Chapter 5

SERVICE DISCOVERY IN CASP

Introduction

Due to the nature of mobile ad hoc networks the proposed approach is implemented in a totally distributed way, without any central servers. The framework used peer-to-peer caching of service advertisements. It is assumed that every node participate in the service discovery process. Whenever a service provider decides to share the service it advertises the services. The advertisements are broadcasted in the network. Each node in the network is associated with a profile that determines the type of service it is interested in. Whenever a node receives the advertisement it stores and forwards to the next hop neighbour. As this neighbour is the same as that of the routing layer, the physical topology is maintained in the advertisement too. Any node that receives a message verifies if the header of this message matches its own interest profile and is not already stored also can be placed in the cache if so it puts it in its cache, else it replaces an uninterested message in the cache with the new message.

Periodic announcements

It is challenging to maintain accurate and valid service information and service state especially in mobile ad hoc networks where the inherent dynamism leads to frequent changes in service availability. It is the job of the service provider to periodically refresh that counter by re advertising the services. If two nodes are passing by each other, the time window of a possible contact gets slimmer with a higher node moving speed. Therefore, announcements are repeated in a periodic way so that two nodes detect each other inside of the contact time window and with additional spare time for any eventual message exchange.
Local cache

Each node in the mobile ad hoc network has a dedicated storage space service as a local cache of messages. There are two kinds of cache maintained

(i) Local Resource Pool – This is used to store the local services and resources that can be shared with other nodes.

(ii) Network Resource Pool – Stores the service descriptions sent by other nodes along with the information provided.

Since these spaces are necessarily limited, the cache removes messages that have reached their end of life and in case the cache grows full.

Cache Replacement Policy

Management of directory content relies on cache replacement policy. This architecture checks the new advertisement’s service if the service is the one in the list of services for the applications in the node, then it checks the existing list of nodes for the applications if any of the service is not needed by the application in the node, that is the candidate element for replacement. Each node will maintain the list of services needed by its application, when there is no space to store the new advertisement, the services which are not in the list are replaced by the new advertisement apart from this the entries are deleted then and there when the entries are timed out, or the explicit request from the providers.

Mobility as an advantage

By storing messages in the local cache, a node can serve as a mobile carrier for these messages while moving in the network. In an environment, a carrier node becomes the physical relay of information. The information can reach the distant node because of the relay nodes even if it does not come into the hoping limit specified by the advertiser.

Economical use of the wireless medium
The protocol is designed so as to maximize the message delivery ratio while using less wireless medium as far as the number and volume of exchanged data are concerned. As the messages cached are as par with the interest profile of the user most of the service request will be matched on the node itself and all the localized messages which are in the reach of the node in a few hop distance. Which is very much essential than storing all the messages in the network.

5.1 Proposed Service Discovery in CASP

This thesis name the framework proposed as CASP (Context Aware Service Provision system). The advertisement and query processing module of CASP is used for advertisement and discovery. The discovery protocol provides facilities to do all the steps described already before invocation such as creation of descriptions, advertisement, collecting the descriptors advertised and select the required provider based on the client request. It provides facilities for the provider to describe and advertise the services. Also it provides facilities to the clients to receive the advertisements group them and select the provider based on its need.

Generally there are three phases in service discovery:

- **Service advertisement**: nodes advertise the service in the device to other node, when a network is established or new node joins the network.

- **Service Request Processing and Propagation**: nodes request other nodes for services they are in need of.

- **Service maintenance and updates**: nodes keep on updating the latest state about the service by using hello packets and update the service list changes to other nodes in the network.

5.1.1 Service Advertisement Format and Processing
The functional entities for the execution of advertisement and caching activities are: (i) message creation (ii) message sending (iii) message receiving and the cache to store the advertisement. Matcher will compare the attributes in the request and the service descriptions stored. After describing the service properties, non-functional properties and provides context the description is enveloped in a descriptor that is advertised to interested client nodes. The descriptor is created as a message and advertised using underlying communication layer. The XML description is put in the message’s payload. The message header contains message id, propagation information. Various types of messages used in this thesis are listed in Table 5.1.

Protocol messages

The framework proposes a set of messages for data exchange between the service provider and the service requester. Two kinds of modules namely casp_useragent and casp_serviceagent are provided. To discover available services in the network, one can make use of the casp_useragent. Similarly the casp_serviceagent is used to advertise the services by the service provider.

