Konark – A Service Discovery and Delivery Protocol for Ad-Hoc Networks

Sumi Helal, Nitin Desai, Varun Verma and Choonhwa Lee
Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL-32611, USA
{helal,nsdesai,vverma,chl}@cise.ufl.edu

Abstract—The proliferation of mobile devices and the pervasiveness of wireless technology have provided a major impetus to replicate the network-based service discovery technologies in wireless and mobile networks. However, existing service discovery protocols and delivery mechanisms fall short of accommodating the complexities of the ad-hoc environment. They also place emphasis on device capabilities as services rather than device independent software services, making them unsuitable for m-commerce oriented scenarios. Konark is a service discovery and delivery protocol designed specifically for ad-hoc, peer-to-peer networks, and targeted towards device independent services in general and m-commerce oriented software services in particular. It has two major aspects - service discovery and service delivery. For discovery, Konark uses a completely distributed, peer-to-peer mechanism that provides each device the ability to advertise and discover services in the network. The approach towards service description is XML based. It includes a description template that allows services to be described in a human and software understandable forms. A micro-HTTP server present on each device handles service delivery, which is based on SOAP. Konark provides a framework for connecting isolated services offered by proximal pervasive devices over a wireless medium.

Keywords: Ad-hoc networks, service discovery protocols, pervasive computing, m-commerce.

I. INTRODUCTION

The last few years have seen a rapid increase in the usage of mobile devices such as laptops, cell-phones, and personal data assistants (PDAs) [7]. The accompanied maturity in wireless technologies such as 802.11 has made wireless networks almost as ubiquitous as traditional wired networks. An important consequence of such developments has been the concept of ad-hoc networks. These networks are characterized by their lack of required infrastructure and ease of formation; each participating device is mobile and the networks are formed temporarily. For example, a network formed when a group of people comes together in a conference, in an emergency relief scenario, or in an airport lounge. As the "deployment" and usage of such networks increases, new paradigms begin to emerge that incorporate and utilize these new communication capabilities in ways that were previously not thought of or were impossible to implement. Some of the possible applications and scenarios for ad-hoc networks are as follows:

- Using your wireless handheld device, the ability to not only locate a nearby restaurant serving your favorite Chinese food, but also reserve a table and order your food before walking to it.

- An application on your handheld device that packages your personal expertise of being a web-security consultant as a service, advertises this service in relevant places like technical conferences, and responds to searches for such services with more details such as your current location, your schedule, and how much you charge for a consulting session.

- The ability to share with others entertainment sources such as music and games available on your personal handheld device. Such a capability would be extremely useful while waiting for a flight in an airport lounge, or other similar situations of killing time in a public place.

In all the above-mentioned scenarios, the resources being shared or searched are packaged as services. As evident, services can be those offered by devices (e.g. printers, fax machines), or simply software services that are device independent. Even software services offer a myriad range of possibilities with opportunities related to an individual's career, entertainment or daily chores like ordering food.

The feasibility of above scenarios requires not only the formation of ad-hoc networks and packaging of resource as services, but also a discovery and delivery mechanism suited to the needs of ad-hoc networks and geared towards m-commerce oriented services. Low-level technologies necessary to form a peer-to-peer, ad-hoc network are available. The missing link is a higher-level framework and protocol, which will enable devices to discover and advertise their services over ad-hoc networks.

To address the new opportunities, issues, and requirements, we present Konark [3] [18], a middleware designed specifically for discovery and delivery of device independent services in ad-hoc networks. Konark is based on a peer-to-peer model with each participating device having the capabilities to host its local services, deliver its own services using a resident micro-HTTP server, query the network for available services offered by others, and use the services it discovered in the network.

We start this paper by discussing the issues we considered during the design phase of Konark. We then present the Konark architectural design and implementation, followed by a discussion of related work and our on-going, second-phase work.

II. DESIGN ISSUES

The computing environment defined by small handheld mobile devices with wireless connectivity will be vastly different from the traditional infrastructure-based environment comprising of PCs with wired connections [1]. While designing a service discovery protocol for ad-hoc networks, it becomes imperative to keep in consideration the challenges posed by these differences. A whole another set of issues is also raised by the type and range of services targeted.

