11.12.2003
Michael Dyrna

TECHNISCHE UNIVERSITÄT MÜNCHEN

Hauptseminar im Wintersemester 2003 / 2004

Ad-hoc networking:
concepts, applications, and security

Peer2peer Network Service Discovery for Ad hoc Networks

Abstract

Ad-hoc networks are networks formed by wireless and mobile devices. These networks do not rely on a fixed infrastructure and thus have to be self-organising. One problem that arises therefore is that components need to find each other in order to provide and use services. The solution to this problem lies in Service Discovery Protocols.

After defining the goals of service discovery, this paper first describes the general design of service discovery protocols and explains the necessary mechanisms in depth. Based on that, the five most important existing protocols will be presented and compared. Finally we will have a look at security aspects by deriving requirements and mechanisms from possible threats and comparing these thoughts to the existing protocols.

I. Introduction

1. Managing services in non-ad-hoc networks

In traditional (non-ad-hoc) networks, system administration is so time-consuming that most companies nowadays have an in house department for that task or even charge other companies with it.

For deploying a new service on the network, the system administrator has to assign it a network address and publish that address to all users who want to connect to it (whether it is a printer, a file server, a database or whatsoever). When a service fails or loses its connectivity, clients cannot automatically be redirected to a substitute service that is up and running, nor are they informed when the component is available again. In many cases, different services need different drivers on every client that wants to use it. In such a network, hosting a guest who might want to use printers, beamers, Internet connection is so complicated that it is usually avoided unless absolutely necessary.

The developers of today's state-of-the-art network architecture TCP/IP assumed rare changes in network topology and did not foresee current developments like the rising desire of mobility, host roaming or hosts that can connect spontaneously to a network, an ad-hoc network.

2. Background

These assumptions are no longer valid. With proliferation of mobile communication – i.e. mobile phones with higher and higher data rates, UMTS, PDAs that allow mobile network connectivity – mobility and modularity are the current goals of system development. The classical client server paradigm is hardly applicable to modern networks any more and is increasingly displaced by peer-to-peer approaches, allowing constant changes in network topology and making fixed infrastructure obsolete.

What is the difference between (mobile) ad-hoc networking and peer-to-peer? The basis of both is self-organisation, independence from centralised servers and support for constantly changing network topology. Whereas ad-hoc-networking refers more to the lower network layers, i.e. radio transmission instead of wired connections, dynamic host address assignment, special routing mechanisms and so on, the peer-to-peer paradigm refers to the application design and is an antipode to the client server paradigm. Thus, applications in ad-hoc networks are very likely to use the peer-to-peer paradigm, but conversely, peer-to-peer applications are not dependent on the network architecture underneath but are currently gaining popularity even in traditional TCP/IP networks.

The displacement of the client server paradigm where services reside on servers whose address and capabilities are well-known and normally statically configured at the client side, makes more flexible management of services necessary. For that, different consortiums have developed approaches of *service discovery* mechanisms that are subject of this paper.

3. Goals of Service Discovery

Service discovery protocols are designed to allow modularity in networks. They enable components to find each other on the network, to join and leave freely and to provide them with a consistent view of other components.

For traditional (non-ad-hoc) networks the main advantage lies in allowing easier administration, particularly change management, and to make deployment of services more flexible. In ad-hoc networks, means for service discovery are even *mandatory* since they are designed for regular and unexpected changes in network topology.

In a service discovery environment, services advertise themselves, supply details about their characteristics and provide an access interface. Clients may locate a service by its type (e.g. "modem") or even by attributes (e.g. "modem that allows dialling international phone numbers and whose baud rate is 33,6 kbps or higher") and make a selection in instances where more than one service was found.

4. Scenario: Mobile phone, PDA and headset

Let us illustrate the usefulness of service discovery with an example scenario that is common today. Someone has a mobile phone, a PDA and a headset that he wants to build an ad-hoc network with. All three devices offer certain services but can also take advantage of the other devices' services. The mobile phone offers

sending short messages, dial-up connection to the Internet or to other phones or modems, audio in- and output (microphone and speaker) and video in- and output (camera and small screen). The PDA offers video output, data storage and audio in- and output. The headset only offers audio in- and output. When the user requires one device to use a service of another (e.g. the PDA connecting to the Internet via the mobile phone or handling telephone calls on the headset), with help of service discovery mechanisms, he does not need to know anything about the network addresses of the three devices. The PDA can automatically search for a device that offers a dial-up Internet connection (mobile phone), and the phone can automatically search for a device that offers audio in- and output (headset).