A programmer creates the service’s binary, functional descriptions and non-functional descriptions. Let this be $S_1$. A provider agent hosts the service binary and creates the service descriptor according to the description and context properties. In order to advertise the services available in the provider node, it has to create advertisement message and broadcast it to the mobile ad hoc network. In the service description the provider has to specify the functional and non functional properties of the services. Non functional properties can be classified as static and dynamic properties
<table>
<thead>
<tr>
<th>Message</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>casp_advertisement</code></td>
<td>To Advertise the service availability</td>
</tr>
<tr>
<td><code>casp_hello</code></td>
<td>To send periodically to make the nodes aware of its presence</td>
</tr>
<tr>
<td><code>casp_service_deregister</code></td>
<td>To make the clients to close their session with this provider.</td>
</tr>
<tr>
<td><code>casp_update</code></td>
<td>To send update of the service parameters if any.</td>
</tr>
<tr>
<td><code>casp_service_reply</code></td>
<td>Reply the request of the service request received with descriptors</td>
</tr>
<tr>
<td><code>casp_service_request</code></td>
<td>To send Service to other nodes in the network</td>
</tr>
<tr>
<td><code>casp_service_invocation</code></td>
<td>To send the invocation request</td>
</tr>
<tr>
<td><code>casp_service_response</code></td>
<td>To send the result to the invocation request</td>
</tr>
<tr>
<td><code>casp_service_status</code></td>
<td>To send the status request to the service provider</td>
</tr>
<tr>
<td><code>casp_service_status_res</code></td>
<td>To send the status to client node</td>
</tr>
</tbody>
</table>

**Table 5.1 Different Message types**

Static properties are the one for which the values can be specified at the time of design itself. For the dynamic properties the values has to be captured at the execution time. For example the performance, memory requirement etc. can be specified at the time of designing. The properties like the availability, response time, distance (number of hops), battery power, moving speed etc. are dynamic in nature. These have to be captured at the execution time only. The dynamic attributes are the context attributes. The context may be service requesters’ context constraints and the service providers’ context requirements. The service provider’s context values can be captured and sent at the time of sending the advertisement. A detailed discussion on non functional properties has been presented in chapter 6.

The service provider will send its id along with the TTL which is the propagation information. This information is based on the type of the application. The type of the application will be specified in the application profile. If the TTL is high then the advertisement will reach
many nodes in the network, whereas if it is less, the reach ability of the service description will be limited. But at the same time if it is set a high value it will make the network full of advertisement making a bottleneck to other useful information exchange.

```
messageType="advertisement"
sender_id= forwarding peer_id
provider_id= "16 digit number"
service_type="servicetype"
portable="yes"
hopcount= 0
adv_num =1
ttl=3
time_of _expiry="time-stamp"

payload (service description)
```

*Figure. 5.1 Structure of Advertisement message*

The service descriptions are retrieved from the local pool. The providers metrics details like battery power, speed, location, number nodes serviced will be accessed from the relevant stores. The dynamic resources details like are accessed using the low level functions and formulated. The details of the payload will be discussed in chapter 6. This advertisement will use broadcasting of lower layer protocol. The structure of the advertisement is provided in Figure 5.1. The pseudo code to create the advertisement is given in Figure 5.2.

During runtime the provider will form other types of messages and sends for the following using their corresponding Create functions. *caspcreateupdateservice()*, *caspcreatehello()*, *caspcreateresponse().*

```
void caspcreateadvertisement()
```
Create a new message body
Append the service description
Append the functional properties
Append the non functional properties
Append the context
Create new message header
Append the header attributes
Create the caspmessage using the message body and the header
Send the message to the network by calling tonetworklistener(message_adv,request);
return

Figure 5.2 Pseudo Code for Creating the Advertisement

Once the advertisement is created it will be sent to the other nodes in the mobile ad hoc network. These advertisements will be stored in the pool. The request pool will be checked if any of the requests can be served by this advertised content. And after the duplication check the advertisements will be stored. The pseudo code for processing the advertisement is presented in Figure 5.3.

---

void caspadvproc(message m)
If (m.messagetype="advertisement")
Access the service table with service_type
If ( no m.service_type from this provider)
If cache not full
  add m into the cache
else
  replace an entry with m
reduce ttl
if (ttl not equal to 0) broadcast m
return

---

**Figure 5.3 Pseudo code for advertisement processing**

In order to show how the information is stored in the cache, consider an example depicted in Figure 5.4 which specifies the physical network. Let P_1 to P_{13} are Service Provider nodes in the mobile ad hoc network. Assume that nodes P_1, P_9, P_4 and P_{11} are providing Service_i and P_1, P_5, P_4 and P_{12} are providing Service_j. Figure 5.5 specifies the overlay for two services. P_{11} – is node with the service service-i with highest rank. P_4 - is the node with next highest metric value and so on.
A provider may belong to more than one overlay if it provides more than one services. In this example provider 4 and 1 provides both service_i and service_j, they will be two overlays.

Once the cache is created Application or the developer can perform the following operations.

(i) \textit{void add (serviceentry[], message m,):} places the advertisement into the network resource pool. It sends the new_service_arrived event to the policy engine in order to further enforce policies if any for the new service.

(ii) \textit{serviceentry servicesearch(request r):} matches the local and network resource pool for a matching entry returns the service entry.

(iii) \textit{serviceentry[] servicesearchall (request r):} matches with the local and network resource pool and returns all the matching entries.

(iv) \textit{void remove (serviceentry[], service_type, message m):} – removes the advertisement m from the network resource pool. The removal of an advertisement from the list signifies that the service is no longer available. This occurs when a remote node sends explicit removal of its service, or the node moves out of range which is signified from the fact of non reception of the hello message.
(v)  `void delete (long provider_id)` – removes all the advertisements from this provider node.

### 5.1.2 Service Request Processing and Propagation

Whenever a node looks for specific service, a query is generated. Queries are compared with the incoming advertisements from neighbours and only forwarded to the ones that can reach the requested service within the specified number of hops.

**Design Choices**

There are two design choices, they are:

(i) *Request format* – The information can be sent as text strings, or XML etc. or other encoding methods. But these encoding methods need further processing of encoding and decoding. Though packet size in XML is higher than the encoded ones, this thesis uses this type of packets as XML has many useful features.

(ii) *Request Propagation* – Request for a service can be done in parallel or sequential way. In parallel query method a query will be forwarded to all the neighbours that show matches. Therefore all the requested information available in $d$ hops can be found simultaneously. It offers choices to end users. In contrast, queries are only forwarded to one corresponding neighbour at a time in sequential manner. If queried neighbour does not have the requested information the query need to be forwarded to another neighbour. Consequently only one reply can be received at once.

**Request format**

In CASP, the node which needs the service will send its request to the resource manager, it checks in the local resource table and in the network resource pool if not present in both it will communicate to the advertisement module to create request and send to other nodes in the mobile ad hoc network. The structure of request is presented in Figure 5.6.

The header attributes of the request message
• *message_type* – As many types of messages as there like advertisement, hello etc. it is needed to specify the message type as ‘request’

• *requester_id* - the requesting nodes address is also passed.

• *request_id* – Every request will have an id so that the duplicate request from alternative paths may be avoided. Every node stores the list of requests it received. If a node receives queries with same id from the same originator it drops the duplicate.

• *ttl* – This specifies the number of hops the request needs to be forwarded. It is used to restrict the query range.

• *service_type* – type of the service it is requesting.

  Time out is the timer used to count down the expiration of the querying period. When the node generates the request it also sets the time-out timer, when the timer reaches 0 the query is expired and will be informed to the application/client.

| message_type = "request" |
| requester_id = "16 bit number" |
| request_id = "integer" |
| ttl = 3 |
| service_type = "service type" |
| time_of_expiary = "time" |

**payload** *(mandatory service request attributes)*

*Figure 5.6 Structure of Request message*
Request Propagation

Following is the way how request propagation is implemented in CASP. The request messages from a node will be propagated to the node that provides the needed information. The issues to handle are (i) the queries should be directed to the node which has that information (ii) congestion due to excessive spreading has to be limited. The methods selected for query processing are: Target selection and multicasting with limited query range.

Target selection means targeting a specific node that is having the required information can be exploited at the application layer by using the information of the address and position of the providers. The service advertisements are cached and grouped based on the service_type. One group for each service type is maintained. Hash tables are used to point to the groups with the service_type as key. The application designer can specify the preference of the metrics they need, based on the requirements the nodes will be sorted. The assessment and selection of the list of providers is done by the ‘Matcher’.

When more than one matching provider is available, the node at the first position of the group will be selected. The routing table information is accessed whether currently the node can be reached;

The second method multicast with limited query range is used if the response cannot be received from the provider within the ‘response time’ then the request will be sent through to the group of nodes which are matching the requirements in the overlay with less rank. Whenever the service provider is ready to serve, both the client and server will maintain this detail in the form of a table.

Data structures used

*service_clients table*: Service_client table is maintained at the service-providing node and contains the service specific data related to the client it serves. The server, while processing the request, checks the client table and updates it if no entry for client node is found. It records the service type it provides to the client and stores information for future responses to the client.

*service_providers table*: Service_providers table at the requester node lists the address of each node from which it receives services. Along with the service type, it also records the
scope of the service-providing node. This table stores the information of the service providing node until the service is needed

Request Processing

In case of sequential search, the originating node checks the stored information about the services, if the service provider exists it will send request message only to the top ranked provider, if the reply is false then the provider in the next position will be sent a request message. It will do the search until all the possibilities are exhausted. If no match it will send a flood request to all its neighbours. The request timer times out then the query will be removed from the list and the application will be informed of the non availability of the service.

The service request may be received by one of these nodes

- Service Provider
- Intermediate node with the cached description for this service

Processing at the provider nodes

If the request is received by the provider, it will send the descriptor of the service which will specify all the details related to invocation of the service, its input and output parameters if any, etc. the requester node will added in to the table service_client. The pseudo code for request processing at the service provider node is given in Figure 5.7.

```
void caspservicesearch (request r)
    get the serviceentry k from the local pool
    if (the functional properties of r and k are matching)
        if (the non-functional properties of r and k are matching)
            (i) create service_response message
                (ii) send it to the requester as unicast communication
            return
        else
            reply with negative response with the request_id
    return
```
Figure 5.7 Pseudo code at provider node when a request is received

Process at the intermediate nodes

Service request pattern is matched with the descriptors. If a match is made from the cached descriptor actual service request will be formulated and sent to the service provider. The pseudo code for request processing at the intermediate node is provided in figure 5.8(a) and pseudo code for request processing at the requester node is given in figure 5.8(b).

Process at the requester node

The received advertisements are store in the Network Resource Pool. In Purely decentralized overlays, the most appropriate approach is to cache whatever the node comes across. This may be alleviated by using the strategy that the most relevant services for the application will be provided from the application profile and the framework starts caching those services from the advertisements. CASP provides User agent to define list of services needed, discovery preferences. The node collects all the advertisement that it is interested in until the invocation time when the user selects a service out of his local directory. If the cache is full then the irrelevant service information will be deleted from the cache the space may be used for the relevant services if the one arrives. CASP provides get method for application developer to ask for needed services.

```c
void casprequestproc(request r)

if service_type exists go to its corresponding overlay

for (all the providers in the list)

  If (match(request r, provider list)

    send request to that provider

end for

done
```

else
When invocation is needed the application may call caspget(service_type) function. This function will call the matcher module which will be discussed in the next chapter and find the matching service from the local or from the remote cache. The service provider with the highest metric will be selected. Parameters of providers metric such as the battery power at the time of sending (BatP), number of services provided (NS) to other nodes, Moving speed (Speed) are considered to calculate the metrics. The provider’s metric will be calculated based on equation (5.1)

\[
Metric = W_1 \times Speed + W_2 \times BatP + W_3 \times NS
\]  

(5.1)

Where Wi represents the weight age given for the parameters. Depending upon the need different values may be given for each parameter. The service_provider table will be altered accordingly. The pseudo code for this function is shown in figure 5.9.

```c
void caspget(service_type)
{
    access the user and device profile values

    form the service request and preference descriptions from the application profile

    if (match in the local cache)
        access the reference and pass that to the application

        return

    else if (match in the remote cache)
        select among the available service provider ‘p’ based on the metrics

        create a message of type request ‘r’ and send to service provider p by unicast message to the provider by using casp_request

    return
}
```
Figure 5.8 (b) Pseudo Code at Requester Node when a Request is Received From the Application

**Choices to send reply**

There are three alternatives for request routing also reply. The first option is to maintain state routing information in the nodes. Nodes record where a request is received from and sent to. The reply of such a request is routed back to the query initiator based on that information. This can put extra burden on those nodes that store the routing information.

Alternatively, the reply path can be stored in the request messages themselves. This result in a large request packet size and increased transmission costs. In the first alternative the reply message is sent along the same path as the request. They both encounter the problem that the reply message gets lost, whenever a node along the path moves or disappears. However, they have the advantage that no extra routing functionality has to be present in the platform.

The third alternative is that the system can rely on an external routing protocol. This has the benefit of no consumption of extra storage room or bandwidth. It does not rely on the same paths. Even if the path to the requesting node has changed, the reply message can still reach the querying node as long as there is at least one path to that. However, this alternative requires extra functionalities of routing protocols. Since routing protocol is needed to set up a connection between the requesting and replying node to deliver the requested information, it could be a good option, which has been used in this thesis.

### 5.2 Service Update and Maintenance

It is challenging to maintain accurate and valid service information and service state especially in mobile ad hoc networks where the inherent dynamism leads to frequent changes in service availability. One approach is to maintain a hard state of services where a provider must
de-register its services before leaving the mobile ad hoc network. However, in mobile ad hoc networks where unpredictable disconnections occur assuming that a provider will be able to de-register its service before disconnecting is not realistic. The opposite approach is to maintain a soft state of services. In this case each service entry is associated with a time to live counter; upon expiry the service entry is automatically deleted. It is the job of the service provider to periodically refresh that counter by re advertising the services in the form of hello packet, so that the nodes will have up to date knowledge of the service availability. In this thesis both soft state and hard state are used to maintain consistency.

5.2.1 Soft-State Maintenance using ‘Hello’ Periodic Message

Nodes should be aware of the existence of their neighbours and be able to detect the appearance and disappearance of neighbours and their service information. In this thesis hello messages are used to keep nodes alert for changes in their environment. The hello message is the shortest message. Fixed time interval between consecutive hello messages is used. It is assumed that the same period is used in the entire network. Each node in the network knows this period.

A node sends out the hello messages periodically. The structure of hello message is shown in figure 5.9. A timer is used to determine the timestamp on when the ‘hello’ messages are sent. Whenever a hello message is sent out, the timer is set back to the hello period. When the timer expires the hello message is sent out by calling the method send_hello.

Whenever a node receives a ‘hello’ messages it should do one of the following actions. When a node receives a ‘hello’ message from an unknown neighbour it will send an update message to that node also store that information in the list of advertisements. It will update the latest timestamp in the stored advertisement of that node. The pseudo code for the hello packet processing is presented in figure 5.10.

A node removes a neighbour and its related information if it does not receive any hello message or update message from this neighbour within two consecutive hello periods.

```
message_type="hello"
provider_id ="16 bit number"
```
In `hello` messages the service providers send the list of available services for sharing, TTL value which is used to limit the scope. Since the parameters are sent already in the advertisement itself they need not be sent again. A facility to send new service advertisement is provided if it is not sent already as a piggyback type service description. This new service-type will be added to the list already available, the processing is similar to the processing of the services in the advertisement.

```c
void casphelloproc()
{
    for (all the entries for this provider_id)
    {
        if (message_type = "hello")
            for (all the services in this packet)
            {
                if (the service_type exists in the table)
                    update the time_of_expiary
                if (piggyback)
                    call the method for advertisement to add these services to the table
            }
            for (all the service without matching entry in the packet)
                delete the entry from the table
            for (all the out-of-date entries regardless of the server)
                delete the entries
    }
}
```
5.2.2 Service Update

It is a way to maintain a consistent list of services. If there is a need to change any of the parameters of the services already sent that can be done by the use of these packets. Service update packets will be generated by the server when the already published service has been modified, or the service descriptions have been changed.

<table>
<thead>
<tr>
<th>message_type = &quot;update&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender_id = forwarding peer_id</td>
</tr>
<tr>
<td>provider_id = &quot;16 digit number&quot;</td>
</tr>
<tr>
<td>hopcount = 0</td>
</tr>
<tr>
<td>ttl = 3</td>
</tr>
<tr>
<td>time_of_expiry = time-stamp</td>
</tr>
<tr>
<td>payload (service description)</td>
</tr>
</tbody>
</table>

**Figure. 5.11 Structure of Service update message**

```c
void casupdateproc()

for (all the entries for this provider_id)

    if (message_type = "update")

        for (all the services in this packet)

            if (the service_type exists in the table)

                replace the service descriptions

                update the time_of_expiry

            else

                add this service to the table

        for (all the out-of-date entries regardless of the service provider)
```
5.2.3 Service De-registering

This message is sent by the service providers so that the clients of the services of this server may find new alternative services for their continuous service usage. The framework sends this message only to the nodes that uses the services of this service provider. Other nodes will delete the entries of advertisement as the entries are timed out. Whenever the service provider decides to deregister it generates deregister message and send to the clients. The pseudo code for deregister processing is shown in figure 5.13.

The client when receives the notification of de-registration from any provider, it removes that provider’s details from the cache. The notification will be sent to the policy enforcer for further searching for services. The service reselection and rediscovery policy will be checked and enforced which will be discussed in chapter 7.

```c
void caspderegisterproc()
{
    go to the overlay of this service
    delete the entry from the list
    notify this to the module which takes care of service rediscovery
}
```

Figure 5.12 Pseudo Code for Processing Service Updation

Figure 5.13 Pseudo code for processing service deregister message
5.3 Implementation of CASP

This section presents a brief introduction to the development platform and the functions and function calls of the module in charge for service discovery of CASP. The framework is implemented using java 2 Micro Edition (J2ME). Java2 Micro Edition is available on almost all devices. J2ME provides light version of the most popular java libraries. It defines standard interfaces for exchanging messages over wireless networks. To facilitate development several emulators are available for programming, testing on standard PC.

This version requires a much smaller memory footprint than the standard or enterprise edition, while at the same time it is optimized for the processing power and I/O capabilities of small mobile devices. Java has been chosen because of its ubiquity, platform independence and the fact that it allows for dynamic code loading and off loading (even for dynamic code generation). The use of Java requires nodes to have the Java Runtime Environment (JRE) installed. Although this is relatively memory-hungry, our hands-on experience shows that even the resource-poor PDAs can comfortably support the execution of the JRE. XML handling necessitates a lightweight XML API and in that respect the kXML2 parser (http://kxml.sourceforge.net/) is used, especially targeted for J2ME.

Resource manager module receives the services that appear in the environment and creates advertisement for those services that can be shared by others. The choice of language selected is java which is motivated by its portability and its capability to provide a strong separation between the APIs and their implementations.

The Advertiser module of CASP forms the different types of caspmessage and sends them by invoking CASP methods. Each message has a caspserviceentry object which contains the payload consists of the advertisement in the form of compressed XML file. The advertisements are formed by the create method which will combine all the parts of the XML file and set the other propagation properties to the lower communication layer. Along with this payload other caspmessage properties like ttl will be formed by the caspcreateadvertisement method. Once the caspmessage is formed it will be sent to the network.
The *NetworkListener* class is responsible for creating sockets and sending multicast address. There are different types of messages in the protocol and the methods corresponding to them will form that particular type of message.

The CASP class is the core class for our service discovery protocol implementation. It provides access to the discovery protocol from a device, so that it can control User and Service Agents and manage and access both local and remote cache memory.

The *caspcache* class implements a cache memory on the device. This memory can be a *local cache memory*, to store services held by the device, or *remote cache memory*, to store services learnt from the network. The basic difference between these memories is found in its elements; a service stored on a remote cache can expire, so this class implements a mechanism to auto-update remote cache memory and remove its invalid service entries.

**Entities used in our implementation:**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>casp_ua</td>
<td>User Agent; Thread responsible for discovering services.</td>
</tr>
<tr>
<td>casp_spa</td>
<td>Service Provider agents; Group of threads responsible for announcing services.</td>
</tr>
<tr>
<td>casp_networklistener</td>
<td>Thread responsible for sending and receiving messages.</td>
</tr>
<tr>
<td>casp_rm</td>
<td>Core of the service discovery</td>
</tr>
<tr>
<td>caspcache</td>
<td>Local Cache to store services held by the device.</td>
</tr>
<tr>
<td>caspremotecache</td>
<td>Place to store services learnt from network</td>
</tr>
<tr>
<td>casp_advqrym</td>
<td>Class that defines different types of messages; and its corresponding processing functions.</td>
</tr>
<tr>
<td>serviceentry</td>
<td>Class that represents an Entry and its processing methods.</td>
</tr>
</tbody>
</table>

To create a reference to new CASP object the following code can be used:

```java
   casp mycasp = new casp (local-cache-size, remote-cache-size);
   mycasp.newua();
```
To add services to the cache the following code can be used:

```java
boolean added = mycasp.addlocal(message advertise_m,url,ttl);
```

To search the services the following code may be used:

```java
mycasp.caspget(servicetype);
```

5.4 Experimental Evaluation of Service Discovery in CASP

The proposed work was modelled in a computer simulation environment, by defining a framework for representing the network, mobility and service discovery scenarios and measuring proper performance metrics. The proposed model is simulated in NS-2 version 2.31.

The parameters and sample values in this simulation are shown in table 5.2. These parameters corresponds to a scenario, in which 10 nodes were randomly moving in an area (300m X 900m), with a maximum speed of 20m/s. Each node might contain two services. A service requester was making search request to the service providers and the success rate and discovery time were measured.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of Nodes</td>
<td>100no.</td>
<td>Routing Protocol</td>
<td>DSDV</td>
</tr>
<tr>
<td>Maximum node speed</td>
<td>20m/s</td>
<td>Channel</td>
<td>WirelessChannel</td>
</tr>
<tr>
<td>Maximum pause time</td>
<td>600s</td>
<td>Propagation</td>
<td>Random way point</td>
</tr>
<tr>
<td>Maximum X</td>
<td>300m</td>
<td>Phy</td>
<td>WirelessPhy</td>
</tr>
<tr>
<td>Maximum Y</td>
<td>900m</td>
<td>Mac</td>
<td>802-11</td>
</tr>
<tr>
<td>Duration of simulation</td>
<td>900ms</td>
<td>Queue</td>
<td>Drop Tail/PriQueue</td>
</tr>
<tr>
<td>Number of service per device</td>
<td>1no.</td>
<td>Link</td>
<td>LL</td>
</tr>
</tbody>
</table>
Table 5.2 Simulation Parameters for Evaluating CASP Service Discovery

| No of total clients | 50 no. | Antenna | OmniAntenna |

Performance metrics

In order to evaluate the performance of a discovery protocol properly, defining appropriate metrics is required. As a measure, the following parameters are used as performance metrics:

*Average Discovery time*: this is the time used for reactively discovering a service, which consists the time from making request, until the reply time, if the discovery is successful. The reactive discovery time is dependent on the delay caused by the network (link rate, routing delay etc.), packet loss and casting type (Unicast/ multicast). When the discovery is timed out, the entire network is examined for measuring discovery failures. All the successful replies are marked as *success* and all undiscovered entities in the network as *failure*.

*Success rate*: this is the ratio of the number of nodes that successful discovered the service providers over the total number of clients attempted. Success rate is calculated based on equation (5.2)

\[
\text{success rate} = \frac{\text{No. of Successful discovery}}{\text{Total number of attempts}} \times 100
\]  

(5.2)

This section is about the evaluations made on CASP in mobile ad hoc network and different background traffic situations. Here the *success rate* and *average discovery time* of the search are measured.

Simulation results and analysis

The *success rate* for various pause times is shown in figure.5.14. When the maximum node speed is 1 m/sec, the system gave about 21% higher than the system without providers metric. When the maximum node speed was 20 m/s, the proposed system provided 16% better
query success ratio the system without providers metric. Such an improvement is possible only because the clients’ nodes make connections to the nodes with higher battery power and low mobility.

![Graph showing success rate with and without service provider’s metrics.](image)

**Figure 5.14 Success Rate with & without Service Provider’s Metrics**

*Average discovery time* for various pause times for both speeds is shown in figure.5.15. The system with metrics resulted 26% faster than the one without using metrics at the maximum speed of 0 m/s and 33% higher than at the maximum speed of 20 m/s. This clearly shows that selecting slow moving providers ensures positive response and thus gives faster response time. It shows the advantage of using the providers’ metrics for selecting the services.
During the simulation, a number of search messages were randomly sent. The background traffic used here was CBR and the impact of 0 to 5 concurrent CBR sessions on the discovery time and success rate was investigated.

**Figure 5.15 Average query response Time with & without Service Provider’s Metrics**

**Figure 5.16 Impact of 0 To 5 Concurrent CBR Flows on the Success Rate**
From figure 5.16 it can be seen that, the success rate degrades with higher background traffic rate and increasing the number of concurrent traffic flows. In addition, discovery time increases with number of flows, but it does not considerably change with background traffic rate. At the same number of concurrent traffic flows, the only delay for search replies can be because of time that the media is occupied and packets are sent immediately after the traffic packets are sent, so the delay in sending the packets and therefore discovery time is negligible. However, when the number of flows increases, the contention for accessing media rises and causes delays in sending the packets considerably influencing the discovery time.

At the same number of concurrent traffic flows, the only delay for search replies can be because of time that the media is occupied and packets are sent immediately after the traffic packets are sent, so the delay in sending the packets and therefore discovery time is negligible. This has been shown in figure 5.17. However, when the number of flows increases, the contention for accessing media raises and causes delays in sending the packets considerably influencing the discovery time.
Figure 5.18 Effect of Mobility on Network Throughput

Figure 5.18 presents an interesting result. The Figure plots the network throughput over time for CASP with 1, 2 and 4 servers and 50 clients with a CBR sending rate of 7.5 packets per second. Nodes move with maximum speed of 20 m/sec. Before mobility starts, network throughput goes up with the number of servers. With mobility however an increase in servers leads to a decrease in network throughput. The increased number of servers in the network causes more interference between client-server pair. So, for mobile networks with no rediscovery decreasing the number of servers may actually increase network performance.

The trend outlined in figure 5.18 motivated the need to re-evaluate severs selection to offset the effects of mobility on network capacity. It is compelling that as the number of servers in the network increases; the need to re-evaluate server selection becomes more urgent. To make a better framework suitable for mobile ad hoc environment this thesis proposes re-evaluation policies which are discussed in chapter 7.
Comparison of CASP with Other Protocols

In this section the simulations to compare the performance of CASP with other service discovery protocols namely SLP, KONARK and ADDER were presented. These protocols were chosen because all of them follow the approach of application layer implementation.

The service location protocol (SLP) is an IETF standard for automatic service discovery in IP networks. The abstract architecture consists of User Agents (clients), Service Agents (servers) and Directory Agents (directories). SLP can operate in either centralised or distributed mode. In the centralised mode, servers advertise their services to Discovery Agents and clients unicast their request directly to the DAs, which respond with a list of all services that match the clients’ request. Clients and servers learn about DAs by reading from a statically configured file, waiting to receive periodic DA advertisement message or sending an unicast message to trigger DA advertisement. In distributed mode, no DAs are present and clients query servers directly by sending multicast request, a server unicast a response to the querying client. The centralised version of SLP is referred to as SLP-CENT and to the distributed version as SLP-DIST.

SLP clients can base service selection on many factors. The versions PING have selected which tries to select the closest server by uncasting ping messages to all available servers and choosing the server whose ping reply arrives first. This thesis uses UDP as communication protocol. The descriptions on ADDER and Konark have been presented in chapter 3.

