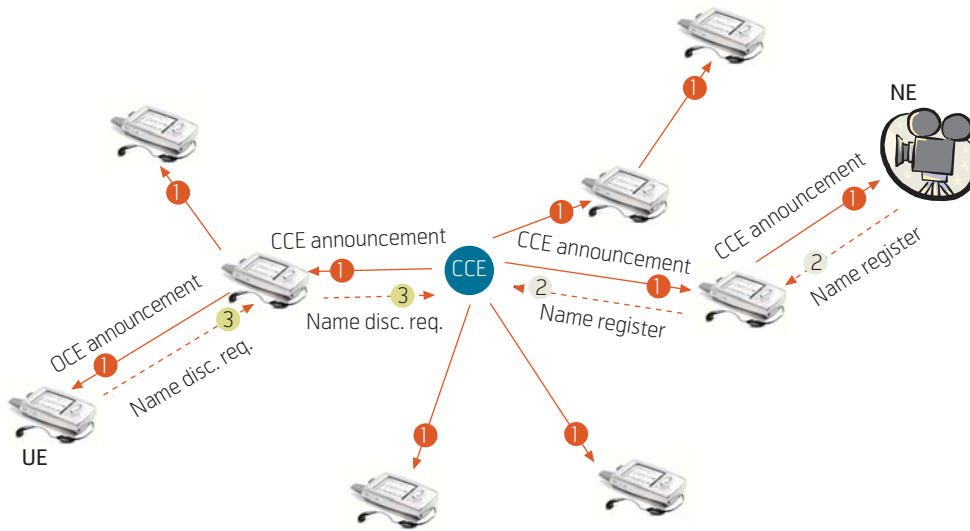


Figure 1-6 The Coordinator based architecture with User Entities (UE), Caching Coordinator Entities (CCEs) and Named Entities (NE)

roundings or if the affiliated CCE returned a negative NREP, the UE will simply fall back to the *Distributed architecture* (Figure 1-5). Both CCEs and Named Entities 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 an NREP for a cached binding, the intermediate node resolves the name on behalf of the NE 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.

1.3.5 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 the negative impact of system-wide flooded broadcasts. Flooded broadcasts not only exhaust the available bandwidth on the network and reduce the scalability in terms of number of nodes accommodated on the network. Broadcasts also consume battery power of all networked devices.

Reactive routing protocols, such as AODV, are designed to reduce to the furthest extent the need for system-wide 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 optimised 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 1-7. Two broadcasts are necessary for the initial name resolution, because the User Entity first has to flood a name resolution request. The reply returned by a node that can resolve the name also requires flooding, since the node does not have a route to the node that issued the request. Finally, the User Entity 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 an RREQ to discover and set up a unicast route to the User Entity 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 User Entity already has a route to the resolved IP address as a result of the name resolution process when it contacts the Named Entity. Further details are provided in [11][12].

It is however possible to reduce the number of flooded broadcasts to one, as illustrated in the right side of Figure 1-7. The solution is to use routing messages as carriers for name resolution. First the User Entity floods the name resolution request (NREQ). By piggybacking the NREQ on a route request (RREQ) packet, a “return route” to the User Entity would be formed as part of this flooding. By also piggybacking a name resolution reply (NREP) on a route request (RREQ) packet, the NREP is sent by unicast along the “return route” to the User Entity. The