One of the primary differences is the formation of the network itself. In the traditional wired networks, network formation can be considered to be systematic, with each participating node being assigned an identity by some administrator, or by another device in the network. Also, events such as a device joining or leaving a network are not very frequent and this provides a semblance of uniformity to the network. In ad-hoc networks, device participation is dynamic. It is not possible to assume that a controlling entity, whether an administrator or another device in the network, will be present to assign addresses to the nodes. To handle this issue, Konark assumes an IP level connectivity in the ad-hoc networks. It is a valid assumption considering most of the modern devices run operating systems that provide automatic configuration techniques [2] [6]. Such an assumption also makes Konark independent of lower network layers, which could either be 802.11, IrDA, Bluetooth or any other protocol on which IP can be implemented.

An important design factor in a service discovery protocol is the storage of services and their metadata, that is, information about available services. One possible approach is to have a centralized repository that keeps track of all available services in the network. Any network member offering a service would register its service with this repository, and all devices seeking any service would query it for available services. This approach is best suited for traditional wired networks where it is possible to assume that some device will always remain in the network. For highly dynamic networks, such an assumption is fallible. It also might not be possible to have algorithms such as leader election to choose a registry server each time some node moves out of the network. Konark uses a completely distributed peer-to-peer approach to solve this problem. It specifies that each device will have a local repository that will maintain the local services being offered by that device.

Traditionally, service discovery protocols have focused on services provided by devices, such as printers, fax-machines, cameras, audio-systems, etc. With the increasing popularity of handheld devices, and the support for mobile commerce, we envision an entirely new set of possible services. These services would be device independent and could vary from resources such as games or music, personal information such as professional expertise, information-oriented services such as maps or weather, or commerce based services like tickets to a movie. Users would be able to package their own resources or information as services and offer them to others. This requires services to have simple and rich description capabilities along with support for cross-platform lightweight usability. Konark defines an XML-based simple and rich description language to describe these kinds of services. This language is similar to WSDL and enables the description of a wide range of services.

Another important issue in service discovery protocols is how to make service information available to other devices in the network. Konark supports both advertisement and discovery. The service providers can push their services into the network. The rate of advertising depends on several factors. It can be on a periodic basis or can be stimulated by events such as a new device joining the network. Advertising can be also based on other factors such as geographical or temporal information, that is, location and time based advertising. We assume that the network formed supports multicasting so that service advertisement requests sent out will reach all nodes in the multi-hop network. These service advertisements contain time-to-live (TTL) information and help to keep the system up-to-date and robust. The clients can cache this service information to use it later. This caching can be based on the user preferences in the form of filtering techniques or the device capabilities like memory. Prospective users may also need to locate services and can use a distributed pull method to retrieve desired services in a timely fashion.

With the increase in the number and variety of services as well as the network size, the information obtained about available services - either via actively discovering them or by passively caching the advertisements - can be in large quantities. Also, the kind of m-commerce oriented services being targeted requires a high level of user interaction. This makes it necessary to have a registry that stores service information in an extremely manageable and user-friendly fashion. Another advantage of this feature would be the support for possible human interaction during advertising and discovery. To handle these requirements, Konark presents a service registry based on a tree-structure. We use a basic tree skeleton with service classification levels that are generic at top and become more specific, as we move down the tree levels. The nodes in the tree are not services themselves but act as placeholders for services in that particular category. Discovery can be done at any level of the tree – the broader the range of services desired, the higher the level of the tree- using the tree-paths. Advertising can be done at various levels of the tree – from generic advertising at the higher levels of the tree (such as all games) to very specific at the leaf node (e.g. chess). The same tree is exposed to the users as an intuitive and effective user interface to help them easily manage and use the services interactively, which gives a clear view of the available services.

Yet another important design issue is the actual delivery of services. As more and more manufacturers push out handheld devices into the market, the heterogeneity of ad-hoc networks vis-à-vis platforms increases. To support this trend, it is critical to support cross-platform service delivery. Konark uses the widely accepted standards such as HTTP and SOAP to handle service delivery. It provides for a micro-HTTP server on each device that can handle service requests from clients. The service requests and responses are based on SOAP.

III. KONARK ARCHITECTURE

Konark enables each device to act as a server and a client simultaneously. We use the term "client" for any device that is

interested in using services being offered by peers in the network. We use the term "server" for any device whose service is being used by peer nodes in the network. Konark defines components that allow devices to assume such a dual role. Each device includes a *Konark Application* that facilitates human interaction to initiate and control advertising, discovery, and service usage. It also includes *SDP Managers* and *Registry* that together maintain service objects and information about services offered by peers in the network. A *micro-HTTP* server is present to handle service delivery requests. Konark's two main parts - service discovery and service delivery - are constituted by these components.