II. Design of Service Discovery Protocols

In this chapter we will not deal with existing implementations of service discovery protocols. Rather we will define properties that those protocols should fulfil and derive necessary entities, methods and data items hence with. All of the protocols that will be presented in chapter IV follow this general design more or less. A more exact analysis of this chapter's topics can be found in [2].

1. Properties

Service discovery protocols are supposed to possess the following four properties.

They must;

(a) enable software components, i.e. services and service users, to find each other on a network.

(b) provide a means to describe services so that the service user can determine if a discovered service matches its requirements.

(c) include techniques to detect changes in component availability.

(d) maintain a consistent view of components in a network.

The latter two are especially relevant in ad-hoc networks where nodes might appear and disappear unexpectedly.

2. Entities

Service discovery protocols define three entities, one of which is optional. All entities are software components that are distributed on a network.

(a) The service manager holds information about one or more services or devices together with their attributes and interfaces.

(b) The service user queries for a certain service or device it wants to use and selects the most appropriate one found.

(c) Optionally, a service cache manager can act as a broker between service managers and service users in order to reduce network traffic and to increase performance. However, the design should ensure that all discovery activities are possible in the absence of service cache managers. Not all of the existing protocols support service cache managers.

Note that in two-party-architecture multicasts are necessary for every discovery process, whereas in three-party-architecture multicasts from service users and service managers are only necessary for initial discovery of the service cache manager. That is why the latter case is desirable for larger networks.

A graphical presentation of the two possible architectures is given in the figures 1 and 2.

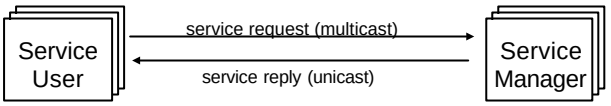

Figure 1: architecture without service cache manager

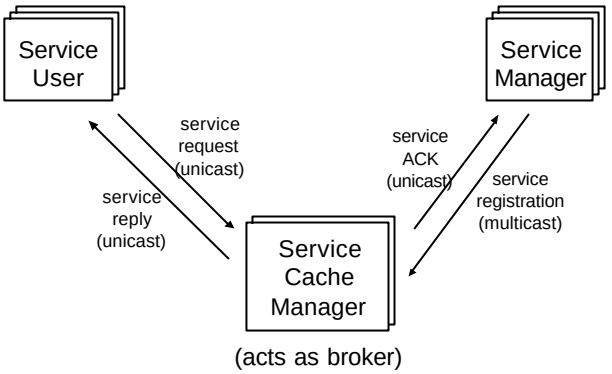

Figure 2: architecture with service cache manager

3. Processes

3.1 Discovery

For services to be able to find each other a discovery process has to take place. There are three different kinds: aggressive, lazy and directed discovery.

In aggressive discovery the service user sends out multicast requests, either a fixed number in fixed intervals or until it has discovered enough service managers or service cache managers. The service managers or service cache managers send a unicast reply to the requestor if they can provide the requested service with the given attributes. Aggressive discovery is normally used when a node has just joined the network in order to discover the existing services for the first time.

In lazy discovery, service managers and service cache managers advertise their services in fixed intervals by multicast communication. Service users and service cache managers can store the received data for later use of the advertised services. Lazy discovery is useful to detect changes in component topology during operation.

In directed discovery a service user contacts a service manager or service cache manager directly in order to probe for a previously advertised or discovered service.

3.2 Registration

A service manager holds a set of service descriptions. It must register all service descriptions with all service cache managers discovered. The service manager and the service cache manager negotiate a lease time, which is the time after which the service cache manager will drop the registration if it is not renewed. After the registration any service user can discover the registered service or can register its interest in receiving notifications about changes concerning the registered service. The reception of notifications is granted only for a negotiated period.

3.3 Consistency Maintenance

Since we are dealing with distributed systems, new services can be deployed, obsolete ones can be removed, nodes and links may appear, disappear or fail. Thus, a consistent view of all services on the network cannot be guaranteed. In order to limit such inconsistencies, a service registration is only valid for a limited period of time, the lease time. If the registration is not renewed within this time, the service cache manager will remove the registered service description. So if a registering component fails, its description will be removed automatically and registered service users are notified about the removal where appropriate.

