Journal of Universal Computer Science, vol. 12, no. 3 (2006), 340-356
submitted: 30/10/05, accepted: 15/1/06, appeared: 28/3/06 © J.UCS

Secure Service Discovery based on Trust Management for ad-hoc Networks[1]

Celeste Campo, Florina Almenárez, Daniel Díaz, Carlos García-Rubio, Andrés Marín López
(Dept. Telematic Engineering - University Carlos III of Madrid, Spain
{celeste, florina, dds, cgr, amarin}@it.uc3m.es)

Abstract: In ad-hoc networks, mobile devices communicate via wireless links without the aid of any fixed networking infrastructure. These devices must be able to discover services dynamically and share them safely, taking into account ad-hoc networks requirements such as limited processing and communication power, decentralised management, and dynamic network topology, among others. Legacy solutions fail in addressing these requirements.

In this paper, we propose a service discovery protocol with security features, the Secure Pervasive Discovery Protocol. SPDP is a fully distributed protocol in which services offered by devices can be discovered by others, without a central server. It is based on an anarchy trust model, which provides location of trusted services, as well as protection of confidential information, secure communications, or access control.

Key Words: ad-hoc networks, service discovery protocol, security, trust
Category: C.2.2, C.4

1 Introduction

Recent advances in microelectronic and wireless technologies have fostered the proliferation of small devices with limited communication and processing power. They are what are known as "pervasive systems". Personal Digital Assistants (PDAs) and mobile phones are the more "visible" of these kinds of devices, but there are many others that surround us, unobserved. For example, today most household appliances have embedded microprocessors. Each one of these small devices offers a specific service to the user, but thanks to their capacity for communication, in the near future they will be able to collaborate with each other to build up more complex services. In order to achieve this, devices in such "ad-hoc" networks should dynamically discover and share services between them when they are close enough.

In ad-hoc networks composed of limited devices, it is very important to minimise the total number of transmissions, in order to reduce battery consumption of the devices. It is also important to implement mechanisms to detect, as soon as possible, both the availability and unavailability of services produced when

[1] This work has been partially supported by the UBISEC (Sixth Framework Program, Contract no. 506929), Everyware (MCyT TIC2003-08995-C02-01) and Easy Wireless (ITEA ip03008) projects.

a device joins or leaves the network. Security in these networks is also critical because there are many chances of misuse both from fraudulent servers and from misbehaving clients.

In this paper, we propose a service discovery protocol with security features, the Secure Pervasive Discovery Protocol (SPDP). SPDP is a fully distributed protocol in which services offered by devices can be discovered by others, without a central server. It provides location of trusted services, as well as protection of confidential information, secure communications, identification between devices, or access control, by forming a reliable ad-hoc network.

The paper is organised as follows: section 2 enumerates the main service discovery protocols proposed so far in the literature, we will see that none of them adapts well to ad-hoc networks. Section 3 presents our secure pervasive discovery protocol, SPDP, with its application scenario, and description of the algorithm. In section 4 we describe the underlying trust model as security support. In section 5 we present the simulation results comparing SPDP with other services discovery protocols. Finally, we conclude with some conclusions and future work.

2 Related Works

Dynamic service discovery is not a new problem. There are several solutions proposed for fixed networks, with different levels of acceptance, like SLP [RFC2608, 1999], Jini [Sun, 1999] and Salutation [Miller and Pascoe, 2000]. More recently, other service discovery protocols, specifically designed for ad-hoc networks, have been defined, some tied to a wireless technology (SDP for Bluetooth [SDP, 2001], IAS for IrDA [IrDA, 1996]), others that jointly deal with the problems of ad-hoc routing and service discovery (GSD [Chakraborty et al., 2002], HSID [Oh et al., 2004]), and others that work at the application layer of the protocol stack (DEAPspace [Nidd, 2001], Konark [Helal et al., 2003], and the post-query strategies [Barbeau and Kranakis, 2003]). Only a few protocols have built-in security, the most important are SSDS [Czerwinski et al., 1999] and Splendor [Zhu et al., 2003].

However, these solutions can not be directly applied to an ad-hoc network, because they were designed for and are more suitable for (fixed) wired networks. We see three main problems in the solutions enumerated:

– First, many of them use a central server, such as SLP[2], Jini and Salutation. It maintains the directory of services in the network and it is also a reliable entity upon which the security of the system is based.

[2] SLP supports a distributed mode, but usually the implementations use centralised mode

An ad-hoc network cannot rely upon having any single device permanently present in order to act as central server, and furthermore, maybe none of the devices present at any moment may be suitable to act as the server.

