Towards Distributed Service Discovery in Pervasive Computing Environments

Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, Tim Finin

Department of Computer Science and Electrical Engineering,

University of Maryland Baltimore County,

Baltimore, MD 21250

Email: {dchakr1, joshi, yeyesha, finin}@cs.umbc.edu

Abstract

The paper proposes a novel distributed service discovery protocol for pervasive environments. The protocol is based on the concepts of peer-to-peer caching of service advertisements and group-based intelligent forwarding of service requests. It does not require a service to be registered with a registry or lookup server. Services are described using the Web Ontology Language (OWL). We exploit the semantic class/subClass hierarchy of OWL to describe service groups and use this semantic information to selectively forward service requests. OWL-based service description also enables increased flexibility in service matching. We present simulation results that show that our protocol achieves increased efficiency in discovering services (compared to traditional broadcast-based mechanisms) by efficiently utilizing bandwidth via controlled forwarding of service requests.

keywords: Service Discovery Architecture, Pervasive Computing, MANET, OWL, Semantic Description, Peer-to-Peer, Advertisements

This work was supported in part by NSF Awards IIS 9875433, IIS 0209001, CCR 0070802, ACI 0203958, and the DARPA DAML program under contract F30602-97-1-0215

I. Introduction and Motivation

Service discovery is a well-recognized challenge in distributed environments [40], [9], [14], [24], [27], [31]. With the decreasing cost and form factor of computing devices, the increase in the information being kept on these devices, and the increasing prevalence of short range ad-hoc wireless networks, service discovery will play an important role in *Pervasive Computing environments*. *Pervasive Computing* environments are comprised of handheld, wearable and embedded computers, besides regular desktop clients and servers. These are connected by some combination of wireless ad-hoc networks and wireless infrastructure based networks such as WLANs. In such environments, the cohort of computing elements participating in any distributed system dynamically changes with time. In other words, a user (her computing device(s) to be precise), spontaneously networks with different devices as she and other users change locations over a period of time. This is not to say that all elements in this distributed scenario must be mobile – only that no particular set of devices/computers is available to form the stable core of a distributed system at all times. For instance, in environments such as shopping malls, conference venues or smart-offices, some devices (e.g. Desktops/Laptops, IP phones, Point of Sale terminals, Projectors, Coffee machines) are static while other devices (Cell phones, Handhelds etc) are mobile. In the extreme case, *Pervasive Computing environments* include MANETs (Mobile Ad hoc Networks) where all nodes are mobile and dynamically change their locations. Examples of such environments can be found in the mobile devices used by emergency response services, by soldiers in battlefields, by people walking on streets etc.

We envisage that in the near future, static, mobile, and embedded devices will provide customized information, services and computation platforms to peers in their vicinity. The primary goal of applications for pervasive computing environments is to perform the task given by the user by exploiting the resources or services that are present in the neighborhood. Some requests need a

single service, which is directly available in the vicinity, whereas some other requests need multiple services or information sources to be integrated to obtain the desired result. In either case, we need a flexible service discovery infrastructure that is tailored towards pervasive environments[10].

Of course, there are issues related to security and privacy in such environments. Other colleagues in our group are building distributed trust and belief based systems for security and privacy in pervasive environments [2], [28], [44]. There is also a question of payments for services offered in this environment. This is outside the scope of our present work, but is being actively researched in the m-commerce and economics domains.

There has been considerable academic and industrial research effort in service discovery in the context of wired as well as partly wired/wireless networked services. Two important aspects of service discovery are the *discovery architecture* and the *service matching mechanism*. Protocols like Jini [1], Salutation and Salutation-lite [40], UPnP [25], UDDI [45], Service Location Protocol [22] have been developed to facilitate applications to discover remote services residing on stable networked machines in the wired network. Some of these protocols (e.g. UPnP) can also be used by mobile devices to discover networked services using wireless networking technologies like 802.11 (a, b or g). The general architecture of these protocols is as follows: a service advertises and registers itself to a service register that keeps track of networked services. Services can de-register at any point of time. Most of the communication happens over IP-type networks, and the discovery protocol relies on multicasts and broadcasts for important functions such as the discovery of the registry. In summary, these architectures are primarily centralized/semi-centralized, registration-oriented and have an implicit assumption that the underlying network is stable and is capable of providing reliable communication. Clearly, service discovery in pervasive computing environments requires a decentralized design approach where a node should not depend on some other node(s) to advertise/register services. Each service should be autonomous and be able to

advertise its presence. Moreover, the discovery should also adapt itself to reflect the changes in the vicinity. A discovery protocol should be able to utilize the underlying network efficiently.

Existing *service matching* techniques in the above-mentioned protocols use simple matching schemas. They use interface descriptions(e.g. Jini) or attributes [1], [40] or even unique-identifiers (Bluetooth SDP[5]). Service matching is done at a syntactic level. However, syntactic level matching and discovery is inefficient for pervasive environments due to the autonomy of service providers and the resulting heterogeneity of their implementations and interfaces. For example, we can have the same service implement different interfaces which could result in the failure of a syntactic match if the service query does not match with any interface. To alleviate this problem, there has been considerable work to develop languages [29], [6], [20] to express service requirements and facilitate flexible semantic-level service discovery [11], [49], [18].

Service *discovery architectures* [23], [3], [2] developed specifically for pervasive environments are either request broadcast based or advertisement-based. In a broadcast-based [1] solution, a service discovery request is broadcast through out the network. If a node contains the service, it responds with a service reply. The protocol, under ideal conditions of a fully-connected network without message losses, offers high reliability in discovering a service. However, it suffers from the following disadvantages. First, global broadcast scales poorly with increasing network diameter and network size. Second, it utilizes resources and computation power on all nodes of the network including nodes that do not even have the service or nodes that may not even fall in the route to the desired service. This extra processing is essentially redundant. Third, it utilizes significant network bandwidth (since the request traverses to all nodes through all paths possible) and hence creates a large load on the network.

The other solution is for the services to advertise themselves to all the nodes. Each node in-

[1]Broadcast-based protocol is also referred to as Request-broadcast based protocol in some parts of the paper

terested in discovering services cache the advertisements. The advertisements are matched with service requests and a result is returned. In this solution, the cache size increases with the number of services. Many of the nodes have limited memory and are unable to store all the advertisements. Soon the cache gets filled up. This is also inefficient in terms of bandwidth usage, since the whole network has to be periodically flooded with advertisements. There are solutions that offers both advertisements and broadcast of requests, but nevertheless do not address the problems of network load, network-wide reachability and scalability.

Existing solutions have mostly considered the *service matching* and the *discovery architecture* as two decoupled fields. This paper introduces a novel approach (dubbed *Group-based Service Discovery or GSD*) that combines the two by utilizing semantic service descriptions used in service matching to develop an efficient, distributed, scalable and adaptive service discovery architecture for pervasive computing environments. Our architecture is based on the concept of peer-to-peer caching of service descriptions, bounded advertising of services in the vicinity and efficient selective forwarding of service discovery requests using functional group information being propagated with service advertisements. Functional grouping of services enables our architecture to encompass a broad range of discovery techniques ranging from simple broadcast to directed unicast, thus making it highly adaptable to the requirements of the network. Our solution exploits the semantic capabilities offered by the Web Ontology Language (OWL)[20] to effectively describe services/resources present on nodes in the ad hoc environment. Furthermore, the services present on the nodes are classified into several groups based on the class-subclass hierarchy present in OWL. A service thus belongs to a hierarchy of groups starting from the parent group called "Service". This group information is used to selectively forward a service request to other devices where there are greater chances of the service being discovered. Semantic grouping of services is not uncommon in the service matching research and has been used to enable functionally similar

or "near" matches [11], [33]. We use the information to enable semantic matching and build a highly integrated yet distributed and an efficient discovery infrastructure.

We have implemented GSD and extensively compared its worst case and average case performance with traditional broadcast-based solution for service discovery. We provide results comparing GSD and broadcast-based service discovery with respect to average response time, average response hops, discovery efficiency, average network load and several other parameters. Our results show that GSD scales very well with respect to increasing network and increasing request load on the system. Our experiments also show that discovery efficiency of GSD is almost as good as discovery efficiency of broadcast-based solutions and in fact performs better than broadcast-based solutions with respect to other parameters like response time and network load.

We will use the term MANET (Mobile Ad hoc Network) and *Pervasive Computing Environment* interchangeably in the rest of the paper. MANET represents the extreme of the pervasive computing spectrum. Our system is designed to handle this extreme case and our simulations are done on a MANET. The remaining part of the paper is organized as follows: In section II we provide a brief description of the ontology and the functional grouping of services. Section III describes our protocol in detail. Section IV describes the various salient features of our protocol. Section V presents our experimental results. We survey other related works in section VI and conclude in section VII.

II. Group-based Semantic Service Description

We have chosen OWL to define an ontology to describe services/resources in a MANET. There are couple of reasons for choosing an ontology-based approach to describe services. (1) The semantics of OWL can be used to describe services in different nodes and also to enable semantic matching support with those service descriptions. Any resource or service is described in terms of

classes and *properties*. In addition, OWL provides rules for describing further constraints and relationships among resources including cardinality, domain and range restrictions as well as union, disjunction, inverse and transitivity. These axioms can be easily exploited to create an ontology describing services and service groups. (2) OWL, which is based on eXtensible Markup Language (XML) and Resource Description Framework [29] is also being used as a standard to describe information/service on the wired infrastructure and the Web. This makes our service description interoperable with other semantic web infrastructures.