4. Data Representation

Service discovery protocols must define a data scheme to represent a service. This is called the service description. A service identity is mandatory. It must be unique and contain the service location, i.e. its network address. Also mandatory is the service type, i.e. what kind of service is being described. The description can furthermore contain attributes that characterise the service more exactly, a user interface and / or a programme interface. The latter two allow access to the service from a remote point and are provided by one single existing protocol. An example is given in Table 1.

Identity	`192.168.8.15/mpool15`
Type	`Modem`

Attributes	`baud=28800`
	`phonenumbersallowed=national`
	`dialprefix='0'`

Table 1: example for a service description

III. Mechanisms and Techniques

In the following paragraphs we have to consider the two architectures mentioned in II.2 separately. We will call the architecture with a service cache manager *three-party-architecture* in contrast to the one without a cache entity (*two-party-architecture*). A more exact analysis of this chapter's topics can be found in [2].

1. Consistency Maintenance

After logging onto a network and discovering all available services, a service user has to ensure that his knowledge about existing services stays consistent with the actual distributed state. There are two basic mechanisms for that: polling and notification. In polling, the service user initiates receiving updates, whereas with notification the service managers propagate changes as they occur.

1.1 Polling

When a service user applies polling, it sends queries to obtain up-to-date information. In a two-party-architecture it sends these queries directly to the previously discovered service managers and receives the

responses via unicast. In a three-party architecture polling consists of two processes. Service managers propagate changes concerning provided services regularly to the service cache managers and each service user queries its relevant service cache manager.

1.2 Notification

When using the notification mechanism, changes in service descriptions are propagated from the service managers to the service users. In a two-party architecture a service user has to register with the service manager. This registration is only valid for a negotiated time and needs to be renewed regularly. The service manager then announces changes to all service users that registered with it. In three-party-architecture service managers announce changes to the service cache managers. The cache managers do not need to register for that purpose. The service users register with the service cache manager in order to be notified about changes as soon as the cache manager receives them.

2. Failure Detection and Recovery

So far we have only considered topology changes that occur on purpose and where components can thus announce their sign-off. Particularly in ad-hoc networks we also have to cope with changes due to failures. Hosts, processes and network links may fail, packets can get lost on the network, transmission can be jammed, hosts may move out of radio coverage, and so on.

To treat these failures correctly, a service discovery architecture must provide means to detect them and to recover after their termination. For that, two basic mechanisms exist: soft state persistence by monitoring periodic announcements and application-level persistence by bounded retries and remote exceptions.

2.1 Soft State Persistence with Periodic Announcements

Discovery protocols define key messages that components send out in fixed intervals to periodically announce their current state. Monitoring these key messages empowers other components (*listeners*) to cache almost-real-time states, i.e. they can store the information and overwrite it with every update. To detect communication failure, i.e. if a component does not receive such a “heartbeat message” from a remote component within the given interval, it may assume that the communication path or the remote component itself has failed. In order to keep a consistent view of reachable components, the listener deletes the cached information after non-appearance of an expected status update. As soon as the remote component or the communication link to it is back up and a new heartbeat message is received, the listener regards the remote component as available again.

2.2 Application-level Persistence with Bounded Retries

Another means to detect a failure are bounded retries. This mechanism is widespread in networking. If a component does not answer a request, the request is resent several times. If the number of trials exceeds a given bound, the client assumes that the remote component has failed and throws a remote exception to the application layer (which is the application that wants to use a service). This mechanism is reasonable in systems where discovery is normally initiated by applications. It is automatically given in discovery protocols that presume reliable network communication, since the transport layer reports the inability to send data by definition.

The application has several possibilities to deal with such a remote exception;

(a) It can ignore it. This is reasonable for polls and notifications since they recur periodically.

(b) It can retry the operation after a certain period of time and thus recover from the failure as soon as the next communication attempt is successful. Until then it must assume that the remote component is not reachable.