- Secondly, the solutions that may work without a central server, like SSDP, are designed without considering the power constraints typical in wireless networks. They make an extensive use of multicast or broadcast transmissions which are almost costless in wired networks but are power hungry in wireless networks.
- Thirdly, security issues are not well covered. SSDS provides security in enterprise environments but may not work in ad-hoc networks with mobile services. Splendor does not provide certificate revocation and trust models of PKIs. They both depend on trustworthy servers and they propose solutions which are provided at the IP level.

Accepting that alternatives to the centralised approach are required, we consider two alternative approaches for distributing service announcements:

- The “Push” solution, in which a device that offers a service sends unsolicited advertisements, and the other devices listen to these advertisements selecting those services they are interested in.
- The “Pull” solution, in which a device requests a service when it needs it, and devices that offer that service answer the request, perhaps with third devices taking note of the reply for future use.

In ad-hoc networks, it is very important to minimise the total number of transmissions, in order to reduce battery consumption. It is also important to implement mechanisms to detect as soon as possible both the availability and unavailability of services produced when a device joins or leaves the network. These factors must be taken into account when selecting between a push solution and a pull solution.

The DEAPspace algorithm is the only service discovery protocol, listed above, that tries to minimise the total number of transmissions. It uses a pure “push” solution and each device periodically broadcast its “world view” although none of them has to request a service.

3 SPDP: Secure Pervasive Discovery Protocol

In this paper we propose a new service discovery protocol, the Secure Pervasive Discovery Protocol (SPDP), which merges characteristics of both pull and push solutions to improve the performance of the protocol. Also, SPDP provides security based on an anarchy trust management model. Such trust management model does not require neither a central trusted server nor a hierarchical architecture, so it is suitable to overcome the challenges imposed by ad-hoc networks such as no central management, no strict security policies and highly dynamic nature (see section 4).

The Secure Pervasive Discovery Protocol (SPDP) is intended to solve the problem of enumerating the services available in ad-hoc networks, composed of devices with limited transmission power, memory, processing power, etc. Legacy service discovery protocols use a centralised server that listens for broadcast or multicast announcements of available services at a known port address, and lists the relevant services in response to enquiries. The protocol we propose does away with the need for the central server. Ad-hoc networks cannot rely upon having any single device permanently present in order to act as central server, and further, none of the devices present at any moment may be suitable to act as the server.

One of the key objectives of the SPDP is to minimise battery use in all devices. This means that the number of transmissions necessary to discover services should be reduced as much as possible. A device announces its services only when other devices request the service. Service announcements are broadcasted to all the devices in the network, all of which will get to know about the new service simultaneously at that moment, without having to actively query for it.

In addition, SPDP allows sharing services safely, through an underlying trust management model between devices, which allows us to store service information from other "alleged" trusted service agents and later to use them if such information is really authentic and upright.

Currently, the security support provided by service discovery protocols are focused on authentication, integrity, and confidentiality [RFC2608, 1999] [Czerwinski et al., 1999] [Zhu et al., 2003]. Even more, some of them include authorisation services as part of the discovery [Zhu et al., 2003]. Such support is based on IPSec [Kent and Atkinson, 1998] or traditional PKI in the last case. However, these security services could be not necessary for the discovery, but they could cause energy and processing consumption. Protecting both energy and processing consumption is a very essential issue for devices with limited capabilities. So we have considered providing basic security services to prevent certain attacks (i.e. DoS, false announcements, and false services) and to avoid the sending of unnecessary messages.

In the remainder of this section, we present the application scenario for SPDP and some considerations to be taken into account. Then, we will formally describe the algorithm used to implement it.

3.1 Application scenario

Let's assume that there is an ad-hoc network, composed of `D` devices, each device offers `S` services, and expects to remain available in this network for `T` seconds. This time `T` is previously configured in the device, depending on its mobility characteristics.

Each device has an SPDP User Agent (SPDP_UA) and an SPDP Service Agent (SPDP_SA). The SPDP_UA is a process working on the user's behalf to search information about services offered in the network. The Service Agent SPDP (SPDP_SA) is a process working to advertise services offered by the device. The SPDP_SA always includes the availability time `T` of its device in its announcements.

Each device has a cache associated which contains a list of the services that have been heard from the network. Each element `e` of the cache associated to the SPDP_UA has three fields: the service description, the service lifetime and the service expiration time. The service expiration time is the time it is estimated the service will remain available. This time is calculated as the minimum of two values: the time the device has promised to remain available, and the time the server announced that the service would remain available.