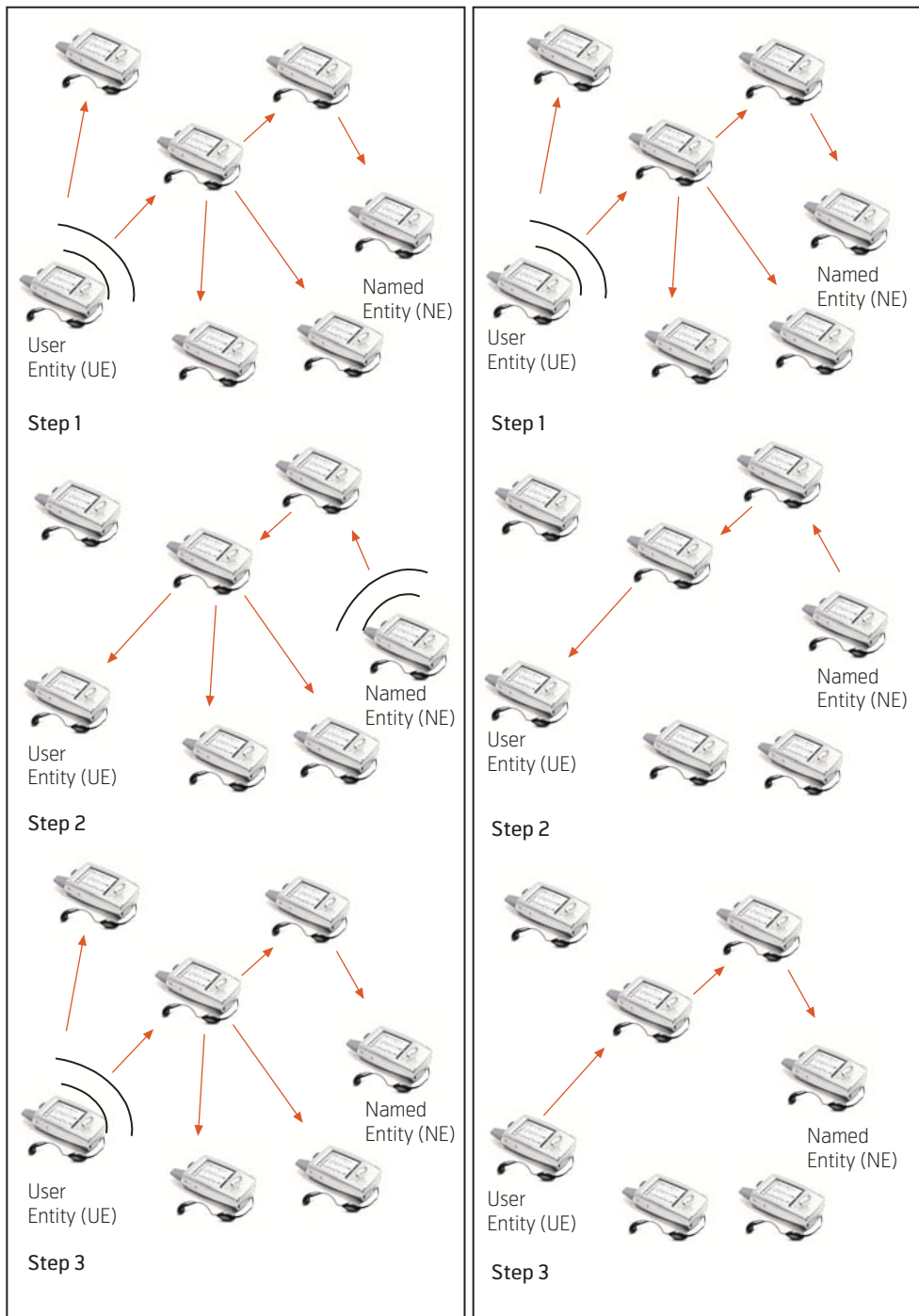


Figure 1-7 Name resolution without optimization (left) and with optimization (right)

RREP also ensures that a “forward route” is formed as part of this transmission. When the User Entity 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 [11]. In fact, the same mechanism can also be used for service resolution, as we will see in the next part on service discovery below.

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

1.3.6 A proposal for name resolution in reactive ad hoc networks

Overview

A mechanism for name resolution in MANET is proposed in [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 that they belong to or subscribe to. The proposed name resolution scheme shares similarities to the Link-Local Multicast Name Resolution (LLMNR) protocol and multicast DNS protocol for local-link name resolution presented above. The mechanism proposed for ad hoc networks specifies compressed message formats allowing for bandwidth-efficient name lookups. As an option, it also specifies message formats that reuse the format of DNS messages, allowing for name lookups that are fully compatible with DNS.

Name Resolution Requests and Replies

The proposed scheme uses the distributed architecture presented above, with no intermediate node caching. No Caching Coordinators are allowed, and instead, only User Entities and Naming Entities are present on the MANET. When an NREQ is broadcast by flooding throughout the MANET, each node with a Named Entity processes the request. By carrying the NREQ as an extension to an RREQ (Figure 1-8), the number of broadcasts required for name resolution is reduced as explained earlier in this article. Hence, a return unicast route to the User Entity 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, indicating the address to which a route is searched for, is set to a pre-defined value. This can be a zero address, a broadcast address or a pre-assigned multicast address to which no node can cache a route. Hence, intermediate nodes without a valid address mapping for the requested name will not respond to the RREQ part of the message.

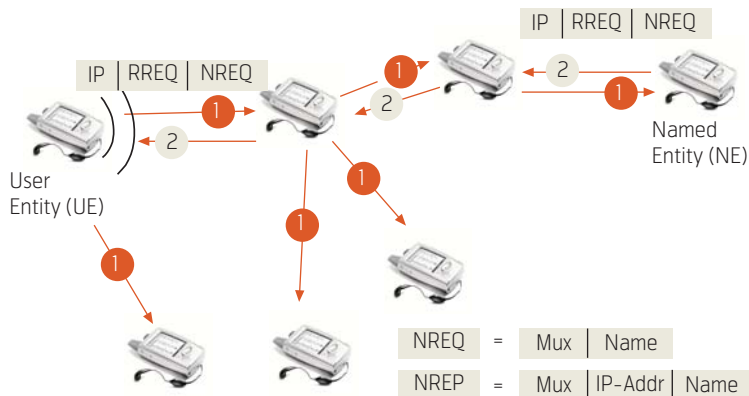


Figure 1-8 A Name Resolver (NR) floods a Name Resolution Request (NREQ), carried by a Route Request (RREQ) header, throughout the network (1). A Name Server (NS) processes with the requested name-to-address mapping unicasts a Name Resolution Reply (NREP), carried by a Route Reply (RREP) header, back along the reverse route formed by the RREQ (2)