(c) It can discard knowledge about the remote component. If the remote component is a service manager (two-party architecture or poll from service cache manager in three-party architecture), knowledge about its service descriptions is discarded. This corresponds to the soft state persistence mentioned in 2.1 and expects the peer entity to send a notification when it is back up and reachable. If the remote component is a service cache manager (three-party architecture), knowledge about its existence is discarded, possibly making it necessary to discover other cache managers.

IV. Existing Service Discovery Protocols

Now that we have gained detailed insight to the general design of service discovery protocols, we will see what protocols exist at the moment. There are different approaches by different working groups, none of which is widely used so far. Most of them are still under development. Their characteristic attributes can mostly be derived from the companies that were contributing to the development.

1. Service Location Protocol, Version 2 (SLP2)

The Service Location Protocol (SLP) is the widest spread and most lightweight of the presented protocols and managers only the discovery but not access to services. It was developed by the IETF (Internet Engineering Task Force) SvrLoc working group, the most important members of which are SUN, HP, Novell, IBM and Apple. This working group also provides two reference implementations.

SLP is vendor and platform independent. It relies on TCP/IP as network protocol. For most communication the unreliable and packet oriented protocol UDP is used. TCP is only used where data does not fit in one datagram. Protocol messages are mixed binary and string-based, whereas binary representation is mostly used for headers and string representation for service descriptions.

It defines the same entities as described in section II and supports both two- and three-party-architecture. It names the entities *User Agent*, *Service Agent* and *Directory Agent*. The Directory Agent can optionally – if present – be announced via DHCP (Dynamic Host Configuration Protocol) or configured statically at the client side. SLP2 names aggressive and lazy discovery *active* and *passive.*

As service identity SLP2 defines *Uniform Resource Locator*s (URL) that are commonly known from other protocols and consist of a service type, host address, port number and path. E.g.:

```
service:printer:lpr://office51.business.com:515/lpr02
```

The set of service attributes is called Service Template here and also consists of attribute-value-pairs. Service Templates have to be registered with IANA (Internet Assigned Numbers Authority). Here is an example:

```
Attributes = (Name=Igore),(Description=For developers only),
             (Protocol=LPR),(location-description=12th floor),
             (Operator=James Dornan \3cdornan@monster\3e),
             (media-size=na-letter),(resolution=res-600),x-OK
```

In order to map administrative groupings to service discovery domains, SLP2 supports *scopes*, which is a great help for scaling SLP deployments to larger networks.

2. Jini

Jini is an extension of Java and has been developed by a consortium lead by Sun Microsystems. Other members are AOL and many mobile equipment vendors. It enables devices supporting Java to connect with and provide services to each other. Jini is an open standard; the only pre-condition is a Java Virtual Machine running on the device. Sun provides a 46 KB reference implementation on its web site.

The design is again very similar to the one described in section II. However, it does not make a difference between service users and service managers – any component can lookup and invoke services from any component. The service cache manager is called *Lookup Service*. The Lookup Server holds a *Lookup Table* and is optional. In case of absence of the Lookup Server, components operate their own lookup table.

The lookup table contains pointers to services and Java-based mobile programme code. Thus, the result of a discovery process is not only a URL to a service, but a Java programming interface that can be accessed directly (*service proxy*) and would correspond to a driver in traditional architectures. This mode of accessing remote components is enabled by *Java Remote Method Invocation* (RMI). This is very specific for Jini and distinguishes it from all other service discovery protocols presented in this paper.

The three operations supported by the Lookup Table are *store*, *match* and *fetch*, which correspond to service registration, service lookup and the download of the service proxy.

To subdivide networks with large numbers of components into administrative scopes, Jini supports *groups*. In order to avoid conflicts, Sun recommends using domain name style (e.g. students.in.tum.edu). Components can be part of zero or more groups.

The process of discovering a service cache manager is called *discovery*. The registration process is called *join*. They do not work any differently than described in the section II.

3. Salutation

Salutation was developed by a consortium of more than thirty companies, the most important of which are IBM, HP, Sun and Cisco. The specifications are freely available like as it is the case with every other protocol presented here. Salutation focuses on platform and network independency. Therefore it does not require TCP/IP but works atop any transport layer protocol. Other than managing discovery and advertising of services, it also handles access to components by providing a transparent communication pipe.