Entries remove themselves from the cache when their timeout elapses. With regard to security, each device handles a list of reliable devices and the trust degree associated with them. Trust helps devices to limit their cache size; services from untrusted devices are not stored in the cache. Depending on the trust degree, a device decides to store the service offered by a device on its cache. When the devices access services, devices with biggest trust degree are selected in the first place.

3.2 Algorithm description

The SPDP has two mandatory messages: `SPDP Service Request`, which is used to send service announcements and `SPDP Service Reply`, which is used to answer a `SPDP Service Request`, announcing available services. SPDP has one optional message: `SPDP Service Deregister`, which is used to inform that a service is no longer available.

Now, we will explain in detail how SPDP_UA and SPDP_SA use these primitives.

3.2.1 SPDP User Agent

When an application or the final user of the device needs a service of a certain type, it calls its SPDP_UA. In order to support different application needs, in SPDP we have defined two kinds of queries:

– **one query–one response** (1/1): the application is interested in the service, not in which device offers it.

– **one query–multiple responses** (1/n): the application wants to discover all devices in the network offering the service. In this kind of query, we introduce a special type of service, named `ALL`, in order to allow an application to discover all available services of all types in the network.

Both types of query use the same message, `SPDP Service Request`. A flag in the header of the message indicates if it is 1/1 or 1/n.

In **one query–one response** queries (see Figure 1 `searchAny`), the SPDP_UA searches for a `service_type` in the list of local services and in its cache. If it is found, the SPDP_UA gives the application the corresponding service description, without any network transmission. If it is not found, the SPDP_UA broadcasts a `SPDP_Service_Request` for that service, waiting `CONFIG_WAIT_RPLY` seconds for replies. If no reply arrives, the SPDP_UA answers to the application that the service is not available in the network. If some reply arrives, the SPDP_UA gives the application the service description received.

In **one query–multiple responses** queries (see Figure 1 `search`), the SPDP_UA makes a list of known services of the type specified, that is, a list of the ones offered locally or stored in its cache (all the services if the service type is `ALL`). Then, it sends a `SPDP_Service_Request` including this list. It waits `CONFIG_WAIT_RPLY` seconds for replies and returns to the application the list of known services plus, if any replies arrived, the service descriptions received.

SPDP_UAs in all devices are continually listening on the network for all types of messages (requests and replies) and updating their caches with the services announced in them. Moreover, the device's cache has a limited size. Whenever a `SPDP Service Reply` announcing a service is received, the SPDP_UA updates its cache accordingly.

It is not necessary to sign `SPDP Service Request` messages. There is another mechanism within the trust management model to protect from multiple false `SPDP Service Request` messages (see section 4.2), provoking a denial of service attack. Nevertheless, `SPDP Service Reply` messages are signed. The signatures are verified if the user requires to use the service announced by a specific peer. However, taking the dynamic nature of ad-hoc networks into account, users can define different policies to specify when the signature verification is needed. For instance, in a hostile environment, the user could verify the messages each time that they are received, whereas in a secure environment, message verifications would take place only when false announcements are detected. If the message is not authentic or upright, then the user removes of his/her cache the announcements from such peer and warns the presence of a malicious user. In addition to this, service descriptions can also be signed for integrity guarantee.

```
searchAny(service_type) {
      foreach (s ∈ Local)
        if (s.type==service_type) return(s);
      foreach (e ∈ Cache)
        if (e.type==service_type) return(e);
      broadcast(SPDP_Service_Request(service_type, 1/1));
      set_timer(CONFIG_WAIT_RPLY, EXPIRED);
      service_remote = hear_network(SPDP_Service_Reply);
      update_cache(service_remote);
      return(service_remote);
  EXPIRED:
      return(NULL);
}

search(service_type) {
      foreach (s ∈ Local)
        if (s.type==service_type) OR (service_type==ALL) known_services+=s;
      foreach (e ∈ Cache)
        if (e.type==service_type) OR (service_type==ALL) known_services+= e;
      broadcast(SPDP_Service_Request(service_type, 1/n, known_services));
      set_timer(CONFIG_WAIT_RPLY,EXPIRED);
      loop (forever)
        service_list+= hear_network(SPDP_Service_Reply);
  EXPIRED:
      update_cache(service_list);
      return(service_list + known_services);
}
```

```
update_cache(list) {
  foreach (l ∈ list)
    if (trust_degree(l.IP) ≥ 0.5)
       Cache += l;
}

trust_degree(ip) {
  if ∃ trust degree of l.IP return this
  else
    goto trust_formation
}
```

Figure 1: SPDP_UA pseudocode implementation

Moreover, the device's cache has a limited size. When an SPDP_UA hears a new announcement but the cache is full, it deletes the service entry offered by the device with less trust degree or less expiration time.