A. Service Discovery

Each device has a Konark SDP Manager that is central to the service discovery mechanism. It discovers the required services on behalf of Konark applications, and also registers and advertises device's local services. Figure 1 shows the Konark service discovery protocol stack. Konark SDP Manager interacts with the messaging layer to send and receive the discovery and advertisement messages, and the messaging layer is built above the transport layer. The service discovery part of Konark deals with interface with underlying networks, service registry structure, and service discovery/advertisement mechanism.
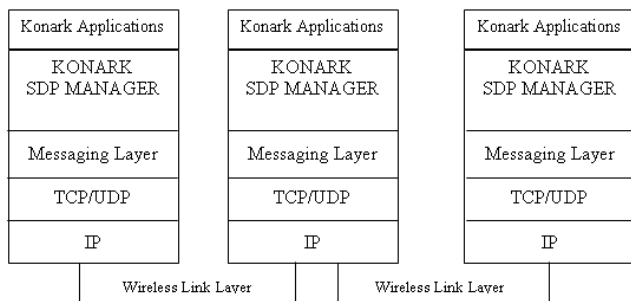

Figure 1. Konark Service Discovery Stack

1) Networking and Addressing Assumptions

The devices with local wireless connectivity form an ad-hoc network, when they come in each other's vicinity. We assume an IP level connectivity between these devices over any wireless link like IEEE 802.11 or Bluetooth. In the absence of the administrative services, these devices obtain an IP address by automatic configuration mechanisms [2] [6]. Most of the current operating systems like Windows-CE already support this technique. If the network is made up of multiple overlapping radio cells, we assume that each node has a routing capability to form a wider ad-hoc network. All these devices join a locally scoped multicast group for Konark – a multicast address out of link-local range 239.255.0.0/16 [10]. This enables peer-to-peer networking among these devices.

2) Service Registry

Service registry is a structure that enables devices to store their local services. It also allows them to maintain information about services that they might have discovered or received via advertisements. The *Konark SDP Manager* maintains a tree-based structure as registry. It allows the tree paths to be directly

used in service query and advertisement messages for service scoping or classification. Also, it enables direct correlation between the internal representation and user interface, which results in smaller footprint for resource-poor mobile devices.
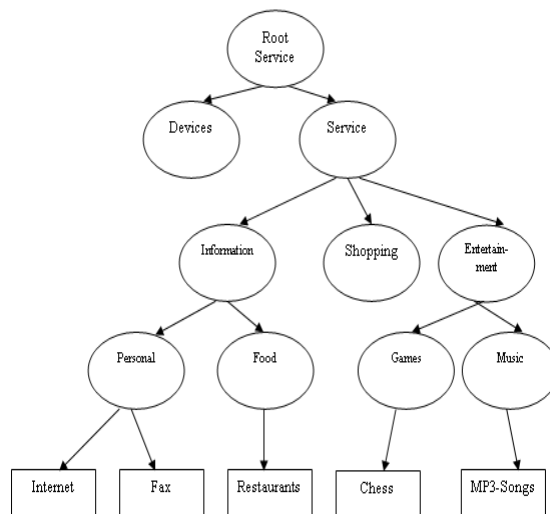

Figure 2. Example Service Tree

The tree has a number of levels that represent service classification. As we move down the tree from root to the leaves, services become more specific. An example service tree is shown in the figure 2. The oval shaped nodes represent generic service types that form a basic service tree. Services shown in rectangles are actual services and can be added under any generic service node based on their classification type. These actual services can either be offered by the device itself or from other nodes in the network. Services can be further classified as "all", "generic" or "specific" based on the tree-level. As an example, services identified at the root node (Path=RootService:Services) will signify "all" services in the tree. Service at any of the intermediate node will signify "generic" services. These include all the actual services present below that node down to the bottom. Actual services are called "specific" services and correspond leaf nodes of the tree.

The service tree is useful to devices when acting as client as well as server. This tree is used by a server to register local services that it wishes to offer, to advertise the registered services at any level, i.e., "all", "generic" or "specific", and to respond to client discovery requests. Clients use the tree to discover "all", "generic", or "specific" services, and manage these services. Use of inclusive or exclusive filtering options at different levels of the tree makes it easy for users to manage services of interest (Refer to the following sub-section for detailed descriptions). Device capabilities like memory size can also be used as a parameter to set the filters to add any discovered or advertised service into the tree.