Salutation's service cache manager is called *Salutation Manager* and is mandatory. Other than broking requests, it handles all communication between clients and servers network independently. For that, it relies on one or more *Transport Managers*, at least one for every network protocol. So the Transport Manager is an abstract communication layer and a Salutation Manager can act as a proxy between components on different network types.

Services register with one Salutation Manager; clients request services only from one Salutation Manager. If a Manager does not offer a requested service by itself, it asks other Managers for the service.

The Salutation Consortium has elaborated *Functional Units*, which are classes of devices and applications provided by server components, e.g. print service or scan service. For every Functional Unit, the consortium has also defined a fixed set of attributes.

The Salutation Manager is involved with four kinds of processes: Service Registration, Discovery and Availability are familiar from section II. The main difference here is that every client and every server communicates only with one assigned Manager. Service Session Management is the process of handling communication between one client and one server. A session can be established in one out of three modes: In Salutation Mode, the Manager does not only forward packets but also defines formats that are used in the session. In Emulated Mode the Manager just forwards packets, whereas in Native Mode client and server communicate directly (which is only possible if they operate on the same transport protocol) over a proprietary application protocol.

4. Universal Plug and Play (UPnP)

The consortium that developed UPnP was founded and is lead by Microsoft. The most important other members are Intel, Compaq and Cisco. UPnP is published under a free license. The most well-known commercial product with an implementation is Windows XP. Thus, this is currently the widest spread service discovery protocol. Because of its simplicity and its supporters on both the software and hardware market, we can expect UPnP to gain importance over the other service protocols in the future.

UPnP is basically an extension of the existing Windows Plug and Play mechanism where components don't have to reside on the same host but rather have to be reachable via a TCP/IP network. HTTP-over-UDP is used for discovery and advertising; SOAP – a protocol atop regular HTTP that transports XML encoded data and that is normally used for remote (web) service calls – is used for transactions.

The entities are called *Control Point* and *(Controlled) Device*. The design only supports two-party architecture, i.e. there is no caching entity and all components have to use multicast to advertise or discover services. That is why UPnP does not scale well on large networks. Furthermore there are no mechanisms for consistency maintenance, which limits its usage to networks with reliable network communication. Thus it is not adequate for wireless ad-hoc networks.

After the initial discovery of devices, a control point can retrieve detailed information about any discovered device (*capabilities*) and interact with it. It is not possible to search for devices with given attributes, though. The representation scheme for device information, actions and responses is XML.

Interaction can be the following three procedures:

a) Control: A control point sends actions to a device and receives actions specific values. The effect of an action is modelled by changes in the run-time variables of the device.

b) Eventing: A control point can subscribe to receive a notification whenever a run-time variable of a device changes.

c) Presentation: A device can provide HTTP access so that a user can view the device status or control the device with a browser.

5. Bluetooth Service Discovery Protocol (SDP)

Bluetooth wireless technology is a relatively new short-range communication system designed for robustness, low power consumption and low cost. Its architecture describes all network layers from physical (radio transmission around 2,4 GHz) up to application layer specific topics like defining so called *profiles* (e.g. data synchronisation profile or telephony control profile) out of which applications may choose.

The Bluetooth specification was developed by Microsoft, Intel and the most important mobile equipment manufacturers.

Service discovery is part of the Bluetooth protocol stack and forms an own sub-layer. Every device has an SDP server and an SDP client (which correspond to the service user and service manager in section II). Bluetooth networks are pico nets with a maximum of 256 devices, only eight of which can be *active*, i.e. can communicate, at the same time. Several pico nets can overlap but the core specification does not define any routing mechanism, so that neither discovering nor using services in a neighbouring pico net is possible. Because devices discover each other when joining the network, no service cache manager is necessary.

Bluetooth SDP allows searching for service type (*ServiceSearchRequest*) and attributes (*ServiceAttributeTransaction*) and browsing (*ServiceSerachAttributeTransaction*) all services available. The latter application is only reasonable in Bluetooth because of the limited number of devices in one network. Providing access to services is not subject to SDP.

Notification about new devices (and thus new SDP servers) becoming available or disappearing is provided by other means of the Bluetooth architecture. Since Bluetooth is designed for ad-hoc use, all consistency maintenance is delegated to lower network layers.

Services are represented by *Service Records* which consist of *Service Attributes*.