The NREP is carried as an extension to an RREP message (Figure 1-8). The User Entity sending of the NREQ will normally include its own IP address as 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 searched for, can supply the User Entity with the resolved IP address in addition to a unicast route to that IP address. Hence, the User Entity 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

MANETs 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 net-

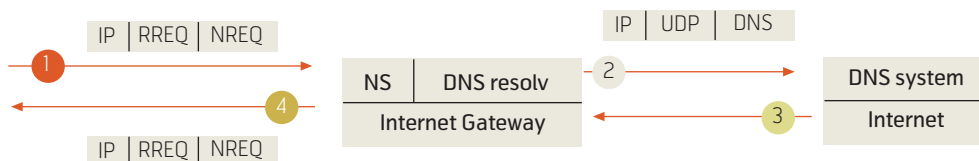


Figure 1-9 An Internet Gateway (IGW) which receives a Name Resolution Request (NREQ) on the MANET (1) may try to resolve the requested name using the Domain Name System on the Internet (2). A successful response (3) may be injected as a Name Resolution Reply (NREP)

work. The scheme proposes to use each IGW as a DNS proxy, as shown in Figure 1-9.

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 NREQ might result in reception of multiple NREPs. If the Named Entity present in the MANET has registered its name in the DNS, both the Named Entity and each IGW 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 article), many Named Entities 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 a Named Entity present locally on the MANET will always have preference over responses arriving from other nodes, such as IGWs, since 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 User Entity has multiple addresses to select from 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 User Entity.

2 Service discovery

The only information two end-points communicating over the Internet need to know, besides their own configuration, is the network address of the end-point they communicate with. Any of the two end-points might offer a wide range of services such as email, ftp, http etc., but there exists no defined way 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 TCP or 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 assis-

tance 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 will introduce 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 since the availability of services is dependent on the networks dynamics.

2.1 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 the question, “*What services are you running?*” and sift through the results looking for printers. This latter approach, which is called *device-centric*, is not only time-consuming, it 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 involves the question “*Who offers these services?*”. Normally three entities, or agents, are needed in a service discovery architecture for a computer network. These are:

- 1 A *User Agent (UA)* that represents the network entity issuing a request to find a service;
- 2 A *Service Agent (SA)* that represents the service offered;
- 3 A *Directory Agent (DA)* for intermediate storage of information about available services.

2.1.1 User Agent

The User Agent (UA) represents the client part in the process of discovering services. The UA may be a software component or an end-user that wants to locate a specific service. In most cases the UA will offer a low level functionality directed towards system components.

2.1.2 Service Agent

The Service Agent (SA) 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.

2.1.3 Directory Agent

A Directory Agent (DA) acts as a cache and will merely collect information from the Service Agents and forward it on demand to User Agents. The UA and SA can use either passive or active DA discovery to find a DA. This is shown in Figure 2-1.

The service model can be divided into three architectures, the *distributed*, *centralized*, and *hybrid* architectures. In the distributed architecture, there are no DAs 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 2-2, 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