Clients communicate with servers by sending 100 bytes packets at a constant bit rate (CBR). Many experiments with several client sending rates were conducted to simulate both unsaturated and saturated networks (2.5 packets per sec and 7.5 packets per sec), number of service providers 1, 2, 4 and 6 and number of clients 30, 50 and 70. For every configuration, the presented results are averaged for 5 movement scenarios.

Static Network
**Route length:** Figure 5.19 shows the optimal average path length from a client to the closest server and the average route length for each of the service discovery protocols running over DSDV. Varying the number of clients did not affect initial route lengths.

CASP picks optimal routes most of the time because of the hop count metric. SLP-DIST and SLP-CENT, ADDER, KONARK have longer average routes because they do service selection based on limited knowledge of network topology. The SLP protocols try to find the closest servers by measuring round-trip-time, where as Konark uses only the discovery messages to a fixed multicast group. ADDER uses the technique of increasing the hop limit gradually to find the services. CASP out performs 25% higher than ADDER. The SLP-CENT picks servers randomly and is obviously the worst case. It represents the case where no topology information is available or recorded.

![Figure 5.19 Average Route Length with Varying Number of Service providers In Static Network](image)

**Figure 5.19 Average Route Length with Varying Number of Service providers In Static Network**

**Throughput:** Overall network throughput for 100 seconds of communication of the service discovery protocols running over DSDV is shown in figure 5.20. Clients send data at a fixed rate of 7.5 packets/per second. At this sending rate which comes close to saturate the network there is a strong correlation between the average path length and network throughput.
At the lower sending rate, the performance of SLP-DIST, CASP, ADDER and KONARK are comparable with delivery rates reaching 90%. For lower sending rate with no node mobility, the network is not saturating enough and packets experiencing low contention levels successfully reach even the farthest servers.

**Effects of Motion**

This section evaluates the effects of node mobility on the service discovery protocols. In all mobility experiments, clients’ starts sending data at 100 seconds, motion starts at 200 seconds and the experiments end at 800 seconds. SLP-CENT has not been included for this evaluation because the initial server selection between the SLP-CENT and SLP-DIST are very similar. Figure 5.21 plots the network throughput over time for all protocols for 6 servers and 50 clients with CBR sending rate of 7.5 packets/second. Nodes move with maximum speed of 20 m/sec. This Figure shows without rediscovery, all the protocols converge to the same throughput. This is an expected result as the initial server choice becomes irrelevant after a period of movement.
In summary service selection has a crucial effect on overall performance. Near optimal service selection based on service providers location and metrics localizing communication and reduces interferences. Moreover with good service selection, as the number of services increases, overall network throughput goes up. Poor service selection on the other hand results in interference that severely limits network throughput. It is interesting to note that with poor service selection as the number of service providers grows interference increases further degrading network capacity.

Discussion
CASP is a context based service provisioning system specially designed for ad-hoc networks. The difference between the protocol of CASP and the proactive and reactive protocols are summarised in Table 5.3.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Service Exchange</th>
<th>Service Query</th>
<th>Service list update and maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASP</td>
<td>Broadcast with d hops – d based on the service type</td>
<td>Queries are sent only to the providers who are having the services</td>
<td>Update upon reception of update packet and maintain with hello messages</td>
</tr>
<tr>
<td>Proactive</td>
<td>Broadcast to part of or the entire network</td>
<td>Query only the node who has required information</td>
<td>Updates in part of or the entire network</td>
</tr>
<tr>
<td>Reactive</td>
<td>No</td>
<td>Query over part of or the entire network</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.3 Comparison of CASP With Reactive and Proactive Protocols

From table 5.3 three features of the framework designed has been compared with the reactive and proactive protocols. While requesting, the request limit will be decided based on the application need. So, it may be one hop or it may n-hop depending upon the need. Since the service exchange is limited to d-hop rather than the entire network, the nodes may be ensured to have only the local information which is immediately usable.
Summary

Services are advertised to nodes in the network. Service discovery protocol proposed in this thesis used peer-peer store and forward method. Nodes store the advertisement in the local cache as and when they receive. If the cache is full the service which is not in the requirement list will be replaced. Service selection has been done based on provider’s metrics such as speed, remaining battery power. The proposed work has been compared with the other application layer protocols to find the performance gain of the proposed protocol. In the proposed work the service selected with metrics shows 21% increase in the success rate compared to one without metrics. Similarly the proposed work shows that the system with metrics discovered 26% faster than the one without metrics. The proposed work showed marginal improvement than the other work in static networks and better performance gain will be achieved when re-evaluation on service provider is used. This topic will be discussed in chapter 7.