6. Comparison and Evaluation

All discovery protocols are relatively new, and none of them has achieved great proliferation so far. Behind every one of them stand important companies that are strong on the hardware and/or software market and that are already implementing or will implement “their” protocol in components (e.g. more and more mobile phones and PDAs support Bluetooth; Microsoft Windows XP implements UPnP). All protocols have different strengths and problems in different environments, so that probably not only one protocol will survive. However, different environments will favour one or a few service discovery architectures.

Table 2 summarises characteristics of the five service discovery protocols presented above. Their suitability for mobile ad-hoc networks will be discussed subsequently.

	SLP2	Jini	Salutation	UPnP	Bluetooth SDP
(main) developer	IETF	Sun	Salutation Consortium	Microsoft	Microsoft + Intel
network transport	TCP/IP	independent	independent	TCP/IP	Bluetooth
programming language	Independent	Java	independent	independent	independent
OS and platform	dependent	independent	independent	dependent	independent
attributes searchable	yes	yes	yes	no	yes
service cache manager	optional	optional	mandatory	no	no
scoping	“scopes”	“groups”	no	no	not necessary

Table 2: Comparison of the five most important service discovery protocols

The two architectures that come from peer-to-peer approaches on top of traditional networks (TCP/IP) are SLP2 and UPnP. Since TCP/IP itself is not designed for ad-hoc networks, operation of these two discovery protocols might not be satisfying in sophisticated ad-hoc environments like military units or automotive where high node mobility is given.

The other three protocols, Jini, Salutation and Bluetooth's SDP, were particularly developed for the purpose of ad-hoc networking. Characteristics necessary for that (powerful consistency maintenance and failure handling) are very distinctive in these protocols.

Bluetooth offers the most integrated solution since it's SDP is part of the Bluetooth protocol stack so that every device that supports Bluetooth automatically supports SDP. Almost all applications that access Bluetooth make use of its SDP protocol, whereas applications for the other four protocols are still rare.

What obviously distinguishes Jini from the other protocols is its integration into the Java programming architecture and the support for code mobility with Java Remote Method Invocation (RMI). In ad-hoc architectures where all devices are presumed to have a Java Virtual Machine (JVM), Jini should be the choice for service discovery.

The main advantage of Salutation is that it is able to connect different network architectures and enables not only service discovery but also access to services across network borders. This is made possible by its Salutation Manager – an abstraction layer over network transport – one for every used network architecture must be implemented. This makes Salutation the only reasonably choice in heterogeneous networks (e.g. Ethernet + IrDA + proprietary home automation bus).

Scalability is an important issue in ad-hoc networks that exceed the requirements of connecting two or a few devices in home or office environment. Bluetooth (and it's SDP) are not designed for large networks at all. Although overlapping small Bluetooth networks can form a so called *scatter net*, there are no means for inter-network-communication. Since UPnP does not support three-party architecture, most requests must be transmitted via multicast. This makes it also improper for large ad-hoc networks. In non-ad-hoc TCP/IP networks multicasts are usually limited to network gateways, which allows at least an inflexible way of scoping. SLP2, Jini and Salutation in contrast support service brokers as well as dissecting large networks into administrative scopes, so that they should be the choice in large networks.

V. Security Aspects

There is not much literature available about security aspects in service discovery. One possible reason for that is that service discovery is a relatively young field in computer science. Another one is that due to their low proliferation, the existing protocols were not exposed to real threats and deep security examination yet.

As in any other context, a designer of a service discovery protocol has to make a trade-off between security and ease of use. In most cases, these two aims contradict. Plug-and-play at its best (i.e. two devices are supposed to connect whether they have seen each other before or not) is very easy to handle but totally insecure, whereas total security takes high discipline and partly contradicts the ad-hoc paradigm (i.e. the term "ad-hoc" is not adequate any more if a new device in a network has to request permission from every component he wants to cooperate with).

Fortunately, many security mechanisms relevant for service discovery are covered by underlying network layers so that a designer only has to consider protocol specific aspects.

1. Threats Specific To Service Discovery

This paper does not claim to examine all possible threads systematically. Rather we are going to consider three obvious threats that might be interesting for attackers and will derive requirements and necessary security mechanisms in the subsequent paragraph.

1.1 Perturbation of Discovery

The goal of service discovery is to enable components distributed on a network to find each other in order to communicate and cooperate. Therefore the most obvious thread is to perturb the discovery mechanisms itself. This comprises denial-of-service attacks against components (attacking service cache managers seems particularly effective), flooding the network with nonsense-services and registering and de-registering services without permission. The latter one might be performed by sending commands directly or – where signatures are used – by replaying a previously eavesdropped command packet.

1.2 Unauthorised Use of Services

The use of services often incurs costs (e.g. print or dialup service) and / or enables access to sensitive data. So using a service without permission can be considered as an attack.

1.3 Man-in-the-Middle attack

A more sophisticated attack one could imagine is an attacker faking a service identity in order to make users send sensitive data to it. To conceal the attack, he forwards the data to the "real" service. This is a classical man-in-the-middle attack.

As an example, an attacker could fake a printer identity, eavesdrop documents to be printed and forward the documents to a real print service. End-to-end encryption would be useless in this case since the attacker's entity is one end of the communication.

2. Requirements and Mechanisms

From the above examples we can derive security requirements for service discovery architectures.

Preventing denial-of service attacks and flooding components with non-sense service registrations means guaranteeing **availability**. For making it unable for service users as well as service managers to fake their identity, we have to ensure **authenticity** of all parties. For only allowing certain components to access certain services, we require **access control**. For preventing un-authorised requests by altering legitimate ones, we have to ensure the **integrity** of all requests and responses. Also **non-repudiation** addresses the problem of unauthorised use of services. **Confidentiality** is a general requirement in networks and is not specific to service discovery.

Guaranteeing availability is as well the task of the service discovery implementation as well as of the underlying network layers. In order to prevent flooding components with non-sense service registrations (and thus provoking an overflow in data structures or just make finding the legitimate services difficult), the implementer could limit the number of services provided by a component. The classical denial-of-service issue cannot be helped by the discovery protocol but must be addressed by the network layer.

As long as all users that are authorised to use the network are concurrently authorised to use service discovery, access control is implicitly covered by the network protocol. Where a more granular assignment of per-

mission is needed, either the service discovery architecture has to enable service managers and service cache managers to identify the service user and to handle per-user-permissions or the services themselves have to perform an identity check before allowing access. The latter case would not prevent unauthorised service users from discovering services but only from using them.

Authenticity, integrity, non-repudiation and confidentiality can be achieved by generic mechanisms which normally rely on (asymmetric) encryption and digital signatures. The main challenge thereby is the secure distribution of keys (see the presentation on the "Evaluation of distributed trust concepts" by Johann Niclas of this course). Special attention has to be paid on the question of what fields of requests and responses are signed, e.g. not including or not signing time-stamps or sequence numbers would enable an attacker to replay packets that would not be recognised as non-legitimate.

3. What Reality Looks Like

As mentioned previously, the designer of a service discovery architecture has to make a trade-off between security and ease of use. The protocols presented in chapter IV mainly rely on security mechanisms of lower network layers and designate very few mechanisms specific to service discovery.

SLP2 only offers an optional authentication feature which is based on asymmetric cryptography and provides thus only a very basic security mechanism. The administrator has to establish trust relationships by manually supplying components with public and private keys. It is remarkable that only service managers and service cache managers have to authenticate to the service users but not the other way around, i.e. restricting access to services is not subject to SLP2. An *Authentication Block* is introduced to hold the signature and other relevant information. A detailed analysis of SLP2 security can be found in [5].

Since Jini is embedded in the Java environment it profits from the Java platform security model which is implemented in the Java Virtual Machine (JVM) and that every Java program is subject to. Java also supports the basic network security mechanisms authentication, integrity and confidentiality. An unsolved security issue up to now is the mobile code that serves as a proxy for accessing services, though. First, we have a mutual authentication and authorisation problem, i.e. the client must be able to restrict access from the downloaded code to its local system. Second, since the service user does not access the service directly but through the proxy, it cannot control what the proxy does exactly. Third, integrity can so far only be granted for messages, not for objects that are sent over the network. The *Davis Project* is currently elaborating a solution for these problems (see [14]).

Since Salutation addresses home and small office users, the only security feature it provides is an optional authentication with usernames and passwords to restrict access to services.

UPnP does not describe any security mechanisms ("Security considerations: to be determined" [8]). Transport Layer Security (TLS) cannot be used since it requires TCP as transport protocol.

Bluetooth SDP relies on Bluetooth's security mechanisms which comprise usage protection and information confidentiality on both the application and the link layer. Thus access is granted to all devices that successfully logged on to the network and encryption for service discovery and for access to services is provided by the link layer.

VI. Summary and Outlook

Since future networks will be much more dynamic than traditional (wired) ones, peer-to-peer approaches and with it service discovery will gain more and more importance. The goal of service discovery mechanisms is to enable software components to find each other on such highly dynamic networks (like mobile ad-hoc networks). Other than arranging advertising and discovery, consistency maintenance and failure handling are important challenges to service discovery.

Currently there are five auspicious protocols, each one eligible in different environments: SLP2, Jini, Salutation, UPnP and Bluetooth SDP. Probably all of them will continue to co-exist, since each one has unique characteristics indispensable for certain applications. It is very probable, though, that more architectures will be developed for more sophisticated and critical scenarios like military operations.

Security is an issue that has not yet been solved in a completely satisfactory way in any of the protocols presented. It is thus expected that future versions will include improvements here.

References

[1] Christian Bettstetter, Christoph Renner: "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", In Proc. 6th EUNICE Open European Summer School: Innovative Internet Applications, Twente, Netherlands, 2000, http://citeseer.nj.nec.com/bettstetter00comparison.html

[2] Christopher Dabrowski, Kevin Mills: "Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach", In Proceedings of Working Conference on Complex and Dynamic Systems Architectures, Brisbane AU, 2001, http://www.itl.nist.gov/div897/ctg/adl/sdp_projectpage.html

[3] Christopher Dabrowski, Kevin Mills: "Understanding Self-healing in Service-Discovery Systems", In ACM Workshop on Self-Healing Systems, Charleston SC USA, 2002, http://www.itl.nist.gov/div897/ctg/adl/sdp_projectpage.html

[4] Erik Guttman: "Service Location Protocol: Automatic Discovery of IP Network Services", In IEEE Internet Computing, volume 3, number 4, pages 71-80, 1999, http://citeseer.nj.nec.com/guttman99service.html

[5] Marco Vettorello, Christian Bettstetter, Christian Schwingenschlögl: "Some Notes on Security in the Service Location Protocol Version 2 (SLPv2)", In Proceedings of Workshop on Ad hoc Communications, in conjunction with 7th European Conference on Computer Supported Cooperative Work (ECSCW'01), Bonn Germany, 2001, http://citeseer.nj.nec.com/vettorello01some.html

[6] E. Guttman, C. Perkins, J. Veizades, M. Day: “Service Location Protocol, Version 2 (RFC 2608)”, 1999, http://www.faqs.org/rfcs/rfc2608.html

[7] Sun Microsystems: “Jini Technology Core Platform Specification”, 2003, http://wwws.sun.com/software/jini/specs/core2_0.pdf

[8] Microsoft Corporation: “Universal Plug and Play Device Architecture, Version 1.0” , 2000, http://www.upnp.org/download/UPnPDA10_20000613.htm

[9] Björn H. Gerth: „Service Discovery in Home Environments" (slides), Braunschweig, 2003, http://www.ibr.cs.tu-bs.de/lehre/ws0203/skm-svs/slides/Service%20Discovery%20in%20Home%20Environments.pdf

[10] "Specification of the Bluetooth System, Version 1.2, Vol. 3, Part B: Service Discovery Protocol (SDP)", 2003, available at http://www.bluetooth.org/spec

[11] L. Küderli, M. Mayer: "Peer-to-Peer und andere Ad-Hoc-Dienste" (Hauptseminar Moderne Trends in Kommunikationsnetzen, TU München), München, 2003

[12] M. Fahrmair, M. Hutsteiner, T. Jöhnk, et. al.: „Developing Ad-Hoc Component Systems for Mobile Computing" (Lehrstuhl für Software und Systems Engineering, TU München), München, 2001

[13] Hollick, Matthias: „Security Awareness in Service Discovery for Multimedia Collaboration", Darmstadt, 2001, http://www.acm.org/sigmm/mm2001/ep/hollick/

[14] Venners, Bill: "A Conversation with Bob Scheifler", Sunnyvale (CA), 2002, http://www.artima.com/intv/jinisecu.html