Each service is associated with a lease time, that is, the time for which the service is expected to remain available. This time is specified as time-to-live (TTL), which is part of service registration or advertisement information. Services should be refreshed before their TTL expires. Otherwise, they are

removed from the registry. This scheme makes the discovery system robust against unexpected failures.

3) Service Discovery and Advertising

To discover services in the network, clients use a discovery process known as *active pull* mechanism. Servers use an advertisement process to periodically announce their registered services. This mechanism is termed as *passive push*. Konark supports both push and pull mechanisms, so that both clients and servers can discover and advertise services on a need basis.

Discovery process has two steps. In the first step, a client sends out a discovery message on a fixed multicast group. In the second step, all the servers that have the service being sought would respond. To accomplish the first step, the client creates a discovery message that contains either path from the service tree or keyword. The path is used, when the client desires "all" services in the network or services defined by some "generic" service type. For the discovery of a particular service ("specific"), keyword will be used. The message also has the port number where the client listens for the server replies by unicast. Figure 3 shows the discovery message.

Path or Keyword	PORT

Figure 3.   Service Discovery Message

On receiving a discovery message sent out on a fixed multicast group by the client, each server node performs service matching. If it is a path-based discovery, the server matches the path with its registry tree and gets all the registered services under that node. If the discovery request is based on a keyword, the server matches this word with each local service's keywords (The keywords is an element of service description as explained in the following service delivery section). If the server finds a match, it creates a service advertisement message for each match found. This message contains the actual service name, the path of the service, the type of the service, the URL where the service description will be available and the time-to-live of the service specified in minutes. Figure 4 shows the service advertisement messages.

Service Name	Path	Type	URL	TTL

Figure 4.   Service Advertisement Message

Similar to the discovery process, advertisement can also be based on "all", "generic", or "specific" services. The server can advertise "all" its registered services, "generic" services identified by some path in the service tree, or a specific service defined by a leaf node. On receiving a service advertisement messages, the clients pick up the path and match it with their registry tree. While doing so, if they find that there is an exclusive filter on any of the node of the path, the service is discarded. If not, it is added under the specified path. If the path has an inclusive filter set, it notifies the user of this new service.

B.   Service Delivery

Service delivery consists of a two-step process – *service description* where the client learns about the properties and

capabilities of the service, and *service usage* where the client avails the capabilities. Before we explain any of these steps, it is imperative to understand our service description language, as both depend upon it.

1)   Service Description Language

Under Konark architecture, each service is a bundle of two components – a service description file that describes a service, and a service object. The registry of any server device contains these two components for each service being offered by that device. While the service object can be in the form of a class file or a DLL, the description file is a plain text file containing complete information about the characteristics and functions of the service. Konark defines an XML based service description language to enable services to explain their characteristics. Figure 5 defines the various tags defined in the aforementioned language.

```
<Service>
    <-- Name of the service -->
    <ServiceName></ServiceName>
    <-- Type of serivvce -->
    <ServiceType></ServiceType>
    <-- Keywords – For matching search requests -->
    <KeyWords>
        <Word></Word>
    </KeyWords>

    <-- Properties: Characteristics of the service -->
    <Properties>
    <-- Properties: A combination of name, user-friendly description,  and value -->
        <Property>
            <Name></Name>
            <Descrition></Description>
            <Value></Value>
        </Property>
    </Properties>

    <-- Available functions -->
    <Functions>
        <Functoin>
            <-- Name of the function in the service object -->
            <Name></Name>
            <-- User friendly description -->
            <Description></Description>
            <-- Parameters required to invoke the function -->
            <Parameter>
                <Name></Name>
                <Type></Type>
                <Description></Description>
            </Parameter>
            <-- Return type for the function -->
            <ReturnParameter>
                <Name></Name>
                <Type></Type>
                <Description></Description>
            </ReturnParamenter>
        </Function>
    </Funcions>
</Service>
```

Figure 5. Service Description Language

The root of the document is the *Service* tag. It represents the start of the service definition. The first child is *ServiceName*. It is used to define a user-friendly name for the service. This is followed by *ServiceType*, which defines the type of the service, e.g. printer or music. The service type along with the relevant path in the tree maybe used to advertise the service. The third child is *Keywords*. These words may describe the service type or some characteristics of the service, e.g., for a service offering opportunities to print documents, the *ServiceType* could be "print", while the *Keywords* could include "laser printers", "color printers", or "printing".