3.2.2 SPDP Service Agent

The SPDP_SA advertises services offered by the device. It has to process `SPDP Service Request` messages and to generate the corresponding `SPDP Service Reply`, if necessary.

In order to minimise the number of transmissions, the SPDP_SA takes into account the type of query made by the remote SPDP_UA. When a SPDP_SA receives a `SPDP Service Request 1/1` (see Figure 2), it checks whether the requested service is one of its local services. In that case, a `SPDP Service Reply` is scheduled for a random time, inversely proportional to the availability time of the device. During this time, if another reply to the same SPDP request is heard, the reply is aborted as the remote SPDP_UA will just pass the first service to the application and discard any others. If the timer expires and no reply has been heard, the reply is sent.

The algorithm awards the more static devices with more opportunities for answering requests. Therefore the algorithm gives higher priority to answers coming from devices with longer estimated availability.

```
receive(SPDP_Service_Request(service_type, 1/1)) {
      foreach (s ∈ Local)
         if (s.type==service_type) new_services_list+= s;
      if (new_services != NULL) {
         set_timer(generate_random_time($\frac{1}{T}$),EXPIRED);
         loop (forever) {
            hear_network(SPDP_Service_Reply);
            exit;
         }
   EXPIRED:
      broadcast(SPDP_Service_Reply(new_services_list));
      }
}

receive(SPDP_Service_Request(service_type, 1/n, known_services)) {
      foreach (s ∈ Local)
         if (s.type==service_type) OR (service_type==ALL) new_services_list+= s;
      foreach (e ∈ Cache)
         if (e.type==service_type) OR (service_type==ALL) new_services_list+= e;
      new_services= new_services - (new_services ∩ known_services);
      if (new_services_list != NULL) {
         set_timer(generate_random_time($\frac{1}{T*new\_services\_list.length}$),EXPIRED);
         loop (forever)
            service_list+= hear_network(SPDP_Service_Reply);
         update_cache(service_list);
   EXPIRED:
         if not (new_services_list ⊆ service_list) {
            new_services_list= new_services_list - (service_list ∩ new_services_list);
            broadcast(SPDP_Service_Reply(new_services_list));
         }
      }
}
```

Figure 2: SPDP_SA pseudocode implementation

When a SPDP_SA receives a `SPDP Service Request 1/n`, (see Figure 2), it checks whether the requested service is one of its local services, or if it is in the cache. If so, it generates a random waiting time, inversely proportional to the availability time of the device and the number of known services. During this time, the SPDP_SA listens the network for any `SPDP Service Reply` of the same request and it updates its cache accordingly. When the timer expires, if the SPDP_SA knows about some additional devices offering this type of service that have not been announced yet, it sends its `SPDP_Service_Reply`.

So, the more time the device is able to offer the service and the bigger the cache, the higher the probability of answering first. We suppose the device with the highest availability time and the bigger cache is the one with the most accurate view of the world.

In certain cases, it is possible to detect when a device is switched off or it roams to other network. If so, the SPDP_SA of the device has to send a `SPDP Service Deregister`, listing all its local services, before switching off or roaming. When a SPDP_UA hears this message, it must remove the services listed from its cache if the user is trusted. For this, `SPDP Service Deregister` messages should be signed.

When a device tries to access a service listed in its cache and the service is down, it may also use the `SPDP Service Deregister` message to inform the rest of the network that this service is no longer available. The device that receives the message may delete the entry from the cache, depending on the trust degree of the device that send this message.

4 Decentralised Trust Management Model

As we before mentioned, ad hoc networks imposed several challenges: a) they do not require the existence of any fixed infrastructure, b) they may operate in a standalone fashion, c) they can be very dynamic (changing topology), and d) the nodes have limited processing power and battery lifetime. In addition, such nodes can belong to different trust domain, that is, they can be unknown and do not have any previous configuration about each other. We based on these requirements, we will see how to provide a secure ad hoc network in a dynamic way, through degrees of trust associated with every device.

In the last decades, some trust management models such as those proposed in [Marsh, 1994], [Beth et al., 1994], [Blaze et al., 1996], [Abdul-Rahman and Hailes, 1997], [Jøsang and Knapskog, 1998] have been defined, in order to establish trust relationships between peers. Other models have been defined for public key infrastructures such as [Zimmermann, 95] and [Maurer, 1996]. These models present characteristics that are not suitable for ad-hoc networks, which can be summarised so: (a) they do not take the dynamicity of the user into account, therefore, they do not consider that the entities interacting are autonomous and mobile (b) they could present scalability problems because they define trust for each specific situation (or service) (c) they do not define a dynamic trust model over time, and finally (d) they have a complex management system to be deployed in limited devices. For these reasons, we have defined our own decentralised trust management model.