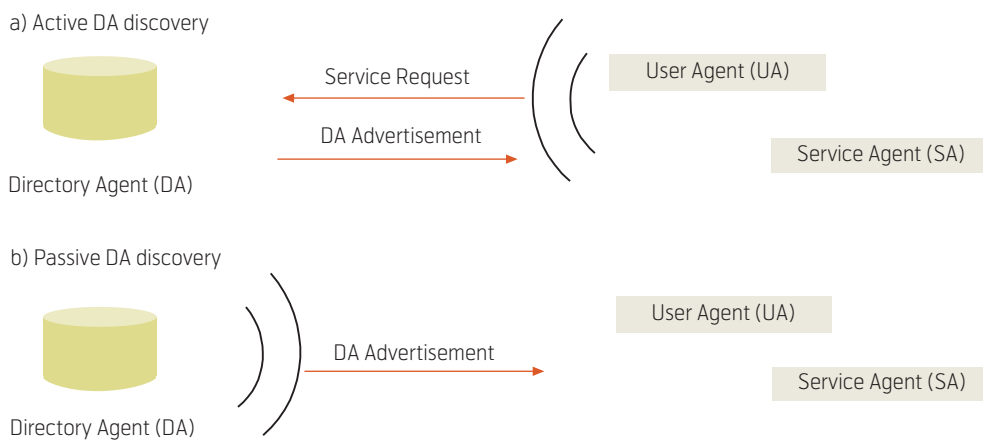


Figure 2-1 Active and passive Directory Agent (DA) discovery

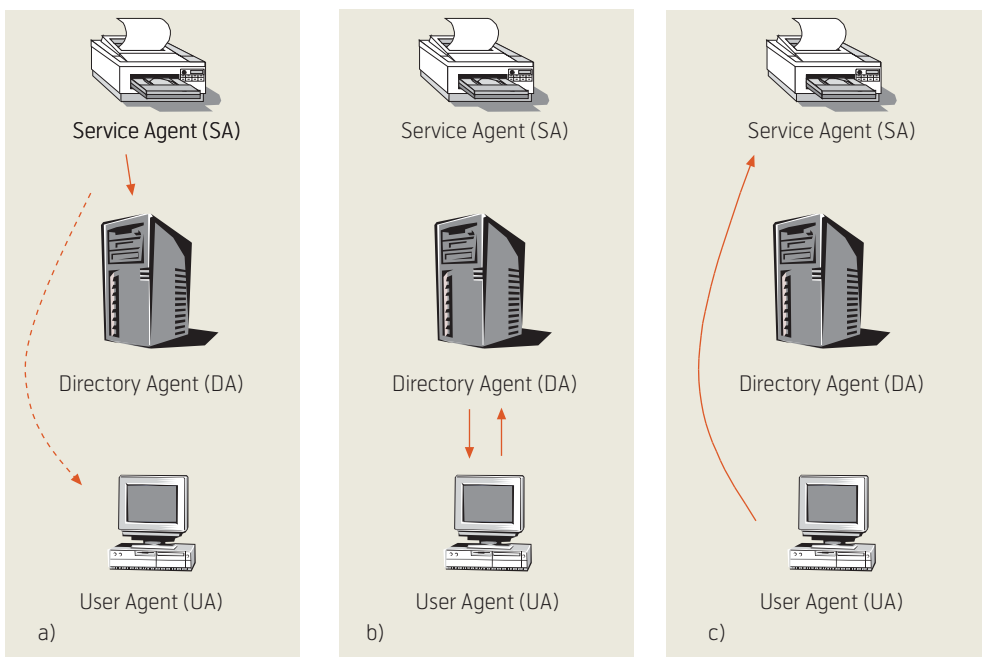


Figure 2-2 A Service Agent announces its services, and announcements can be cached in both the DA and the UA (a). If the UA is searching for a service it has not heard being announced, it can broadcast a service request or contact a DA directly. The DA will respond with information about where the service can be located (b). The UA can access a service it has learnt about via a DA, or from the SA directly (c)

push and *pull* approaches were described in the previous chapter 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.

2.2 Service discovery on the Internet

2.2.1 Current practice for service discovery

When an application on computer *A* wants to connect to 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*. Since *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 consist of:

- 1 Mapping the service name to the name of the computer offering the service;
- 2 Mapping the port number of the service to a service name;
- 3 Resolving the computer name to a network address using DNS or a local name service.

There is no widely used mechanism to undertake the first step of this process in today's IP network. 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*, or a public file transfer server which is given the name *ftp*.

Example:

```
www.some_domain.com
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:

```
<name> <port number>/<transport
protocol> <aliases>
```

An example of a port to service name mapping can be:

```
http 80/tcp www 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 there is no access to any name server.

As can be seen, the procedure for discovering services on the Internet is cumbersome and not very efficient. Other alternatives will be outlined in the following.

2.2.2 Service Location Protocol

The Service Location Protocol (SLP) is an emerging Internet standard provided by 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], which was a mechanism created by Apple and which proved to be a huge success much 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 in the network:

- 1 *User Agent (UA)*: The SLP *User Agent* is a software entity that looks for the location of one or more services. This 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.
- 2 *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.

3 *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 is degraded. By deploying one or more DAs, both SAs and UAs make it a priority to discover available DAs, since 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.

SLP introduces dynamic naming services without the need for any centralized name server or other agents. Since 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 SLP protocol suffers from the lack of implementation support, since companies such as Apple and Microsoft, each on their own push a different service discovery technology. Generally, standardisation is a good thing, but not very useful if no one provides implementation support for the standards. The Service Location Protocol has proved to be useful, especially for UNIX® variants, and an open source project exists (an implementation of SLP can be downloaded from www.openslp.org) to support service discovery on operating systems such as Linux and BSD.

2.2.3 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 protocols *http* and *smtp* would look like this in the zone file for *company_A*:

```
$ORIGIN company_A.com.
@      SOA  server.company_A.com.
root.company_A.com. (1995032001 3600
3600 604800 86400)
      NS  server.company_A.com.
http.tcp.www SRV 0 0 80 server.com-
pany_A.com.
smtp.tcp      SRV 0 0 25 mail.com-
pany_A.com.
server A      172.30.79.10
mail  A      172.30.79.11
```