While the first three children of the *Service* tag are primarily used in service discovery and advertisement, *Properties* and *Functions* are used in service description and delivery. The *Properties* describes the characteristics of the service. Each service can have any number of properties. Each property is a combination of a name, a description, and a value. The *Name* of the property is used for communication between server and client. Using this name, the client application may be allowed to subscribe to events related to this property, e.g., the client may be interested in being informed when the property changes to a particular value. The *Description* is a user-friendly explanation of the property. The *Value* gives the current value of the property. The *Functions* are related to the actual invocation of the service. A service can have any number of functions. Each function is a combination of a name, a description, parameters, and a return parameter. The *Name* of the function is the name given to the actual method in the service object. The *Description* is an explanation to the user as to what action the function provides. The *Parameter* represents arguments needed to invoke a function. Similar to properties, a parameter also contains a name, a description and a type. While the *Name* is for communication between server and client, the *Description* is to guide the user as to what information is required to invoke the function. The *Type* specifies the data type for the argument, e.g., type could be string, integer, or file. The client application uses the *Type* information to enforce the validity of user input. Each function also includes a *ReturnParameter* tag. This is similar in structure to the *Parameter* tags. It represents the information obtained on invoking the function.

The language described above is loosely based on WSDL (Web Services Description Language) [21], the emerging standard for describing web services. Since our description language is designed considering resource-poor mobile devices, it may not be as deep and powerful as WSDL. But it has its own merits of simplicity and a wide range of services that can be described.

2) Service Delivery Architecture

Service delivery primarily involves communication between a *Client Application* and the *micro-HTTP server* of the service provider as illustrated in figure 6.
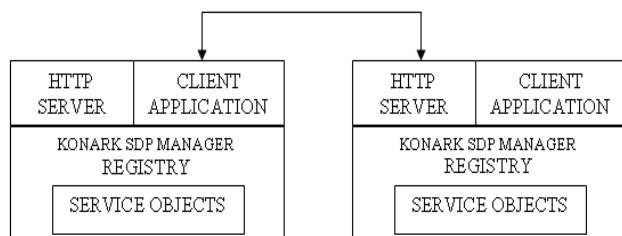

Figure 6. Service Delivery Components

Once a service has been discovered by a client device, the client has very limited knowledge about the service. It only knows the user-friendly name of the service, service type, IP address of the device offering the service, and the duration of availability of the service. Based on this information, the user of the client device may seek to get more information of the

service. This stage is known as service description, and is the first step in service delivery. The client device follows the URL obtained during the discovery phase for more details. A typical URL would be similar to http://169.254.30.42/music.xml. This implies that a device with IP address 169.254.30.42 is offering some service whose description can be found in the file music.xml.

```
POST 169.64.32.10 HTTP /1.1
SOAPACTION  Music for you!
(other HTTP Headers)

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
<SOAP:Body>
<MethodName>GetSong</MethodName>
<Parameters>
        <Parameter>clip.wma</Parameter>
</Parameters>
</SOAP:Body>
</SOAP:Envelope>
```

Figure 7. Service Delivery Request

The service description file contains complete information about the properties of the service and the methods provided by the service. The user can interact with the service by invoking any of the available functions with proper parameters. The user function invocation is packaged as a SOAP [20] request and sent to the micro-HTTP server. Figure 7 shows an example of a method invocation request.

IV. IMPLEMENTATION

Our Konark architecture is operating system and programming language independent, since it provides a framework in which services are described and delivered using open standard XML technology over IP network connectivity. We have implemented our prototypes in two versions of Java, i.e., Personal Java 1.2 [14] and J2ME CLDC/MIDP [16]. Personal Java 1.2 is JDK1.1.8 compliant. For this implementation, we used Pocket-PC 3.0 based iPAQs from Compaq and the Jeode VM from Insignia Inc [9]. These iPAQs have SA-1110 processor from Intel. Lucent WaveLAN cards for IEEE 802.11 interface are used to form an ad-hoc network among the iPAQs. Our second prototype is for J2ME CLDC/MIDP platforms on devices like Motorola iDEN phones.