In our model devices are autonomous entities which act on behalf of a user, organization, etc. This way, the trust relationships are established, implicitly, between them, for instance, between a user agent A and a service agent B. This model is very simple indeed since, unlike SSDS and Splendor, the trust relationships are established only between two components and do not require manual configuration.

A trust relationship is not associated with a specific service, it means instead a belief that one entity has about another one based on past experiences, knowledge about entity's nature, or recommendation from trusted third peers. This belief represents how an entity will behave, implying a potential hazard; therefore, it is fully subjective. Each device handles a list of trustworthy users and untrustworthy users. These users are identified using a unique identifier, for instance their public key. The user's public key is also used to verify digital signatures.

The trust relationships are expressed using fuzzy logic rather than the usual Boolean logic or deterministic values. 0 and 1 are extreme cases, but intermediate values are also possible, for instance, 0.5 could be considered as an ignorance value. These trust relationships fulfil four properties: (1) *reflexive*, every devices trust on itself, (2) *asymmetric*, a trust relationships established between A and B is different from one established between B and A, (3) *conditionally transitive*, if A trusts on B and B trusts on C, then A conditionally trusts on C, and (4) *dynamic*, trust changes over time according to the user's behaviour. User's behaviour includes to send authentic and upright requests, to answer requests in a right way, to announce true services, to cooperate in order to maintain the security in the environment, etc.

4.1 Starting a trust relationship

At the beginning, new devices have no evidence of past experiences to establish an initial trust relationship. They could establish a new trust relationships based on: knowledge about entity's nature, recommendations, or applying trust rules. The first and third options belong to a personal opinion (direct trust) and the second one belongs to opinions from trusted third parties (indirect trust).

– Two entities can establish a new trust relationship in a direct way, because the user previously knows another user from the physical world, so he/she could manually configure the new relationship and associated it a new trust value. Otherwise, if the users are unknown and there are no recommendations, then a decision is taken based on trust rules, for instance, in an average environment the trust rule allows to assign the ignorance value (0.5) to unknown users.

– When the users willing to communicate with each other are unknown, recommendation requests are sent to nearby peers. In this way, we benefit from existent common knowledge in the environment. Recommendation replies are sent when there already exist trust relationships between some devices. Such replies are only accepted if they come from trusted peers. We consider trusted peers those who have a trust value greater than a configurable threshold.

With the recommendations, we calculate an initial trust value, applying the weighted average operator. This operator is simple, effective, and takes also the source's trustworthiness into account.

In order to facilitate off-line recommendations, because the presence of trusted peers is not always possible, we assume that certificates issued by third trusted parties are a recommendation mechanism, too. In this case, the recommendation value would be α multiplied by an uncertainty factor. Such uncertainty factor tries to capture our ignorance since the certificate was issued to the certificate is verified.

4.2 Sharing trust information

Trust information or recommendations are shared between trustworthy devices through a recommendation protocol, which has been designed for adhoc environments. This protocol has three messages: `Recommendation Request`, `Recommendation Reply` and `Recommendation Alert`.

`Recommendation Request` is used to request recommendations about other devices. By using `Recommendation Reply` we answer to this request. When we detect an attack from other device, then we use `Recommendation Alert` in order to warn all devices within the network about it. For example, when a device sends us multiple `SPDP Service Request` messages during a very short period of time. In this case, the device is considered untrustworthy, because it is attempting to perform a denial of service attack.

Figure 3 shows the algorithm to request and reply recommendations. When we detect an unknown device (target) we broadcast a `Recommendation Request` to other devices. Then, we wait a time `x` for the replies. When recommendations are received, we recalculate the trust value taking into account only the replies from trusted devices (recommenders).