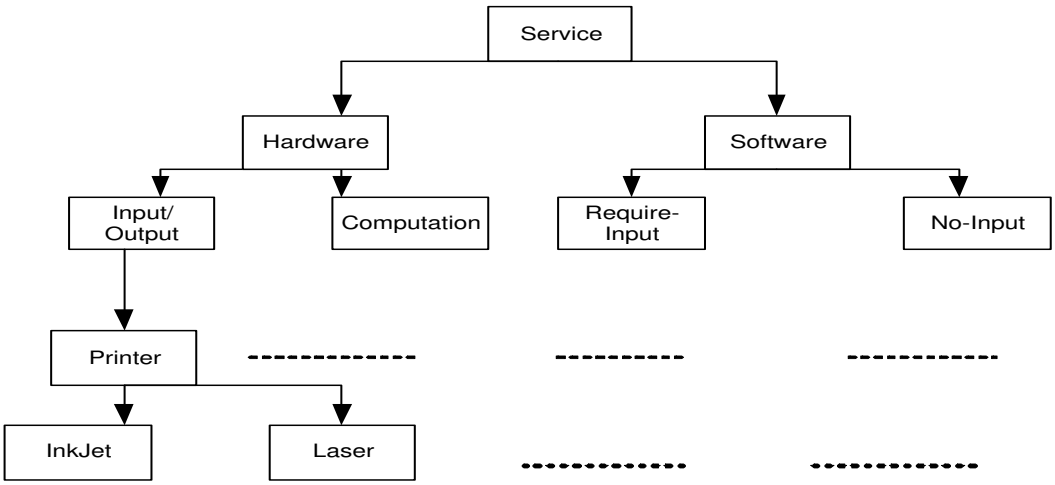

Fig. 1. Hierarchical Grouping of Services

We have leveraged our prior work in development of the DReggie Ontology [11] that contains a comprehensive ontology for describing services in terms of its capabilities, inputs, outputs, platform constraints, and device capabilities of the device on which it is residing etc. Using the class/subClassOf axiom of OWL, we have incorporated a preliminary grouping of different possible services in a MANET primarily based on service functionality. A significant advantage of our discovery architecture is that the ontology is extensible and one can modify it without altering the discovery mechanism. The discovery mechanism would take into account the modification. Due to space restrictions, we are unable to provide the ontology. However, it is available at `http://daml.umbc.edu/ontologies/dreggie-ont.owl`. The generic class *Ser-*

vice is functionally classified into two main sub-groups: Hardware and Software Service. Each sub-group is further classified in this manner till we reach a very specific service. For example, a *color printer* service may be classified under *Service/ Hardware/Input-output-type-Service/ Printer-Service*. Figure 1 shows the functional hierarchy.

III. Service Discovery Protocol

Our protocol (GSD) is based on the concepts of (1) Bounded advertising of services in the vicinity (2) Peer-to-Peer dynamic caching of service advertisements (3) Service group-based selective forwarding of discovery requests. Our protocol also has multiple user-controlled parameters that determine the extent of bounds for advertising, service caching and discovery request propagation. In this section we describe these key aspects of our protocol in detail.

A. Service Advertisements and Peer-to-Peer Caching

Each Service Provider (SP) periodically advertises a list of its services to all the nodes in its radio range. An advertisement message consists of the following fields:

<Packet-type, Source-Address, Service-Description, Service-Groups, Other-Groups, Hop-Count, Lifetime, ADV_DIAMETER>

A monotonically increasing identifier called *broadcast-id* along with the *source-address* uniquely identifies a broadcast and detects duplicate advertisements. Please note that this identifier is different from *source sequence numbers* maintained by nodes in traditional ad-hoc routing literature. Sequence numbers refer to a single message identifier whereas *broadcast-id* refers to a broadcast event that may generate multiple messages. The *Service-description* and *Service-groups* contain information about the local service(s) and their corresponding service groups.

Additionally, each node receiving the advertisement can forward it to all other nodes in its radio range. The field ADV_DIAMETER determines the number of hops each advertisement travels. Each node increments the Hop-Count when it forwards an advertisement that is in turn used to compute whether the advertisement can be forwarded any further. Figure 2 shows the pseudo code for sending advertisements.

```
Function SendAdvertisement(..) :-

1.  After each ADV_TIME_INTERVAL period do {
2.    Initialize Adv_Message;
3.    Adv_Message[Service-Description]=GetLocal_ServiceInfo(Service_Cache);
4.    Adv_Message[Service-Groups]=GetLocal_ServiceGroupInfo(Service_Cache);
5.    Adv_Message[Other-Groups]=GetVicinity_GroupInfo(Service_Cache);
6.    Adv_Message[Hop-Count]=0;
7.    Adv_Message[Lifetime]=ADV_LIFE_TIME;
8.    Adv_Message[Adv_Diameter]=ADV_DIAMETER;

9.    Transmit Advertisement to all nodes in the radio range;
10. }
```

Fig. 2. Pseudo Code of the Process of Advertising Services in the Vicinity

Each node on receipt of an advertisement stores it in its *Service Cache*. Each entry in the Service Cache contains the following fields:

<Source-Address, Local, Service-Description, Service-Groups, Other-Groups, Lifetime>

Apart from storing advertisements, a Service Cache also stores descriptions of local services in the node(identified by the *local* field in each cache entry). The field *Other-Groups* contain a list of the groups that the corresponding *Source-Address* (sender of the advertisement) has seen in its vicinity. We follow a least-remaining-lifetime replacement policy to replace entries when the cache is full. However, we are aware of work in predictive cache modeling [13] and profile-driven caching [35], [15] that can be used in our architecture to model the cache replacement strategy. However, since cache replacement policies are not the focus of this paper, we chose a simple uniform cache replacement strategy for all the protocols. Figure 3 displays the pseudo code of the peer-to-peer caching and advertisement forwarding process.

```
Function P2PCacheAndForwardAdvertisement(..) :-

1. if (Duplicate(Adv_Message))
2.   then discard Adv_Message;
3. else {
4.   Serv_Cache=Initialize_Entry_in_Service_Cache(..);
5.   Serv_Cache[Source-Address]=Adv_Message[Source-Address];
6.   Serv_Cache[local]=0;
7.   Serv_Cache[Service-Description]=Adv_Message[Service-Description];
8.   Serv_Cache[Service-Groups]=Adv_Message[Service-Groups];
9.   Serv_Cache[Other-Groups]=Adv_Message[Other-Groups];
10.  Serv_Cache[Lifetime]=Adv_Message[Lifetime];

11.  if (Adv_Message[Hop-Count]<Adv_Message[ADV_DIAMETER]) {
12.    Increment_HopCount (Adv_Message);
13.    Retransmit_Advertisement (Adv_Message);
14.   }
15. }
```

Fig. 3. Pseudo Code for Peer-to-Peer Caching and Forwarding of Service Advertisements

The advertisement frequency, advertisement diameter and advertisement lifetime are user-controlled parameters that enables GSD to be adapted to the necessities of the device and the environment. Thus, devices in relatively static environments may choose to have a low advertisement frequency with a high advertisement diameter whereas the reverse can be applied towards highly mobile scenarios where devices have low availability. We follow the policy of *passive pushing* of advertisements rather than *active pulling* of descriptions from nodes. Passive pushing enables a device to detect changes in the environment by the receipt of a new advertisement, thus making the detection process simple, efficient and localized to the device. Active pulling of information on the other hand has greater chances of collision of messages at the receiving node.

B. Advertising Service Groups

Apart from advertising its own services, GSD also uses the same advertisements to advertise functional group information of services a node has seen in its vicinity. The field *Other-Groups* in an advertisement contains an enumerated list of the service groups of all the non-local services seen by sender node. This information is obtained from the advertisements stored by the node in its

service cache (line 5 in Figure 2). Figure 4 shows the pseudo code for the function that computes this information.

```
Function GetVicinity_GroupInfo(Service_Cache) :-

1. Other-Groups={};
2. For each Entry S in the Service_Cache do {
3.   If (S is not local){
4.    for (each group Gi belonging to S[Service-Groups] or S[Other-Groups]) {
5.         if (Gi is not in Other-Groups) then
6.                Add Gi to Other-Groups
7.    }
8.   }
9. }
10. return Other-Groups;
```

Fig. 4. Algorithm to Determine the Service Groups Present in the Vicinity of a Device

We observe that this service group information gets propagated from one node to another and may potentially cover the whole network (if the network is partition free). Functional group information provides a good abstraction to represent services and are enough to divert a discovery request towards the appropriate region. They also provide a good measure to aggregate the service descriptions and hence save on network bandwidth.