We provide a set of APIs, so that actual services can be built easily. The API set includes service advertisement and discovery, service description and invocation, service registry management, service lease management, several utility, and user interface APIs. Using these APIs that map Konark functionalities, we built the prototype shown in figure 8 to demonstrate that they are designed to well address all the needs of real ad-hoc applications. Figure 8 (a) shows registered service "Song for you!" under "Music" and available service "Print Your Files" under "Personal". The client device is about to discover service under "RootService:Information:Food" as shown in figure 8 (a). *Services* and *Tree Options* menus together provide all the functionalities of discovery, advertisement, service registration and exclusive/inclusive filter setting. Figure 8 (b) shows invocation of action on a

"Song for you" service by another client iPAQ. This comes under service delivery phase. The user can enter proper parameters for the function a dialog box shown, and can invoke a function to get a music file from the iPAQ of figure 8 (a). We implemented all the functionalities of service discovery and delivery through this intuitive, simple user interface.

Figure a – Discovering "Food" Service   *Figure b – Invoking Action on a Service*

Figure 8. A Prototype Implementation

V. RELATED WORK AND ON-GOING WORK

There have been intensive research efforts in the field of service discovery protocol. IETF SLP [4] [5], Sun's Jini [15], UPnP [17], and Salutation [13] are some of the frontrunners in this area. Being developed assuming the availability of stable (wired) infrastructure support, most of them are mainly targeting relatively static networks and bigger size networks - up to enterprise network. They (except for UPnP) introduce a central service registry that maintains all the service information available in the network. Typically, it is hosted by a relatively powerful device with wired network interface. Services are discovered and advertised through the central service registry among clients and service providers. Although the service discovery protocols are successfully dealing with the need of dynamic, impromptu service discovery for mobile devices, the network environment for which they are designed is more static than the one that Konark aims. As discussed in section II, it is not appropriate to presume the device(s) stable and powerful enough to play the role of the central service registry, i.e., the infrastructure support, in highly dynamic ad-hoc network.

Unlikely other major service discovery protocols, UPnP may be viewed as closer to Konark, considering its distributed service discovery protocol and its leverage of open standards such as HTTP and SOAP; Targeting relatively smaller networks, e.g., home networks, it introduces no central service registry. In addition, service discovery requests are made on demand, and services are advertised as desired and when necessary. In this sense, UPnP is closer to Konark than others, sharing several features with Konark.

IBM has developed DEAPspace [12] [8] that addresses the service discovery problem in wireless single-hop ad-hoc networks. Each device in DEAPspace maintains a view of all the services present in the network, and periodically exchanges its view of the world, i.e., the full list of its service map, with its neighbors. The periodic broadcast is scheduled in a proactive way in that when a device finds its local services absent in messages broadcast by its neighbors or about to expire, it schedules its broadcast sooner than usual. They showed the timely convergence of their approach compared to regular periodic broadcast scheme.

After having successfully developed and demonstrated a service discovery and delivery framework for mobile devices in ad-hoc network, our Konark project is now in the second phase in which our main focus is on an efficient service discovery protocol to exploit the very nature of highly dynamic ad-hoc networks.

The DEAPspace's target environment and goal seems similar to Konark's, since it envisions dynamic ad-hoc networks and pursues a decentralized solution to the problem. However, the DEAPspace confines itself to a small network by assuming a single-hop ad-hoc network and broadcasts fit in a single message. Its primary goal is faster convergence of service information available in the network. But we argue that the effort to maintain airtight convergence in such a highly dynamic ad-hoc network may not be paid off or it may be impossible to attain. Instead, we believe that striking a balance between the convergence time and network bandwidth use is more desirable in a dynamic environment.

Our new algorithm for fast, distributed service discovery is a kind of gossip protocols. Unlike the DEAPspace algorithm, each node multicasts a subset of its knowledge, i.e., a relevant part of services in question. In other words, when a node sees a service message on Konark multicast address, it multicasts the difference of its own view of relevant services and services in the received message. The multicast is scheduled to wait a random time, so that the network can avoid the multicast storm of immediate, simultaneous multicasts and have time to learn of available services. Then, nodes without any new information other than multicasts by others up to their scheduled point will be able to cancel their multicast. This way we can reduce network bandwidth usage due to unnecessarily repeated multicasts found in DEAPspace and plain periodic advertisement schemes. (We note that a significant amount of the exchanged information would be a waste, since nodes in such a network will likely have almost the same service information.) Also, the multicast round of the delta message is triggered by a service query message or service (re-)advertisement message in the absence of query messages within a certain time window. As a result, our delta gossip protocol achieves fast convergence of service information efficiently and in a distributed way without incurring excessive advertisement traffic.