Figure 4 shows how alerts are sent and received when the presence of a malicious peer is detected. For that, the trust model includes an action monitor, which identified anomalous behaviour in accordance with certain patterns. As `Recommendation Request`, we broadcast `Recommendation Alert` to other devices. If we receive it, then we perform an action depending on our trust on the sender.

```
**recommendation_request(trgtId)** {
  broadcast(RecommendationRequest(rqstId, type, rqstrId, trgtId, nstamp));
  $Total_{rec} = 0$;
  timeout (x, EXPIRED);
  loop {
    R = hear_network(RecommendationReply(rqstId, rcdrId, trustInf, nstamp);
    if $T_{recommender} \geq \alpha$ {
      if isTrustValue(Type) {
        Trust += trustInf*$T_{rcdr}$;
      }
      else {
       Trust += calculateRecommendationValue(trustInf,$T_{rcdr}$);
      }
      $Total_{rec}$ ++;
    }
    if (expired x) exit;
  }
  EXPIRED:
  if ($Total_{rec}$ == 0) Trust = ignorance_value;
  else Trust = $\frac{1}{Total_{rec}}$*Trust;
  return Trust;
}
 }

**send_recommendation_reply()** {
  A = hear_network(RecommendationRequest(rqstId, type, rqstrId, trgtId, nstamp));
  if known(trgtId) {
    if isTrustValue(type) {
       TrustInf = search_trust_value(trgtId);
    }
    else {
        TrustInf = search_trust_information(trgtId);
    }
   unicast(RecommendationReply(rqstId, rcdrId, trustInf, nstamp);
 }
}
```

Figure 3: Recommendation Protocol pseudocode implementation

4.3 Evolving trust values

As we mentioned previously, trust learning is gradual and dynamic, since the trust degree changes over time. In fact, it is often a consequence of a complex set of beliefs, perceptions and interpretations. Trust value changes according to positive and negative experiences in an specific context, therefore, we calculate a new trust value T_i taking into account both past and present, that is, the previous trust value T_{i-1} and the value of the interaction V_{ai} that represents device's behaviour. Both the previous trust value and the interaction value are weighted by a disposition factor, which allows assigning a weight to the past with respect to the present. Such disposition factor (ω) is within the interval $[0, 1]$, but it must be an intermediate value if we want to be a pragmatic or realistic peer.

V_{ai} is calculated in accordance with the action's weight affected by the historical behaviour. The historical behaviour allows us to build our evidence space that represents facts in the knowledge base of each device. These facts are useful

```
send_alert() {
  broadcast(RecommendationAlert(sdrId, trgtId, nstamp));
}

receive_alert() {
 A = hear_network(RecommendationAlert(sdrId, trgtId, nstamp));
 if unknown(sdrId) ignore(message);
 else if $T_{sdrId} < T_{trgtId}$ ignore(message);
     else if $T_{sdrId} \geq T_{trgtId}$ decrease($T_{trgtId}$);
}
```

Figure 4: Alert Recommendation Protocol pseudocode implementation

to identify trustworthy and untrustworthy devices. It is important to identify untrustworthy devices because mistrust is different from a simple absence of trust (ignorance).

The Figure 5 depicts the evolution of the trust value according to a variable behaviour, that means, the user begins with positive interactions, but later he/she performs some negative actions, then positive actions and negatives again, and finally he/she performs a few positive actions. In this figure, we can see that the trust is hard to gain and very easy to lose. The trust increase is inversely proportional to the number of negative interactions.

Finally, the growth in the trust assessment is each time bigger as much as the positive interactions increase.

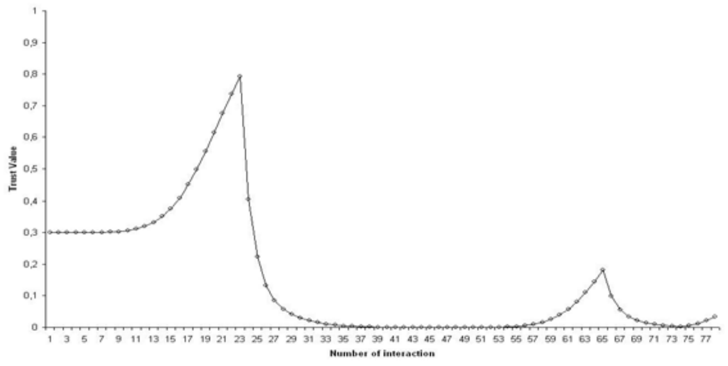

Figure 5: Trust Evolution

In this way, we establish a reliable ad-hoc network within which peers might discover trusted services minimizing the risk of deceit.

5 Evaluating the SPDP protocol

In this section we present a performance evaluation study of SPDP in a ubiquitous computing environment. We compare our protocol with the theoretical distributed approaches, push and pull; because all the service discovery protocols defined in the literature are based on one of these approaches; and also we compare PDP with the service discovery protocol standard in Internet, SLP, and with UPnP's SSDP. This study was carried out through simulation using the well-known network simulator, NS-2. Our simulator is available in [Campo and Perea, 2004].