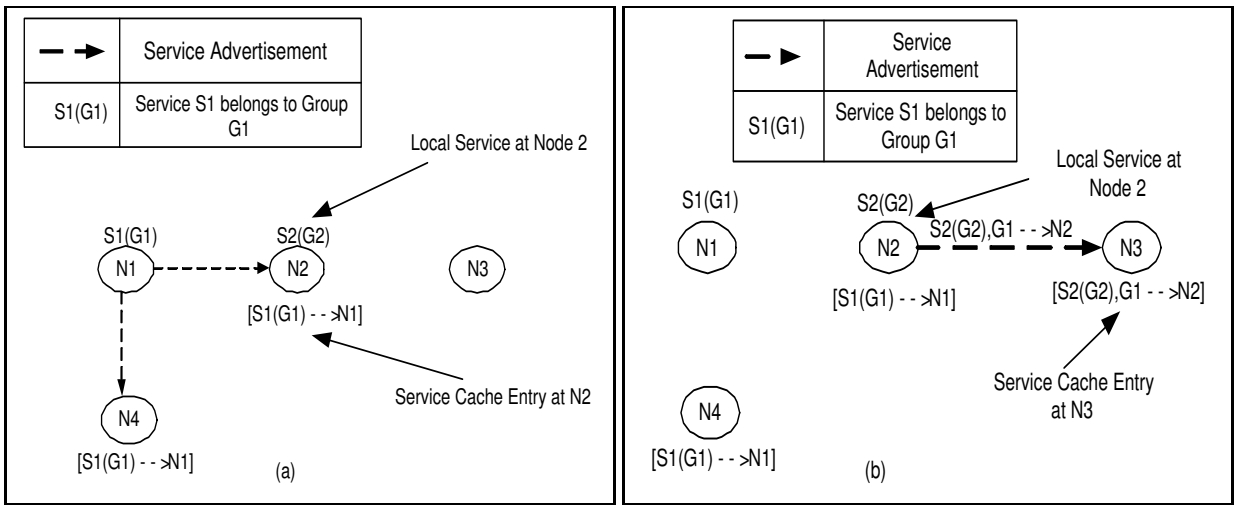

Fig. 5. Service Advertisements and propagation of service group information. Figure 5(a): Advertisements being sent by node N1. Figure 5(b): Service Group information being propagated by node N2 during its advertisement phase.

Figure 5 shows an example of propagation of service advertisements and the associated service

group information for a simple ad-hoc network. We note that with an increase in the diversity of services in a pervasive environment, the different functional groups of services would also increase. Each device has a maximum limit of the number of service groups it keeps for a certain neighboring node. Currently, the limit is set to the size of the hierarchical tree. However, for memory constrained devices, our protocol allows lower values for the maximum number of stored service-groups. Section III-C explains actions taken when a node does not have enough group information to forward a discovery request.

C. Request Routing

A service discovery request originates from a Request Source (RS) whose application layer requests the service. A request consists of an ontology based description of the service requested, and optionally includes descriptions of service groups to which the requested service belongs. The request is matched with the services present in the local cache of the RS (that might also be a SP). A service discovery request is formed on a local cache miss and contains the following fields:

<Packet-type, BroadcastId, Service-Description, Request-Groups, Source-Address, Last-Address, Hop-Count>

The field *Request-Groups* contains the service group(s) to which the requested service belongs. *Hop-Count*, a user-controlled parameter specifies the maximum propagation limit for the request. We use the information regarding *Other-Groups* present in the service cache of each node to selectively forward a discovery request in case of a local cache miss. Recall from the previous subsection that each entry in the service cache of a node contains a field *Other-Groups*. Thus, if the request belongs to one of those groups, then there is a chance that the requested service might

be available near the node that sent the advertisement. Consequently, instead of broadcasting the request, GSD selectively forwards the request to those nodes.

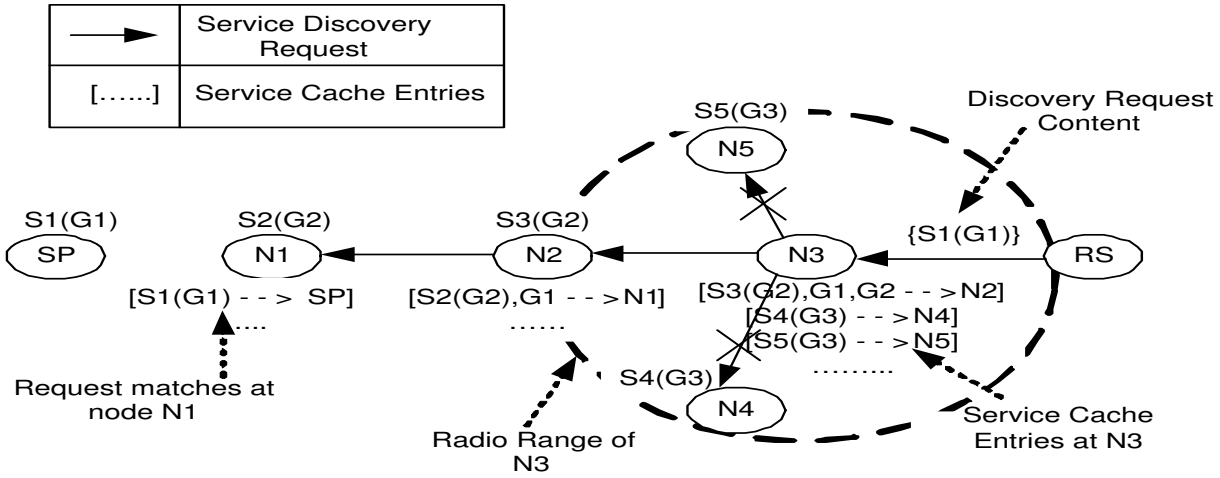

Fig. 6. Group-based Selective Forwarding of Service Discovery Request

The selective forwarding process is explained in Figure 6 for a simple ad-hoc network. It shows a sequence of nodes connected to each other with RS being the requesting source and SP being the service provider where the requested service (S1) is available. For the sake of simplicity, we only display a linear connection of nodes and do not show other nodes that might be present in the vicinity. We do not show the exchange of advertisements in the figure. Assuming that each node has advertised its own services and other remote service groups, Figure 6 shows the partial service cache entries in each node. For example, the entry

$$S2(G2), G1 -> N1$$

in node N2's cache means: (1) N2 knows that node N1 has service S2 belonging to group G2; (2) N2 knows that N1 has seen a service belonging to group G1 in its vicinity. When a request belonging to group G1 comes to N3, then instead of broadcasting it again to all nodes in its vicinity (N4, N5), N3 selectively forwards it to node N2. This is because only N2 claims to have seen

a service belonging to group G1 in its vicinity. This process continues in all other nodes until the request has reached N1 where it finds a direct match of the requested service (present in the service cache of N1).The request is by default broadcast to other nodes when the algorithm fails to determine a set of nodes to selectively forward the request to. Figure 7 shows the pseudo code of the selective forwarding process.

```
Function Selective_Forward(..) :-

1. if (Hop-Count of Discovery_Message >0) then {
2.  Request-Groups=Discovery_Message[Request-Groups];
3.   for (each entry S in Service_Cache} do {
4.     If any group Gi in S[Other-Groups] belongs to Request-Groups then {
5.       Node N=S[Source-Address];
6.       Decrease the Hop-Count of the packet by 1;
7.       Forward the Discovery_Message to N;
8.       }
9.   }

10.  if (the request was never forwarded) then {
11.     Decrease the Hop-Count of the packet by 1.
12.     Broadcast the request to the neighboring nodes.
13.   }
14. }
```

Fig. 7. Algorithm showing the Selective Forwarding Process in GSD

We observe from the above algorithm, that when a node does not have enough information to selectively forward a request, it broadcasts the request to its neighboring nodes. As a practical example, a Service Request for a Printer Service could specify its Request-Group to be <NULL>, or <Input/Output>, or <Input/Output, Hardware>, or <Input/Output, Hardware, Service>. Thus depending on the amount of Request-Group information, the request would be selectively forwarded (or broadcast) to other nodes.

We observe that the selective forwarding process might also result in *false forwards*. The request might be forwarded to a region where the service is no longer available (due to mobility of nodes) or has the right group but not the exact service and neither a “near” match. This might result in the failure to discover a service that simple broadcasting of the request would have succeeded in

discovering. In section IV we explain how our protocol can be adapted to reduce false forwards. Moreover, our experiments show that the decrease in efficiency is insignificant.

D. Reverse Routing of Service Reply

Service reply is generated from the node that matches a service discovery request. There are a couple of approaches to route the reply back to the RS. (1) One can use any standard ad-hoc routing protocol like AODV [37], TORA [34], DSDV [36] to route the reply back to the RS. (2) The path traversed by the discovery request could be retraced by the reply using a reverse routing mechanism. Standard routing protocols try discovering a new route to the destination that involve steps like route discovery or broadcasting link-state information that generate additional network load. On the other hand, using the already known route traversed by the request could easily reduce this additional load. Bhagwat et al. [4] in prior work, and our own recent studies [17] indicate that integrating routing with service discovery increases system efficiency. Hence, we use the concept of reverse routing to route the service reply back to the RS. However, reverse routing fails if the route becomes stale or some of the nodes in the previously established path move away. We detect such failures and resort to traditional routing using Ad-hoc On-Demand Distance Vector protocol (AODV) to route the reply from the point of failure to the RS. The node upstream in the path detects the failure to transmit the reply to the next hop. We illustrate the concept in Figure 8.