The Konark aims at wireless multi-hop ad-hoc network that will be usually larger than the one the DEAPspace can support. It implies that the service information exchange messages may exceed broadcast message length limit. In this case, our protocol randomly chooses services up to the number of service instances that a single multicast message affords, hoping other nodes covers the rest. This distributed and

cooperative nature of our algorithm will provide robustness that a discovery system for a highly dynamic ad-hoc network mandates.

VI. CONCLUSION

We designed a framework for service discovery and delivery suited to dynamic ad-hoc networks for device independent services. Although it can be implemented on any operating system and in any programming language, our prototypes are for two Java versions, namely, Personal Java 1.2 and J2ME CLDC/MIDP. All service discovery and delivery functionalities are embodied in a set of APIs, including service advertisement and discovery, service description and invocation, service registry management, service lease management, several utility, and user interface APIs. We also demonstrated that actual services could be easily built using these APIs.

We are currently in the second phase of our Konark project. A new efficient service discovery algorithm is being explored and analyzed through both a real implementation and simulation. For our simulation, we are writing ns-2 simulation modules based on a ns-2 simulator with wireless multicast augmented [22].

REFERENCES

[1] D. Buszko, W. Lee, and A. Helal, "Decentralized Ad-Hoc Groupware API and Framework for Mobile Collaboration," Proceedings of the ACM International Conference on Supporting Group Work (Group'01), September 2001.

[2] S. Cheshire and B. Ababa, "Dynamic Configuration of IPv4 Link-Local Addresses," IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-zeroconf-ipv4-linklocal-04.txt, 2002.

[3] N. Desai, "Konark – An Ad-Hoc Service Discovery Protocol," Master's Thesis, University of Florida, August 2002.

[4] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," IETF RFC 2608, June 1999. http://www.ietf.org/rfc/rfc2608.txt.

[5] E. Guttman, C. Perkins, and J. Kempf, "Service Templates and Service: Schemes," IETF RFC 2609, June 1999. http://www.ietf.org/rfc/rfc2609.txt.

[6] M. Hattig, "Zeroconf IP Host Requirements," IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-zeroconf-reqts-09.txt, 2001.

[7] A. Helal, B. Haskell, J. Carter, R. Brice, D. Woelk, and M. Rusinkiewicz, "Any Time Anywhere Computing: Mobile Computing Concepts and Technology," Kluwer Academic Publishers, ISBN 0-7923-8610-8, October 1999.

[8] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner, and A. Schade, "DEAPspace – Transient Ad-Hoc Networking of Pervasive Devices," Computer Networks, vol. 35, pp. 411-428, 2001.

[9] Insignia Inc, Jeode PDA Edition, http://www.insignia.com/content/products/pda.shtml, 2002.

[10] D. Meyer, "Administratively Scoped IP Multicast," IETF RFC 2365, July 1998.

[11] B. Miller and R. Pascoe, "Salutation Service Discovery in Pervasive Computing Environments," IBM Pervasive Computing white paper, http://www-3.ibm.com/pvc/tech/salutation.shtml, February 2000.

[12] M. Nidd, "Service Discovery in DEAPspace," IEEE Personal Communications, August 2001.

[13] The Salutation Consortium, Salutation Architecture Specification, http://www.salutation.org, October 1997.

[14] Sun Microsystems, Personal Java Specifications, http://java.sun.com/products/personaljava/, 1997.

[15] Sun Microsystems, JINI technology, http://www.sun.com/jini, January 1999.

[16] Sun Microsystems, "Java 2 Micro Edition Technology for Creating Mobile Devices," Sun white paper, http://java.sun.com/j2me, February 2001.

[17] UPnP Forum, www.upnp.org, 2000.

[18] V. Verma, "Konark – A Service Delivery Protocol," Master's Thesis, University of Florida, August 2002.

[19] W3C Consortium, XML Architecture Domain, http://www.w3.org/XML, 1998.

[20] W3C Consortium, SOAP Architecture Domain, http://www.w3.org/TR/SOAP, 2000.

[21] W3C Consortium, WSDL, http://www.w3.org/TR/wsdl, 2001.

[22] Wireless Multicast Extensions for ns-2.1b8, http://www.monarch.cs.rice.edu/multicast_extensions.html, 2000.