During the simulation, devices join the ubiquitous environment at random times, request and offer random services, and leave the network after a random time. The number of devices in the network varies over time, but its mean remains stationary. Random times follow exponential distributions, while random services follow uniform distributions. For simplicity we assume that each device offers just one service.

The parameters of the simulation are: the mean number of devices, the mean time they remain available in the network, the size of the caches, the mean time between service requests, and the total number of service types. The results of interests are: the number of messages (the number of messages transmitted in the network normalised to the number of service request), the service discovery ratio (the ratio of services discovered to the total number of services available in the network) and the error ratio (the ratio of services discovered that were not available in the network to the total number of services discovered).

Figure 6 shows the number of messages transmitted, the service discovery ratio and the error ratio, in a scenario with 20 devices, an average device life time ranging from 600 to 19200 seconds, a cache size of 100 entries, 5 different types of services, and each device requesting a random service every 60 seconds. The SPDP number of messages is quite under those obtained for SLP and for pull solutions, while keeping the same service discovery ratio and error rate of them.

6 Conclusions and Future Work

Ad-hoc networks are becoming increasingly common thanks to the development of mobile device technology. When a device connects to an ad-hoc network, it wants to know the services offered by the network and in turn it may offer its own services. Client applications in the device want to discover trustworthy services automatically, while server applications want to be used by trustworthy clients that will not misuse or attack them. Additionally, secure network communication is also an important issue. These goals are carried out by SPDP.

SPDP is a suitable service discovery protocol for ad-hoc networks since:

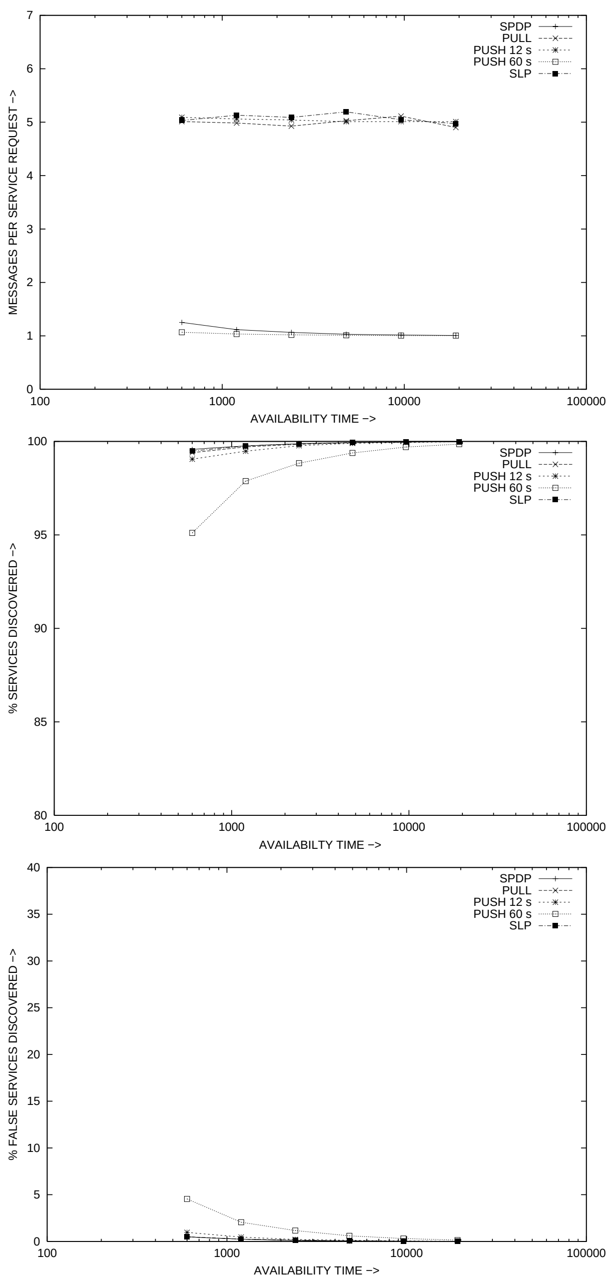

Figure 6: Comparison of SPDP with others protocols.

- It is based on a distributed open architecture, therefore, it does not require central servers;
- It has a simple architecture which contains only two type of components, user agents and service agents;
- It provides autonomous and mobile agents with a simple method for discovering services that are available;
- It minimises battery use in all devices since the number of transmissions necessary to discover services is reduced as much as possible;
- It integrates a security model in order to guarantee the security level required by devices. Security issues include authenticity, and data integrity based on a decentralised trust management model.

Thus, we fulfil the challenges imposed by ad-hoc networks.