Each Request Packet contains a *Last-Address* field, which contains the address of the node from which a request is coming. Each node in addition to maintaining the Service Cache also maintains a Reverse-Route table. Each entry in the Reverse-Route table contains the following fields:

<*Source-Address, BroadcastId, Previous-Address*>

An entry is added to the table at the time of forwarding the discovery request. The entry is kept for REV_ROUTE_TIMEOUT time units. When a service reply corresponding to a request reaches

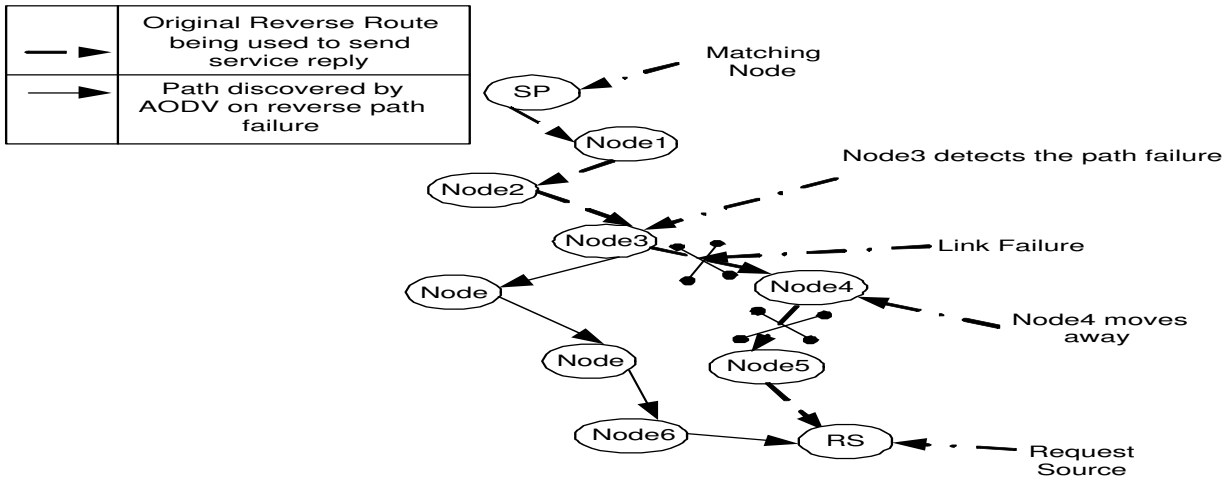

Fig. 8. Reverse Routing of Service Reply

this node, the table is consulted to determine *Previous-Address* in the path to the RS to forward the reply to. The *Source-Address* and *BroadcastId* uniquely identifies a service reply that corresponds to a particular service request.

E. Service Matching

Service matching, even though not the key aspect of this paper is important in enabling flexibility and richness in the discovery process. Apart from representing services using our functional hierarchical groups, our OWL ontology also provides constructs to describe services in terms of input/outputs, functional similarity, service capabilities, device/resource requirements etc. Additionally, each node in our architecture contains a *service matching* module that encapsulates functionalities for matching a service discovery request with a service description. We inherit various semantic features from OWL (class/subClassOf, unionOf etc) to match services with multiple request types. This allows the request to be specified in a flexible manner. For example, the same query can be represented using different requirements to match a certain service. More details of the service matching algorithm and the ontology can be found in our prior work [11].

We have augmented the service matching module to extract service group related information

from a service advertisement. This is used by the protocol to store service group information separately in the service cache of each node and facilitate the selective forwarding process.

IV. Discussion of Salient Protocol Features

This section discusses some salient features and presents some theoretical evaluations of GSD that we believe would help in better understanding the benefits of our protocol. These include enabling a broad range of discovery mechanisms, adaptability to different pervasive environments, scalability and network-wide reachability, dynamic self-starting property and network load analysis.

A. Enabling Broad Range of Discovery Mechanisms

GSD by virtue of its hierarchical grouping of services can enable a broad set of discovery mechanisms ranging from *broadcast* to *directed unicast* of the discovery requests. Service discovery requests contain information regarding the group(s) to which the service belongs. Thus, at its limit, this could represent a leaf node group in the hierarchical tree (Figure 1). If the number of selective forwards at each intermediate node is one, then this results in a directed unicast of the discovery request.

However, as described in section III, directed unicast in mobile environments may result in *false forwards*. The hierarchical grouping of services allows the discovery request to specify parent-groups (that are higher up in the functional hierarchy in Figure 1). This increases the range of nodes to which the request is selectively forwarded. This is because, higher the service group is in the tree, the more is the chance of nodes having seen a similar service. At its limit, the request is in fact broadcast if the service-group specified is the root of the hierarchical tree. Broadcast-based discovery suits some constrained pervasive environments like office space or environments where most devices are at one hop distance.

Additionally, by varying the service-group information in the request, GSD also can control the chances of the protocol in discovering a *nearly-matching* service. For example, a discovery request looking for a LaserJet color printer with a service-group value of *LaserJet printer* would not be able to discover (or reach) an *Inkjet printer* service. However, a service-group value of *printer* (that is the parent of the class *LaserJet printer* might be able to discover an *Inkjet color printer* instead since it belongs to the same parent group of services called *printer*.

B. Adaptability

GSD offers users control over several aspects of the protocol like advertisement diameter, maximum hop-count of discovery requests and advertisement frequency. This enables our protocol to easily adapt to the needs of users and pervasive environments. For example, an office environment can enforce a policy on the devices that the advertisements be broadcast only up to 1 hop. GSD does not impose any restriction on the minimum number of entries in the service cache of devices. This makes our protocol well-suited for heterogeneous devices with varying memory constraints. GSD by virtue of its registry-less structure makes a service and a device autonomous. This is very important in pervasive computing environments since dependence on other mobile lookup servers/registries makes the protocol prone to faults, due to failure of such registries/lookup servers. Services announce themselves when they come to a new environment. Services are expunged from the service caches passively if the advertisement has not been renewed for a certain time. The registry-less nature of our architecture makes it highly adaptable to changes in the vicinity due to mobility as well as device unavailability.

C. Scalability and Network-wide Reachability

Request-broadcast based protocols can theoretically cover the whole network. Hence, under ideal conditions of non-partitioned network and no message loss, request-broadcast based proto-

cols can guarantee the discovery of a service (if present). However, this protocol trades off network load to increase its discovery space. The network load due to discovery requests increases significantly with increase in the network size. GSD on the other hand, can theoretically discover any service in the network with bounded broadcasts.

Consider the network (G) in Figure 9. Let RS= Request Source that is looking for a service S, SP= an arbitrary service provider having the service S. Let us also assume that it is the only instance of S present in the network.

• **Request-broadcast protocol:**. Let D= broadcast diameter. Hence, this protocol can only cover the nodes within D hops of RS (marked by the circle with RS at its center in Figure 9). Let N= set of nodes that this protocol can cover. Clearly if SP does not belong to N, then this protocol would fail to discover S.

• **GSD protocol:** Let P= an arbitrary node lying on the edge of the network formed by the broadcast diameter D from RS. Then, assuming that the network does not have any partition, there will be at least one path leading from P to SP. This further means, that due to service advertisements, the group information of the service S will eventually reach the node P through the path. Thus, in GSD, if the discovery request reaches P, it will be selectively forwarded towards SP and would eventually be able to discover the service. Thus, GSD would essentially cover the whole network under identical conditions.

This makes our protocol highly scalable with respect to large-scale ad-hoc networks and high request load. It might appear that advertising increases the total load of our system. Our experiments show that even with bounded advertising, our protocol scales much better than broadcast-based service discovery. In fact, GSD performs much better in terms of network load for large networks.

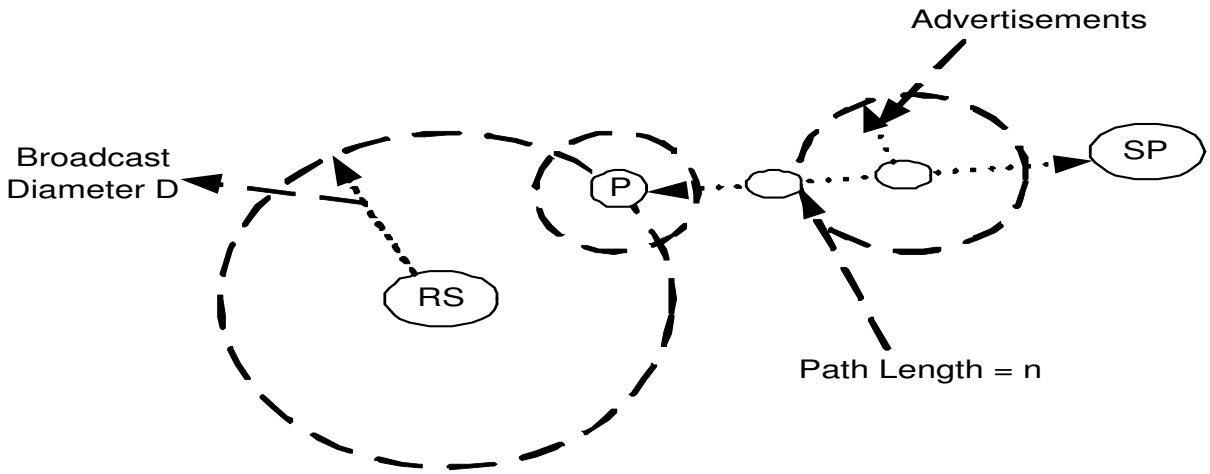

Fig. 9. Network-wide Reachability study of GSD

D. Dynamic Self-Starting Property

GSD has a dynamic self-starting property and is not dependent on any bootstrap mechanism or fixed hosts for startup. Neither is it dependent on the topology, nor the mobility of the nodes for its stability. Each node maintains a soft state of the services present in its vicinity and hence on failure, does not need to do any fault-recovery during start-up. It passively collects the information by listening to advertisements.

E. Network Load Analysis

It might appear that GSD with bounded advertisements and selective forwarding of requests may impose greater network load (in terms of number of messages) than simple global-broadcast based protocol. A global broadcast-based protocol does not have any advertisements. However, it broadcasts the requests to all nodes in the network. In this section, we layout simple equations that approximatex the network load for each of these protocols for a bounded network.

Let N=number of nodes in the network G. Let us consider that all nodes send out advertisements in GSD.

Let b= total number of nodes that generate service discovery requests

Let *T*= total time of observation.

• **Broadcast-based Protocol:** Let R_f= Request Frequency (number of requests/second). All requests are broadcast to the whole network. Let *m*= total number of messages generated in the system due to a single service request being broadcast in the network. Thus, in time *T*, the total network load generated by Broadcast is

$$M_{BCast} = R_f * m * T * b \tag{1}$$

• **GSD Protocol:** Let A_f= Average Advertisement Frequency (number of advertisements/second) across all the nodes *N* in *G*. Let *n*= total number of messages generated in a single bounded advertisement from a single node in *G*. Thus total number of messages generated by advertisements in time *T* by all nodes in *G* is $M_{Adv} = n * N * A_f * T$.

Let *p*=average number of messages generated in the system due to a single discovery request in GSD. Observe that $p \leq m$. This is because at its worst case, GSD discovery request would be broadcast through out the network. Total number of messages generated in the network due to requests in time *T* is $M_{Req} = p * R_f * T * b$.

We observe that total number of messages generated in GSD is a sum total of the request messages and the advertisement messages. Thus, M_{GSD}= total network load in *G* due to GSD is given by

$$M_{GSD} = n * N * A_f * T \quad + \quad p * R_f * T * b \tag{2}$$

We also note that for GSD to have lesser network load than Broadcast, $M_{BCast} \geq M_{GSD}$, or

$$R_f * (m - p) * b \geq n * N * A_f \tag{3}$$

V. Experimental Evaluation

We simulated the GSD protocol using the ad-hoc network simulator Glomosim [50]. We primarily compare various discovery mechanisms of GSD with simple broadcast-based discovery that has been predominantly used so far to discover services in ad-hoc/pervasive environments. It is worth noting again that in a broadcast-based discovery protocol (dubbed as BCast), a service request is globally broadcast to other nodes in the network until the required service has been discovered. There are no advertisements and the broadcast request dies down after all nodes have received the request once.

Clearly, the worst case performance of GSD (in terms of network load) is when the service request is broadcast to other nodes. This happens when enough service group information to do selective forwarding is unavailable. We call this protocol GSD-B. We also compare average case performance of GSD when GSD performs selective forwarding of a request. We call this GSD-S. We also compare performance of the protocols with varying advertisement diameter. We do not compare GSD with global advertisement based protocol, since it generates 'n' times the load generated by request broadcast-based protocol (assuming the request rate is same as the advertisement rate) and hence is a very inefficient solution for large scale networks. We observe that performance of GSD will deteriorate as the average advertisement diameter is increased. Our experiments show that an advertisement diameter of 1 provides best results.

We assume a pessimistic evaluation strategy and compare GSD in environments less favorable to it. A pessimistic evaluation strategy helps us better justify the effectiveness of GSD in more conducive environments. We impose the following restrictions on the simulation environment:

- *Request Source Restriction:* The number of request sources sending discovery requests is restricted to 1. This makes b=1 in equation 3. This reduces the additive effect formed due to multiple request sources and makes it more difficult for the equation to be true, thus favoring BCast.

• R_f/A_f *Ratio:* In equation 3, since the values of *m*, *p*, *n* are not known beforehand, we observe that a low value of R_f and a high value of A_f would make BCast more favorable as far as network load is concerned whereas the vice versa would make GSD more favorable. Hence, in our experiments, we have varied the ratio of R_f/A_f from 0.25 to 2.0. This will favor BCast on one end GSD on the other.

• *Density of Matching Services:* The more the number of SPs, the greater is the chance of either protocols discovering the service. Hence, in our experiments, only 10% of the SPs contain the service desired by the discovery request. The initial placement of the matching services were at the edge of the network.

A. Experimental Model and Evaluation Metrics

Our experimental model consists of mobile service providers (SP) containing one or more services connected to each other using an ad-hoc network. The mobility of the nodes was assumed to follow random-waypoint [26] pattern. We used an application layer packet generation function to generate service requests at regular time intervals. For the purposes of the simulation, we used representative services S0 to S99 to represent actual services and groups G1 to G10 to represent service groups with G10 being equivalent to the parent service group called "Service" at the root of our hierarchical tree.

All our experiments were carried out with a fixed node density so as to appropriately simulate the effect of increased network size. The results are an average of experiments run for 3 different randomization patterns for a total time of 75 minutes with the value of R_f ranging from 1 request/minute to 8 requests/minute. Thus, the plots are averages over a minimum of 225 data points to a maximum of 1800 data points. Figure 10 represents the various experimental parameters used and varied in our simulations.

We evaluated the protocols with respect to several metrics like average response time, average

Duration	4500 seconds
Network Area (x,y)	(145 X 145m) to (200 X 200m)
No. of Nodes	50,100, 200
Network Diameter	10, 14, 19
Tx Range (Transmission Range)	30m
Tx Throughput	20kbps
Advertisement Interval	15 seconds
Advertisement Timeout	40 seconds
broadcast jitter	10 milliseconds
Mobility	Random way-point with 2 m/s speed and 5 s stoppage time
Initial topology	uniform topology with nodes equally spaced out in (x,y)
MAX_RETRIES to discover a service	4
Advertisement Diameter	1,2
R_f/A_f	0.25 to 2.0

Fig. 10. Experimental model parameters

response hops, discovery efficiency, average network load, average message processing per node and other metrics that provide statistics regarding the usage of service groups in GSD. We present the results in the next subsection.

B. Simulation Results

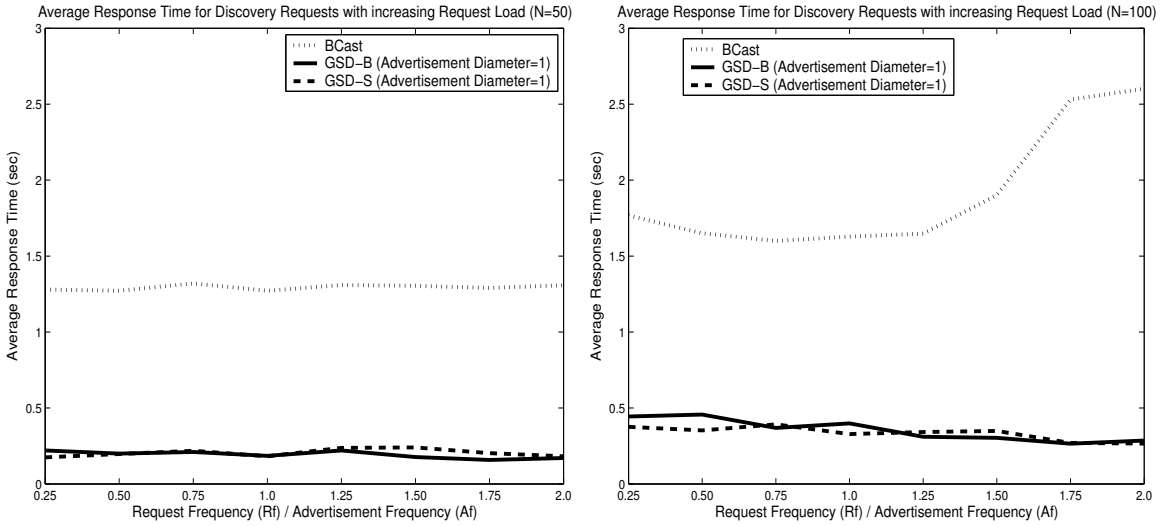

Fig. 11. Average Response Time statistics for the various protocols

Average response time for discovery requests is the time from the instant a request is sent out to

the instant a service reply is obtained. We observe in Figure 11 that the average response time of BCast is at least 2 times higher than the average response times observed in GSD-S and GSD-B. We also observe in Figure 12 that the average Response Hops or average number of hops traveled by the response is greater for BCast. Moreover, average response hops in GSD-S seems to be marginally lower than GSD-B. This shows that our protocol performs better than BCast in terms of response time and average response hops. We believe that the increase in response time is mostly due to the average response hops being about 2 times greater in BCast. The average response hops decrease in GSD because each request could travel only up to an intermediate node where a matching service description is available. The discovery request does not need to reach the actual service provider (as explained in section III-C).

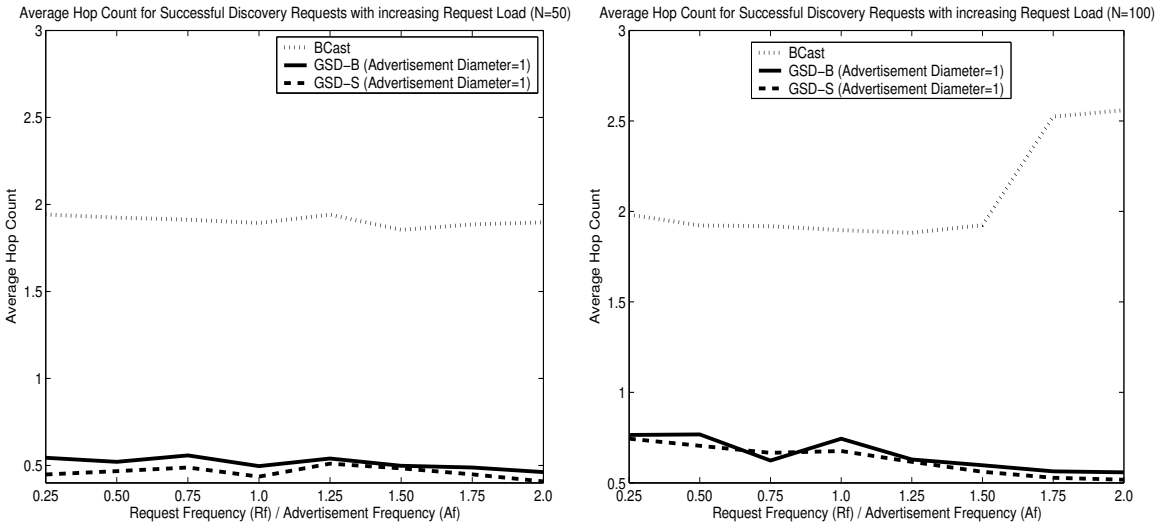

Fig. 12. Average Response Hops observed for the various protocols

Figure 13 shows the amount of network load generated by the various protocols. Average network load is defined as the average number of messages (advertisements and discovery requests) processed per node. We observe that network load of GSD-S and GSD-B increases very slowly with increasing request load. We also observe that BCast performs better for a low value of R_f/A_f. This is intuitive since according to equation 3, a low value of R_f/A_f favors BCast. However, for

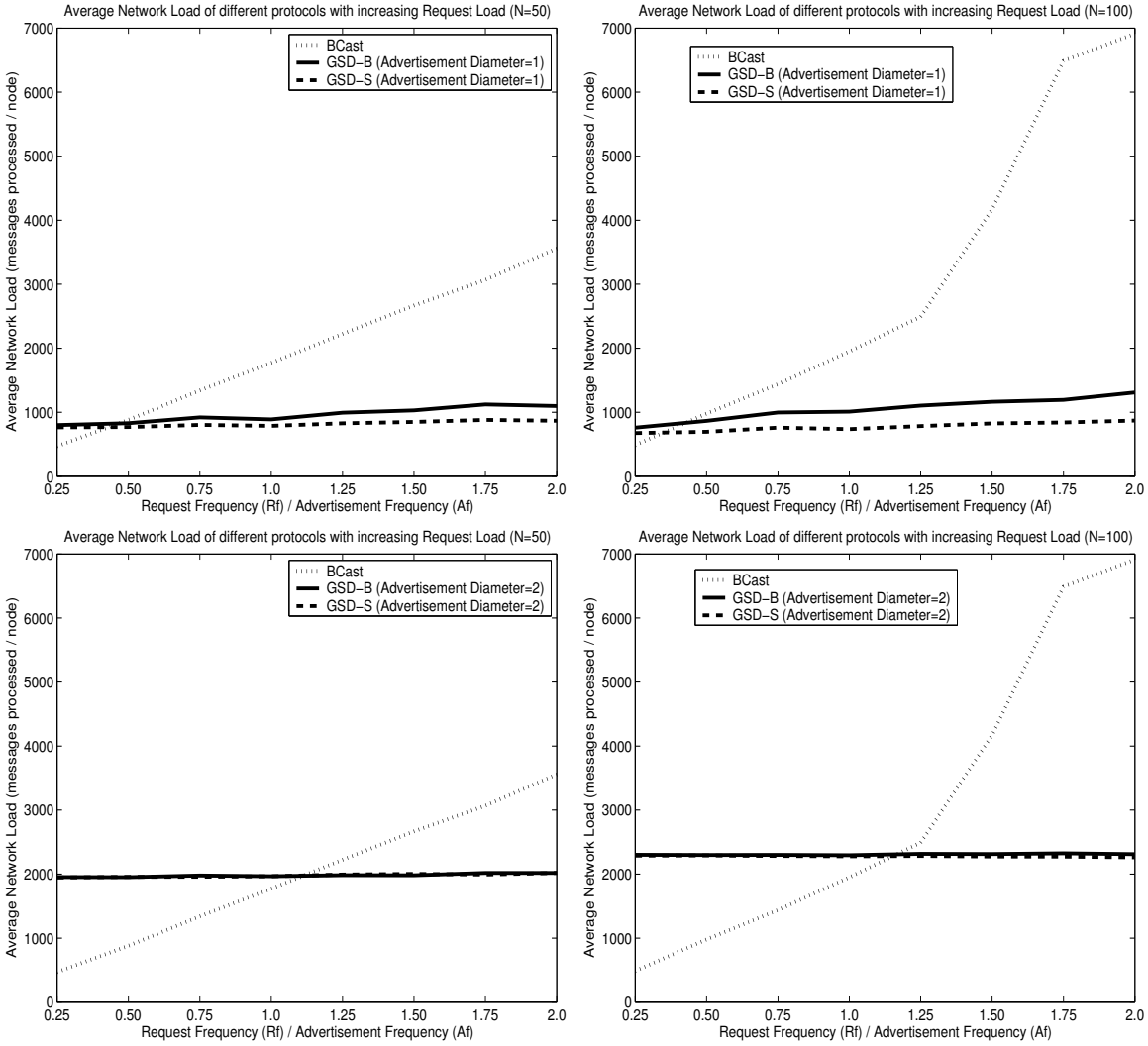

Fig. 13. Average Network Load comparison of the various protocols

values of $R_f/A_f \geq 0.50$ and advertisement diameter of 1, GSD starts performing better. We also notice similar performance improvements of GSD for advertisement diameter of 2. This shows that our protocols are very scalable with respect to increasing request load as well as network size. Understandably, GSD (both GSD-S and GSD-B) generates greater network load with increasing advertisement diameter (in terms of average number of messages processed per node). However, the increase in the network load with increasing request load is very low. Our experiments suggest that GSD-S with an advertisement diameter of 1 provides best results as far as network load and response time statistics are concerned. We also observe that the gradient of increase in the load

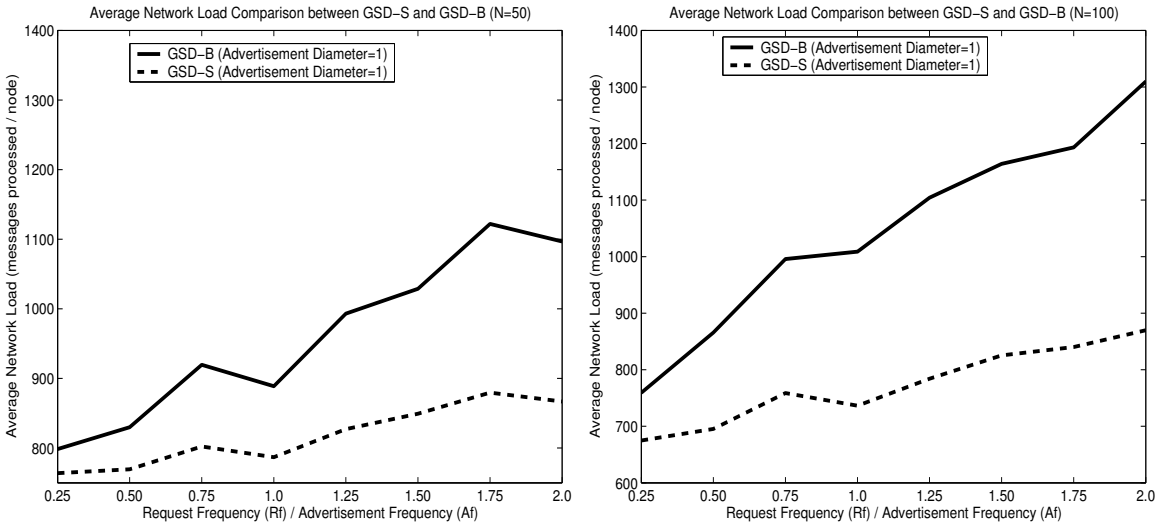

Fig. 14. Comparison of GSD-S and GSD-B in terms of Network Load

is much higher in BCast for N=100. This further proves that BCast scales poorly with increasing network size.

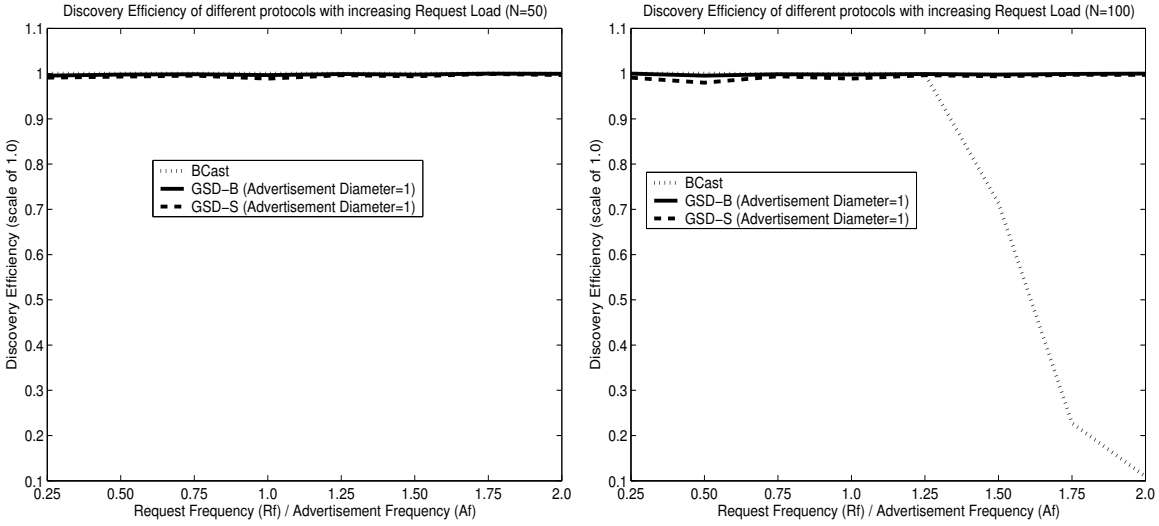

Fig. 15. Discovery Efficiency comparison of the various protocols

Discovery Efficiency is defined as the fraction of discovery requests that are successful in discovering the required service. One important tradeoff between BCast and GSD-S is that GSD-S might generate *false forwards* leading to a discovery failure. Thus, intuitively, BCast should have a greater discovery efficiency, especially in mobile environments. Figure 15 shows the various

discovery efficiencies we observed for BCast, GSD and GSD-S. The efficiencies are remarkably similar for N=50. This shows that our protocol performs almost as well as BCast but uses the network more efficiently and hence is a more scalable and efficient solution.

The efficiency of BCast drops drastically for a greater network (N=100) with high request load. We believe that this is mostly due to the huge network load generated due to broadcasting of all the requests due to which many of the service requests/responses are dropped or lost due to collisions. We could not calculate the number of messages being dropped in case of broadcasts, since Glomosim silently discards broadcast messages if there are collisions. However, Figure 16 gives us a comparison of the increase in the number of discovery requests processed per node for the various protocols that further corroborates our argument.

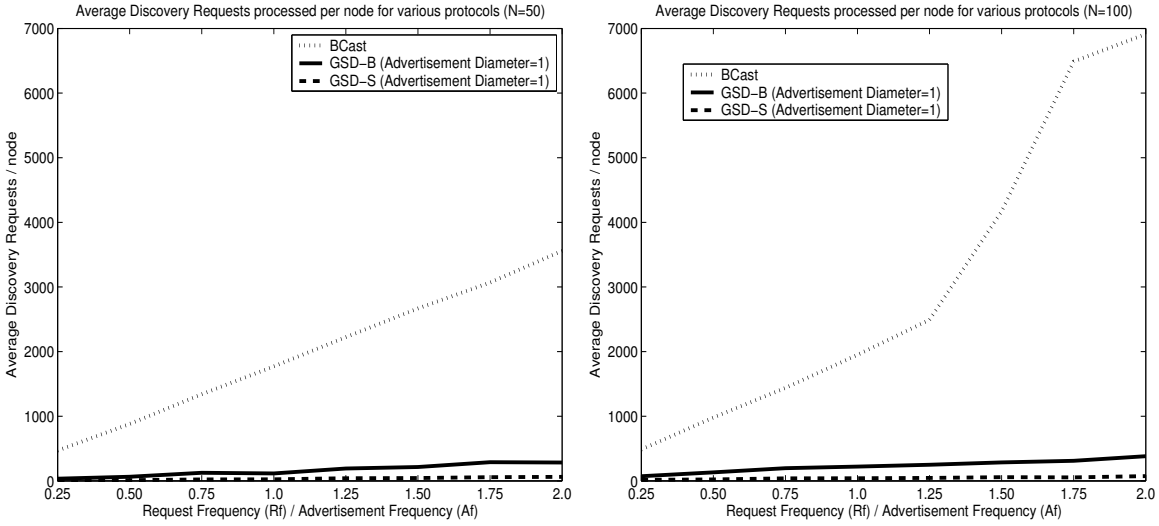

Fig. 16. Average discovery requests processed per node for the various protocols

It might seem from Figure 13 that GSD-S and GSD-B perform similarly. However, this is not true. As seen in Figure 14, selective forwarding brings about 50% reduction in total network load. The difference is not evident due to compression of the plots in Figure 13. Moreover, from Figures 11,12 and 15, we observe that GSD has this performance gain without any significant loss in terms of response time, response hops and discovery efficiency. However, we do not observe such drastic

differences for advertisement diameter of 2. We attribute this to a higher advertisement diameter that replicates the same service information across a greater number of nodes, thus reducing the number of effective selective forwards.

Figure 17 provides an estimate of the decrease in the average number of selective forward events in the nodes due to an increase in the advertisement diameter in GSD-S. We observe that that GSD-S with an advertisement diameter of 2 performs better in reducing the amount of selective forwards. This follows from our protocol, since an increase in the diameter would cause the service to be replicated in greater number of nodes, thus increasing its chances of being discovered with lesser number of selective forwards. However, we would still argue that GSD-S with advertisement diameter of 1 performs better, since it generates lesser overall network load (Figure 13).

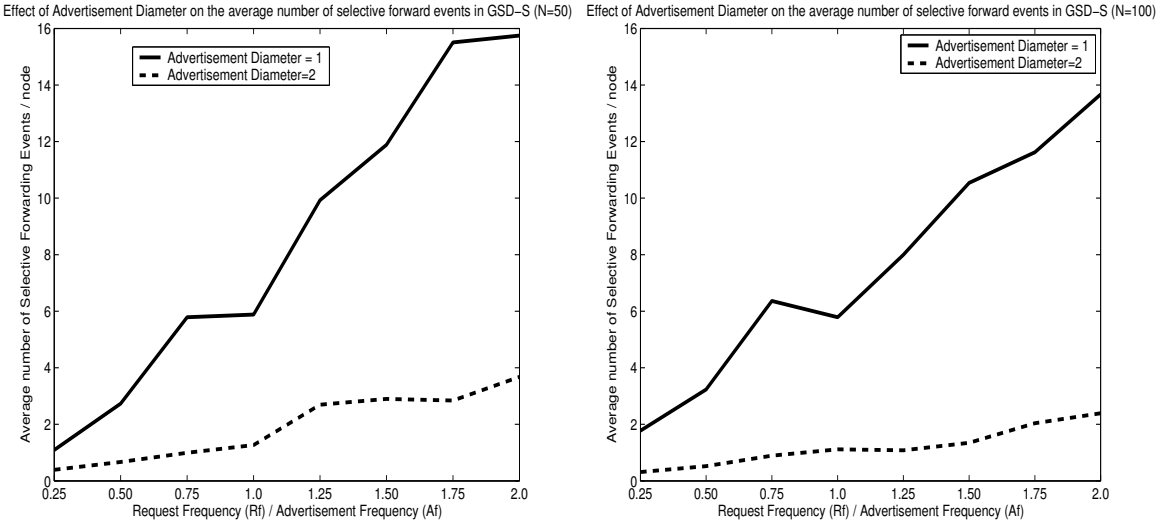

Fig. 17. Average Selective Forward events processed per node for GSD-S

We also conducted experiments with a network size of 200. The results we obtain follow similar patterns as those reported in this paper. We do not present those results due to space restrictions. However, they are available at `http://www.cs.umbc.edu/~dchakr1/papers/ieeetmcGraphs.pdf`.

VI. Related Work

Service discovery is an important and active area of research [8], [21] and has been studied widely in the context of web-services. Research in this field has forked along two branches, namely *service description and matching* and *service discovery architectures*.

Service description languages like Web Services Development Language (WSDL) [47], Web Services Flow Language (WSFL) [48] and DARPA Agent Markup Language for services (DAML-S) [19] have been developed to describe web services in a flexible manner. The Web Services Description Language (WSDL) by W3C [47] is an XML format for describing network services as a set of endpoints operating on document-oriented or procedure-oriented messages. The DAML project by DARPA and the W3C focus on standardizing OWL as the language for describing information available on any data source. The information thus may be understood and used by any class of computers, without human intervention. We have used a OWL-based ontology to describe our services and our logic behind using OWL is explained in section II.

Service Discovery Architectures like Jini [1], Salutation and Salutation-lite [40], UPnP [25], Service Location Protocol [22], have been developed over the past few years to efficiently discover wired infrastructure based services from wired as well as wireless platforms. However, most of these service discovery infrastructures have a central lookup server type architecture for service registration and discovery. Central lookup server/registry-based mechanism for doing service discovery is inappropriate in ad-hoc/pervasive environments due to the dependence of the whole infrastructure on a central point/node, which might as well be mobile and unreliable.

Research in the area of service discovery for ad-hoc networks is relatively new. Solutions [23], [43] primarily utilize the broadcast-driven nature of the underlying ad-hoc network to carry out service discovery. We have shown in section V that broadcast-driven protocols do not work well in terms of scalability and efficiency of discovery for large-scale pervasive environments. There has

been work in the field of wired networks to develop server-less peer-to-peer architectures as shown in [39], [41], [30]. However, some key limitations of such approaches with respect to pervasive environments are: (1) Traditional P2P networks derive basic boot-strap support from some *trusted hosts* that are robust and available. We cannot assume such support in an ad-hoc environment (2) Underlying protocols to discover resources are essentially broadcast-driven thus potentially generating significant network load (3) The virtual network topology of these P2P networks do not use the underlying physical Internet topology effectively, thus affecting their scalability and efficiency. Service discovery architectures in pervasive environments not only have to utilize the underlying dynamically changing topology, but also have to be independent of any boot-strap servers.

There has been work on content-centric networking and content-based message routing architectures [46], [7] that use publish-subscribe based architectures to route data based on its content. However, such architectures do not perform well in a distributed ad-hoc environment due to their centralized/semi-centralized architecture.

The Bluetooth Service Discovery protocol [42] is a peer-to-peer service discovery protocol that can be used over ad-hoc environments. However, apart from the fact that it supports very rudimentary unique-identifier based matching, the discovery is also driven by broadcast in a piconet. GSD is targeted towards generalized ad-hoc networks that are a better representation of pervasive environments. Our prior work enhances the Bluetooth service discovery protocol to include service description-based reasoning [2] using Prolog. However, it only enhances the service matching part of Bluetooth and does not address discovery architecture.

Recently, work done by Garcia-Molina et. al [16] addresses resource discovery using routing indices. They use routing indices to measure the "goodness" of neighbors in answering a query or providing a resource. However, the solution places index values on different paths in the peer-

to-peer system and hence requires huge amount of updating in the event that the paths change dynamically (as they do in pervasive environments). Our group-based service discovery protocol does not place any weight on paths; rather it adapts itself depending on the movement of the devices in the vicinity.

Advertisements in pervasive environments is coming up as a new area of research and our protocol can benefit by using intelligent schemes for adaptive advertising of services. For example, Anand et al. [38] talks about serendipitous advertising and Ratsimor et al. [32] talks about policy-based advertising that can easily perform better than periodic advertising of services. Our architecture is extensible and can easily be enhanced to accommodate these protocols.

VII. Conclusions

In this paper, we have introduced a novel architecture and protocol (GSD) for service discovery in pervasive computing environments. Service Discovery is done in a peer-to-peer mode rather than a centralized mode, and we use advertisements to disseminate service information. We use an ontology based on OWL to describe services and use the Class/SubClass hierarchy of OWL to group services based on their functionality. We use this group information to intelligently route service requests. GSD is scalable in terms of request load and network size and highly adaptable to various pervasive computing environments. We have presented exhaustive experimental results of performance of GSD in mobile environments for various kinds of request load and network sizes. Our results show that GSD scales very well with increasing request load and network size whereas standard broadcast-based solutions used so far for service discovery in ad-hoc networks do not. Moreover, our protocol provides the same standards of efficiency in discovering services when compared to Broadcast-based solutions. In fact, for large networks and high request loads, broadcast-based solutions performs worse than GSD in terms of discovery efficiency. We have implemented a restricted version [12] of GSD over Bluetooth to supplement our work in the area of service composition.

REFERENCES

[1] K. Arnold, B. Osullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification (The Jini Technology)*. Addison-Wesley, Reading, MA, June 1999.

[2] S. Avancha, A. Joshi, and T. Finin. Enhanced Service Discovery in Bluetooth. *IEEE Computer*, 35(6):96–99, June 2002.

[3] D. O. Awduche, A. Gaylord, and A. Ganz. On Resource Discovery in Distributed Systems with Mobile Hosts. In *ACM Internation Conference on Mobile Computing and Networking (MOBICOM), Newyork, USA*, November 1996.

[4] P. Bhagwat B. Raman and S. Seshan. Arguments for Cross-Layer Optimizations in Bluetooth Scatternets. In *The 2001 Symposium on Applications and the Internet (SAINT)*, January 2001.

[5] Bluetooth SIG. Specification. `http://bluetooth.com/`.

[6] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0 - W3C Recommendation. `http://www.w3.org/TR/2000/CR-rdfschema-20000327`, 2000.

[7] A. Carzaniga and A.L. Wolf. Content-based Networking: A New Communication Infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems. In conjunction with the International Conference on Computer Communications and Networks (ICCCN), Arizona, USA*, October 2001.

[8] F. Casati, D. Georgakopoulos, and M. Shan Editors. Special Issue on E-Services. *VLDB Journal*, 2001.

[9] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and Dynamic Service Composition in eFlow. In *Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, California, USA*, March 2000.

[10] D. Chakraborty and A. Joshi. GSD: A novel group-based service discovery protocol for MANETS. In *IEEE Conference on Mobile and Wireless Communications Networks, Stockholm, Sweden*, September 2002.

[11] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: A Smart Service Discovery Technique for E-Commerce Applications. In *Workshop in conjunction with 20th Symposium on Reliable Distributed Systems*, October 2001.

[12] D. Chakraborty, F. Perich, A. Joshi, T. Finin, and Y. Yesha. A reactive service composition architecture for pervasive computing environments. In *7th Personal Wireless Communications Conference (PWC 2002). Singapore*, October 2002.

[13] D. Chakraborty, A. Shenoi, A. Joshi, and Y. Yesha. Queuing Theoretic Approach for Service Discovery in Ad-hoc Networks. In *Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS), San Diego, California, USA*, January 2004.

[14] H. Chen, A. Joshi, and T. Finin. Dynamic Service Discovery for Mobile Computing: Intelligent Agents meet Jini in the Aether. *Baltzer Science Journal on Cluster Computing, Special Issue on Advances in Distributed and Mobile Systems and Communications*, 2001.

[15] M. Cherniak, M. Franklin, and S. Zdonik. Expressing User Profiles for Data Recharging. In *IEEE Personal Communications*, July 2001.

[16] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria*, July 2002.

[17] A. Joshi D. Chakraborty and Y. Yesha. An Integrated Service Discovery and Routing Protocol for Ad hoc Networks. *Ad Hoc Networks Journal, Elsevier Science, To Appear*, March 2003.

[18] T. Finin D. Khushraj and A. Joshi. Semantic Tuple Spaces: A Coordination Infrastructure in Mobile Environments. In *Second International Semantic Web Conference (ISWC), Florida, USA*, October 2003.

[19] DARPA Agent Markup Language for Services Specification Draft 0.5. `http://www.daml.org/services/daml-s/2001/05/`, May 2001.

[20] M. Dean, D. Connolly, F. V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web Ontology Language (OWL) Reference Version 1.0. http://www.w3.org/TR/2002/WD-owl-ref-20021112/, 2002.

[21] G. Weikum. Editor. Special Issue on Infrastructure for Advanced e-Services. *IEEE Data Engineering Bulletin, 24(1)*, March 2001.

[22] E. Guttman, C. Perkins, and J. Veizades. RFC 2165: Service Location Protocol, June 1997.

[23] S. Helal, N. Desai, and C. Lee. Konark-A Service Discovery and Delivery Protocol for Ad-hoc Networks. In *Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, USA*, March 2003.

[24] T. Hodes and R. Katz et. al. An Architecture for a Secure Service Discovery Service. In *Fifth International Conference of Mobile Computing and Networks, Washington, USA*, August 1999.

[25] R. John. UPnP, Jini and Salutaion - A Look at some popular Coordination Frameworks for Future Network Devices. Technical report, California Software Labs, 1999. http://www.cswl.com/whiteppr/tech/upnp.html.

[26] D.B. Johnson and D.A Maltz. The Dynamic Source Routing Protocol for Mobile Ad-hoc Networks. *Mobile Computing, Kluwer Academic Publishers*, pages 153–181, 1996.

[27] R.H. Katz, Eric. A. Brewer, and Z.M. Mao. Fault-tolerant, Scalable, Wide-Area Internet Service Composition. Technical Report. UCB/CSD-1-1129. CS Division. EECS Department. UC. Berkeley, January 2001.

[28] T. Finin L. Kagal and A. Joshi. Trust-Based Security in Pervasive Computing Environments. In *IEEE Computer*, December 2001.

[29] O. Lassila and R. Swick. Resource Description Framework. `http://www.w3.org/TR/1999/REC/rdf-syntax-19990222`, February 1999.

[30] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *16th ACM International Conference on SuperComputing, New York, USA*, June 2002.

[31] D. Mennie and B. Pagurek. An Architecture to Support Dynamic Composition of Service Components. In *5th International Workshop on Component-Oriented Programming (WCOP), Sophia Antipolis, France*, June 2000.

[32] T. Finin O. Ratsimor, A. Joshi and Y. Yesha. eNcentive: A Framework for Intelligent Marketing in Mobile Peer-To-Peer Environments. In *The 5th International Conference on Electronic Commerce (ICEC)*, October 2003.

[33] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara. The daml-s virtual machine. In *Proc. 2nd International Semantic Web Conference (ISWC)*, October 2003.

[34] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Kobe, Japan*, April 1997.

[35] F. Perich, S. Avancha, D. Chakraborty, A. Joshi, and Y. Yesha. Profile Driven Data Management for Pervasive Environments. In *13th International Conference on Database and Expert Systems Applications, (DEXA), Aix-en-Provence. France*, September 2002.

[36] C.E Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *Computer Communications Review*, pages 234–44, October 1994.

[37] C.E. Perkins and E.M Royer. Ad-hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[38] A. Ranganathan and R. H. Campbell. Advertising in a Pervasive Environment. In *2'nd ACM International Workshop on Mobile Commerce*, pages 10–14, September 2002.

[39] M. Ripeanu and I. Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In *1st International Workshop on Peer-to-Peer Systems*, March 2002.

[40] The Salutation Consortium Inc 1999. Salutation Architecture Specification (Part 1), Version 2.1 Edition. `http://www.salutation.org`.

[41] J. Sauver. Percentage of Total Internet Traffic Consisting of Kazaa/Morpheus/FastTrack. `http://darkwing.uoregon.edu/joe/kazaa.html`, 2002.

[42] Bluetooth Specification. `http://www.bluetooth.org/specifications.html`.

[43] D. Tang, C. Chang, K. Tanaka, and M. Baker. Resource Discovery in Ad hoc Networks. Technical report, Stanford University, August 1998. CSL-TR-98-769.

[44] J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal, A. Joshi, and T. Finin. A Secure Infrastructure for Service Discovery and Management in Pervasive Computing. *ACM MONET : The Journal of Special Issues on Mobility of Systems, Users, Data and Computing*, 2002.

[45] Universal Description Discovery and Integration Platform. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`, September 2000.

[46] E. Schwartz W. Adjie-Winoto and H. Balakrishnan. The Design and Implementation of an Intentional Naming System. In *In Proceedings of the Symposium on Operating Systems Principles, South Carolina, USA*, December 1999.

[47] WSDL. Web Services Description Language 1.1. `http://www.w3.org/TR/wsdl`, March 2001.

[48] WSFL. Web Services Flow Language. `http://xml.coverpages.org/wsfl.html`.

[49] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *2nd International Semantic Web Conference (ISWC), Florida, USA*, October 2003.

[50] R. Bagrodia X. Zeng and M. Gerla. GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks. *12th Workshop on Parallel and Distributed Simulations, Alberta, Canada*, 1998.