For example, to locate a *http* server that supports TCP protocol and provides web service, it does a lookup for:

```
_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 solely depends 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.

2.2.4 XML Web Services / UDDI

An XML Web Service is a service that accommodates direct interaction using XML-based messaging (such as 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 WSDL specification [21].

2.2.5 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 the Transport Manager (TM) are used for the actual communication, using Remote Procedure Calls (RPC). 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 allows clients also to download Java code from the lookup service, which is used to access the service. It requires however 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.

2.3 Service discovery on link local networks

2.3.1 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 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. HTTPU and HTTPMU are used by SSDP to generate requests over unicast and multicast.

The SSDP architecture introduces three entities in the network:

- *SSDP Service*: The SSDP service is a Server Agent and represents the individual resources in an SSDP enabled network. The agent is defined in two versions, depending on whether an *SSDP Proxy* is available in the network or not. An 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. When starting, the SSDP Client will search for a proxy, followed by the search for other relevant resources. If an SSDP Proxy is available, all requests are done using unicast. If 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 and 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 improves the scalability when deployed in large networks.

2.3.2 Multicast DNS

DNS Service Discovery is a way of using standard DNS programming interfaces, servers, and packet formats to browse the network for services. As shown in the previous chapter, 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 2-3.

When an mDNS query is sent out for a given service type and domain, any matching service replies with their names. The result is a list of available services to choose from.

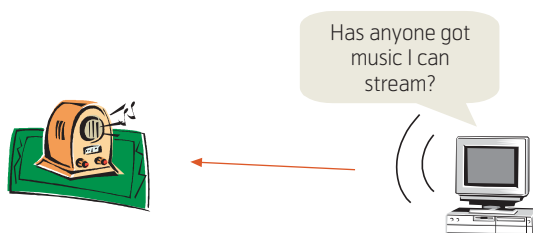


Figure 2-3 Illustrating the principle of multicast DNS

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, i.e. color, pages per minute etc.

Using a mechanism for automatic service discovery greatly simplifies the job of connecting PCs, terminals, wireless units and consumer electronics.

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 that all local hosts can see it. The next time a host needs a list of print spoolers, it already has the list in its cache and does not need to reissue the query.

2.4 Service discovery in ad hoc networks

2.4.1 Service discovery mechanism for MANETs

Discovery of services and other named resources is an important feature for the usability of mobile ad hoc networks (MANETs). The characteristics of ad hoc networks were described earlier. We recall that a MANET is anticipated to be without any pre-existing 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, which enable the client to initiate the selected service on the appropriate server.

Existing service discovery mechanisms, described in previous sections, are designed with a fixed network in mind, and might not fit well to MANETs.

Before a service discovery mechanism for ad-hoc networks can be designed, we need to 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.

2.4.2 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 is referred to as a Service Agent (SA). Furthermore, the architectures are often referred to as the *Distributed*, the *Service-Coordinator-based* and the *Hybrid Service Location Architectures*. The Distributed architecture for MANETs is illustrated in Figure 2-4, and the Service-Coordinator-based architecture is shown in Figure 2-5. The hybrid architecture is a combination of the two: The UA tries

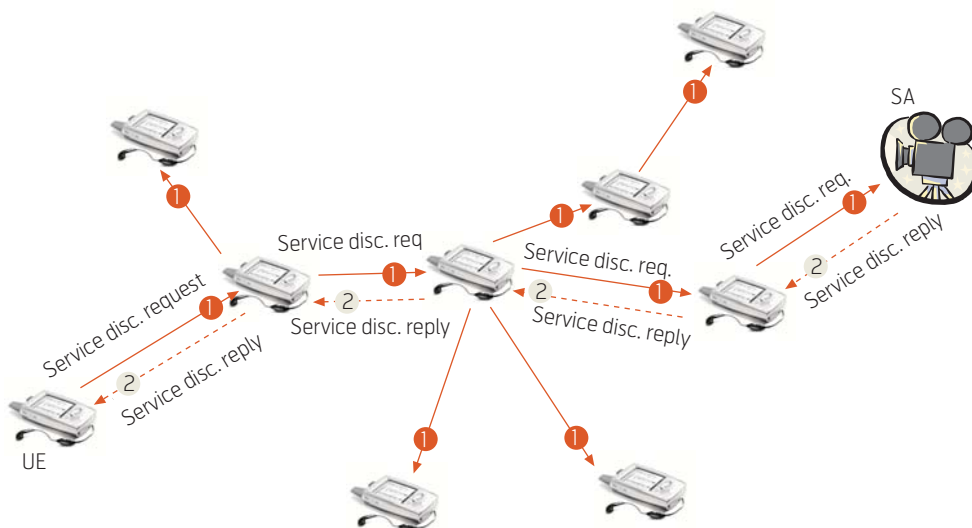


Figure 2-4 The Distributed architecture

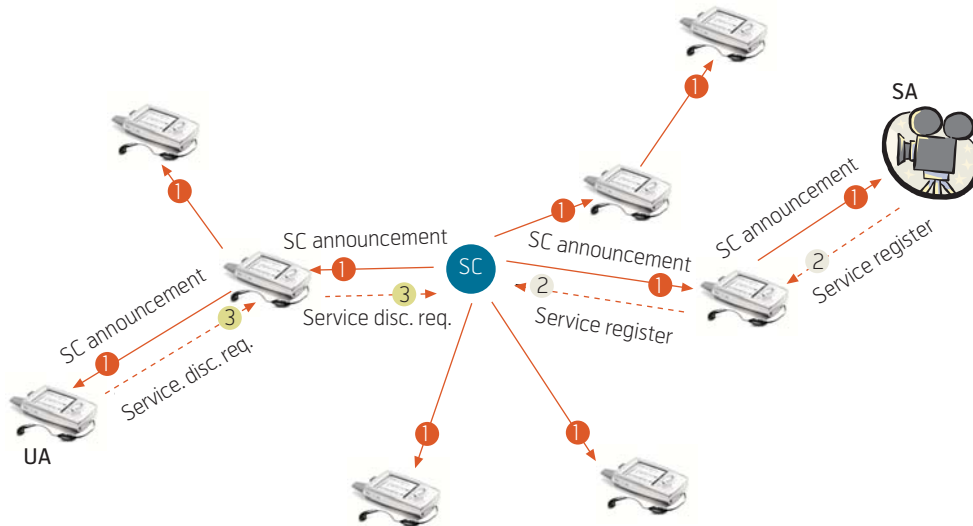


Figure 2-5 The Service-Coordinator-based architecture

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.

2.4.3 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 in the previous chapter), 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. This is illustrated in Figure 2-6.

The advantages of piggybacking service discovery on routing messages are:

- 1 Reverse routes to the User Agent (i.e. client) are established along with the SREQ so that no additional route discovery is necessary to relay the SREP back to the requestor.
- 2 Forward routes to the SC are established along with the SC Announcements so that SREQs and Service Registrations can be unicast to the SC.

- 3 A forward route is established along with the SREP so that no additional route discovery is necessary for further communication with the node issuing the reply.

Figure 2-6 shows how service discovery can be streamlined with the reactive routing protocol. Service Discovery Requests (SREQs) are piggybacked on Routing Request (RREQ) packets, and Service Discovery Replies (SREPs) are piggybacked on Routing Reply (RREP) 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.

2.4.4 Proposed solution for service discovery in reactive ad hoc networks

A solution for service discovery in reactive ad hoc networks is proposed in [27]. It basically uses the same mechanism as was presented for name resolution in the previous 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.

In [27] a *service binding* is defined as a mapping of a service name to an IP address. Different encoding schemes, such as Service Port Request or Service URL, can be used to request a binding for an IP address. An SREQ for a Service URL contains a ser-

Service Discovery Request (SREQ)	=	IP RREQ	Service description
Service Discovery Reply (SREP)	=	IP RREP	Service description and binding
Service Coordinator announcement	=	IP RREQ	Service coordinator information
Service Registration	=	IP RREP	Service description and binding

Figure 2-6 Routing packets carry Service discovery messages

vice 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 SREP 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 a Service Port 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 needs to contain the port number associated with the service application requested.

The proposed service discovery protocol not only 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.

2.4.5 Evaluation of service location architectures in ad hoc networks

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

- 1 Information about services offered on the network is stored on one or a few centralized nodes, referred to as Service Coordinators (SCs) in this article;
- 2 Information about each service is stored on each node that offers the service.

In previous sections we have defined the service discovery architectures according to the two mechanisms above. A solution that is only based on the first mechanism is referred to as a *Service Coordinator based architecture*, while a solution only based on the second mechanism is referred to as a *Distributed architecture*. Finally, a solution based on a mixture of both the first and the second mechanism is referred to as a Hybrid architecture.

In the next section we evaluate the performance of the Hybrid and Distributed architectures in a reactively routed MANET. The architectures were presented in detail in the previous chapter and summarized more briefly above in this article.

2.4.6 Architecture evaluation

The evaluation of the Distributed and Hybrid architectures is based on results from [28]. The architectures can be configured by different settings of the following two parameters:

- 1 *Flooding scope*: This parameter determines the maximum number of hops a flooded Service Discovery Request is allowed to traverse in the network. (For example, the flooding scope is of four hops in Figure 2-4 and of two hops in Figure 2-5.)
- 2 *SC announcement scope*: This parameter determines the maximum number of hops a flooded SC Announcement is allowed to traverse in the network. (For example, Figure 2-5 illustrates a situation with 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.

In [28] it was observed that the differences in delays between the two architectures are only in the order of a few milliseconds. Since 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]:

$$Serv.Avail. = \frac{\text{number of positive service replies}}{\text{total number of service requests generated}}$$

A *positive Service Discovery Reply* means a successful contact to this server via the given access information, i.e. that a route to the resolved server can be found.

It is observed in Figure 2-7 that the service availability is indeed higher with the Hybrid approach. Figure 2-7 also shows how the presence of SCs (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 that SCs improve the service availability, is that in some cases the SC will be positioned

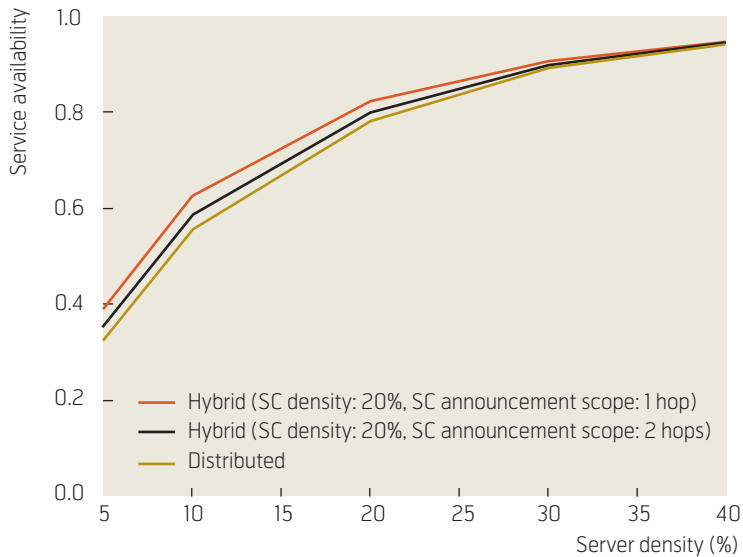


Figure 2-7 The introduction of SCs improves the service availability

in between the UA and SA. We may return to Figure 2-5 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, since 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 2-7 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 %. Since 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 SCs introduced in the Hybrid architecture yield higher service availability, it also results in extra message overhead, as observed in Figure 2-8. 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 2-7 and Figure 2-8 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 [28]. It takes into consideration a large range of control parameters, such as server density, SC 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 [28] it is generally observed that for any Hybrid configuration with a given SC 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. Hence, service discovery protocols that use Service Coordinators (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 Service Coordinators is negligible compared to the extra message overhead it causes.

In addition to the analyses presented in [28], there are several other arguments that are in favor of not 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 pro-

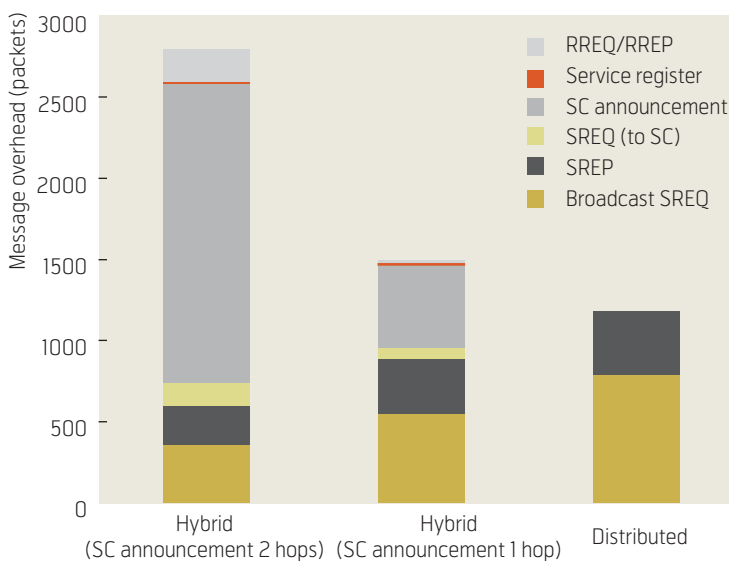


Figure 2-8 Detailed comparison of message overhead by message type. (Server density at 20 %, flooding scope of two hops)

viding 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 Service Coordinators, which might require a substantial amount of network resources.

References

- 1 Mockapetris, P. *Domain names – concepts and facilities*. STD 13, RFC 1034, Internet Engineering Task Force, November 1987.
- 2 Mockapetris, P. *Domain names – Implementation and Specification*. STD 13, RFC 1035, Internet Engineering Task Force, November 1987.
- 3 Microsoft TechNet. *Windows Internet Naming Service (WINS)*. <http://www.microsoft.com/technet/archive/windows2000serv/evaluate/featfunc/nt5wins.msp> (Last visited: 7 April 2005).
- 4 Sun Microsystems. *The Network Information Service (NIS) / Yellow Pages*. <http://www.sun.com> (Last visited: 7 April 2005).
- 5 Wellington, B. *Secure Domain Name System (DNS) Dynamic Update*. RFC 3007, Internet Engineering Task Force, November 2000.
- 6 Chesire, S, Krochmal, M. *Multicast DNS*. IETF Internet Draft, draft-cheshire-dnsext-multicastdns-04.txt, February 2004. Work in Progress.
- 7 Aboba, B, Thaler, D, Esibov, L. *Linklocal Multicast Name Resolution (LLMNR)*. draft-ietf-dnsext-mdns-39.txt, March 2005. Work in Progress.
- 8 Macker, J, Corson, S. *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*. RFC 2501, Internet Engineering Task Force, January 1999.
- 9 Perkins, C E, Royer, E M, Das, S R. *Ad-hoc On Demand Distance Vector (AODV) Routing*. RFC 3561, Internet Engineering Task Force (IETF), July 2003.
- 10 Johnson, D B, Maltz, D A, Hu, J-C. *The Dynamic Source Routing Protocol*. draft-ietf-manet-dsr-10.txt, July 2004. Work in Progress.
- 11 Engelstad, P E, Do T V, Egeland, G. Name Resolution in On-Demand MANETs and over External IP Networks. *Proceedings of IEEE Int. conf. on Comm. 2003 (ICC 2003)*. <http://www.unik.no/~paalee/publications/NR-paper-for-ICC2003.pdf>
- 12 Engelstad, P E, Do T V, Jønvik, T E. Name Resolution in Mobile Ad-hoc Networks. *Proceedings of 10th Int. Conf. on Telecom. 2003 (ICT 2003)*. <http://www.unik.no/~paalee/publications/NR-paper-for-ICT2003.pdf>
- 13 Clausen, T, Jacquet, P (eds.). *Optimized Link State Routing Protocol (OLSR)*. RFC 3626, Internet Engineering Task Force (IETF), October 2003.
- 14 Engelstad, P E, Egeland, G, Koodli, R, Perkins, C E. *Name Resolution in On Demand MANETs and over External IP Networks*. IETF Internet draft, draft-engelstad-manet-name-resolution-01.txt, February 2004. Work in Progress.
- 15 Apple Computers. *AppleTalk*. <http://www.apple.com> (Last visited: 7 April 2005).
- 16 Guttman, E, Perkins, C, Veizades, J, Day, M. *Service Location Protocol, version 2*. RFC 2608, Internet Engineering Task Force (IETF), June 1999.
- 17 Gulbrandsen, A, Vixie, P, Esibov, L. *A DNS RR for specifying the location of services (DNS SRV)*. RFC 2782, Internet Engineering Task Force (IETF), February 2000.
- 18 Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework*. <http://www.w3.org/TR/soap12-part1/>, W3C Recommendation, June 2003.
- 19 Bray et al. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation, <http://www.w3.org/TR/2000/REC-xml-20001006>, October 2000.
- 20 OASIS UDDI. *Universal Description, Discovery, and Integration (UDDI) 2.0*. <http://www.uddi.org> (Last visited: 7 April 2005).
- 21 Christensen, E, Curbera, F, Meredith, G, Weerawarana, S. *Web Services Description Language (WSDL) 1.1*. W3C Note, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, March 2001.
- 22 The Salutation Consortium. *Salutation Architecture Specification Version 2.1*. <http://www.salutation.org>, (Last visited: 7 April 2005).

- 23 Sun Microsystems. *Jini Network Technology*. <http://www.jini.org> (Last visited: 7 April 2005).
- 24 Goland et al. *Simple Service Discovery Protocol /1.0*. Internet Draft, draft-cai-ssdp-v1-03.txt. Work in Progress, Oct 1999.
- 25 Microsoft Corporation. *Universal Plug-and-Play (UPnP) Forum*. <http://www.upnp.org> (Last visited: 7 April 2005).
- 26 Chesire, S, Krochmal, M. *Multicast DNS*. IETF Internet Draft, draft-cheshire-dnsexst-multicastdns-04.txt, February 2004, Work in Progress.
- 27 Koodli, R, Perkins, C E. *Service Discovery in On Demand Ad Hoc Networks*. IETF Internet draft, draft-koodli-manet-servicediscovery-00.txt, October 2002. Work in Progress.
- 28 Engelstad, P E, Zheng, Y, Koodli, R, Perkins, C E. *Service Discovery Architectures for On-Demand Ad Hoc Networks*. *International Journal of Ad Hoc and Sensor Networks*, Old City Publishing (OCP Science), 1 (3), 2005.
- 29 Engelstad, P E, Zheng, Y. *Evaluation of Service Discovery Architectures for Mobile Ad Hoc Networks*. *Proceedings of the 2nd annual conference on Wireless On-demand Networks and Services (WONS 2005)*, St. Moritz, Switzerland, Jan. 19–21, 2005.

Paal E. Engelstad completed his PhD on resource discovery in Mobile Ad hoc and Personal Area Networks in 2005. He has also a Bachelor and Masters degree (Honours with Distinction) in Applied Physics from NTNU, Norway, and a Bachelor degree in Computer Science from University of Oslo, Norway. After working five years in industry, he joined Telenor R&D where he focuses on IETF and IP technology (e.g. IP mobility, IPv6, QoS, MANET and AAA-issues) and IEEE wireless technologies (e.g. 802.11, 802.15 and 802.16). Paal Engelstad has published 28 refereed papers and holds three patents (two pending).

email: Paal.Engelstad@telenor.com

Geir Egeland holds a B.Eng (Hons) from the University of Bristol and has for the last ten years worked as a research scientist in the field of mobile networks. He is currently with Telenor R&D where his work is mainly focused on mobility for IP networks, with a particular emphasis on MANET and IPv6. Geir Egeland was formerly employed by the Norwegian Defence Research Establishment (NDRE) as a research scientist working on design and analysis of MAC and routing protocols for mobile ad hoc network.

email: geir.egeland@telenor.com