We have built the SPDP protocol and software that uses it to discover the services offered in its surroundings in Java 2 Micro Edition (J2ME), using the Personal Profile of the Connected Device Configuration (CDC). This implementation has been tested successfully in Pocket PC Windows Mobile 2003 devices.

The security support of SPDP has been developed as an independent module, in order to provide security services to other kind of applications. The main component is the *"Trust Manager"*, which implements the basic functions of the model. This component is supported by the *"Recommendation Manager"*, which implements the recommendation protocol and the *"Monitor"*, which keeps watch over anomalous actions.

As future works, we are currently working on the implementation of SPDP in other devices without support of Java Virtual Machine, such as web-cams, to be integrated in a test-bed to obtain results based on real experiments.

References

[Abdul-Rahman and Hailes, 1997] Abdul-Rahman, A. and Hailes, S. (1997). A distributed trust model. In *Proceedings of the ACM Workshop on New Security Paradigms*, pages 48–60, Cumbria, United Kingdom. ACM SIGSAC, ACM Press.

[Barbeau and Kranakis, 2003] Barbeau, M. and Kranakis, E. (2003). Modeling and Performance Analysis of Service Discovery Strategies in Ad Hoc Networks. In *International Conference on Wireless Networks. ICWN 2003*, Nevada. Canada.

[Beth et al., 1994] Beth, T., Borcherding, M., and Klein, B. (1994). Valuation of trust in open networks. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS '94, Brighton, UK)*, number 875 in Lecture Notes in Computer Science, pages 3–18, Heidelberg, Germany. Springer-Verlag.

[Blaze et al., 1996] Blaze, M., Feigenbaum, J., and Lacy, J. (1996). Decentralized trust management. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, number 96-17, Oakland, CA. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[Campo and Perea, 2004] Campo, C. and Perea, J. C. (2004). Implementation of pervasive discovery protocol. http://www.it.uc3m.es/celeste/pdp/.

[Chakraborty et al., 2002] Chakraborty, D., Joshi, A., Yesha, Y., and Fini, T. (2002). GSD: A Novel Group-based Service Discovery Protocol for MANETS. In *4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002)*, Stockholm. Sweden.

[Czerwinski et al., 1999] Czerwinski, S. E., Zhao, B. Y., Hodes, T. D., Joseph, A. D., and Katz, R. H. (1999). An architecture for a secure service discovery service. In *Mobicom'99*.

[Helal et al., 2003] Helal, S., Desai, N., Verma, V., and Arslan, B. (2003). Konark: A System and Protocols for Device Independent, Peer-to-Peer Discovery and Delivery of Mobile Services. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(6):682–696.

[IrDA, 1996] IrDA (1996). Infrared data association link management 1.1.

[Jøsang and Knapskog, 1998] Jøsang, A. and Knapskog, S. J. (1998). A metric for trusted systems. In *Proc. 21st NIST-NCSC National Information Systems Security Conference*, pages 16–29.

[Kent and Atkinson, 1998] Kent, S. and Atkinson, R. (1998). Security architecture for the internet protocol (IPSec).

[Marsh, 1994] Marsh, S. (1994). *Formalising Trust as a Computational Concept*. PhD thesis, Department of Mathematics and Computer Science, University of Stirling. citeseer.ist.psu.edu/marsh94formalising.html.

[Maurer, 1996] Maurer, U. (1996). Modelling a public-key infrastructure. In Bertino, E., editor, *European Symposium on Research in Computer Security (ESORICS' 96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 325–350. Springer-Verlag.

[Miller and Pascoe, 2000] Miller, B. A. and Pascoe, R. A. (2000). Salutation service discovery in pervasive computing environments. Technical report, IBM.

[Nidd, 2001] Nidd, M. (2001). Service Discovery in DEAPspace. *IEEE Personal Communications*, 8:39–45.

[Oh et al., 2004] Oh, C.-S., Ko, Y.-B., and Kim, J.-H. (2004). A Hybrid Service Discovery for Improving Robustness in Mobile Ad Hoc Networks. In *The International Conference on Dependable Systems and Networks. DSN-2004*, Florence, Italy.

[RFC2608, 1999] RFC2608 (1999). Service location protocol, version 2 (RFC 2608).

[SDP, 2001] SDP (2001). *Bluetooth Specification v1.1, Part E: Service Discovery Protocol (SDP)*.

[Sun, 1999] Sun (1999). Jini Architectural Overview. White Paper.

[Zhu et al., 2003] Zhu, F., Mutka, M., and Ni, L. (2003). Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (Percom'03)*, pages 235–242. IEEE Computer Society.

[Zimmermann, 95] Zimmermann, P. R. (95). *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA.