RESOURCE AND SERVICE DISCOVERY IN WIRELESS AD-HOC NETWORKS WITH AGILE COMPUTING

Niranjan Suri[1,2], Matteo Rebeschini[1], Maggie Breedy[1], Marco Carvalho[1], and Marco Arguedas[1]
[1]Institute for Human & Machine Cognition
[2]Lancaster University
{nsuri,mrebeschini,mbreedy,mcarvalho,marguedas}@ihmc.us

ABSTRACT

Resource and service discovery in tactical networking environments is a challenging problem. The ad-hoc and peer-to-peer nature of the network invalidates many traditional approaches to resource and service discovery that rely on registries. The agile computing middleware supports opportunistic service and resource discovery and tasking in tactical environments. The group manager component of the middleware supports bandwidth-efficient discovery of neighboring nodes, their resources (including CPU, memory, storage, and connectivity), and their services. Nodes may join one or more groups that allow resources and services to be categorized into related sets. Applications using the group manager are notified about the appearance and disappearance of peers as well as changes in memberships of groups. The group manager is efficient in finding neighboring services and is designed to improve the probability of finding resource-rich nodes. Cross-layer integration between the group manager and the MANET layer reduces the network overhead and improves performance.

INTRODUCTION

Agile computing is an approach to improving the performance of systems in highly dynamic environments by opportunistically discovering and exploiting resources [1] [2]. The agile computing middleware is designed for tactical environments with wireless ad-hoc networks, where resources appear and disappear unexpectedly. One of the fundamental requirements for the middleware is the ability to discover the nodes that are currently reachable and the resources available at those nodes. The group manager component provides this capability to the agile computing middleware.

The group manager supports discovery of neighbors and the resources and services available at neighbors. In addition, a search mechanism supports discovery of resources and services at non-neighboring nodes. The design principles of agile computing imply that finding resources and services that are nearby is better than finding those that are farther away (in terms of network hops). Therefore, one optimization criterion for finding nodes is network proximity. A second optimization criterion is finding nodes that are resource rich or have excess capacity. Both of these criteria are incorporated into the approach realized by the group manager.

Groups are used to partition nodes in the network into different sets, thereby placing constraints on resource sharing. Even if nodes A and B are network neighbors and node B has spare resources, the group manager will not facilitate sharing resources between A and B if they do not belong to the same group. The group manager supports the notion of managed groups and peer groups. , but this paper focuses on peer groups. In addition, passwords may be used to restrict group membership as a security mechanism.

While the group manager directly supports the requirements of the agile computing middleware, it is designed to be generic and operate as a standalone component. A simple API (application programming interface) allows applications to take advantage of the capabilities offered by the group manager.

The group manager can optionally operate in an integrated fashion with an extended MANET implementation. This cross-layer integration reduces the network overhead incurred by the group manager thereby improving performance.

GROUP SEMANTICS, TYPES, AND OPERATIONS

Figure 1 shows the different types of groups that are supported. Fundamentally, groups are divided into managed groups and peer groups. A managed group is created by a particular node and is owned by that node. Other nodes may join a managed group, which allows the node that created the group to have access to nodes that join. The creator of the group may optionally protect the

group with a password, which prevents nodes that do not have the password from joining the group.

On the other hand, peer groups do not have an owner or manager and there is no mechanism to join a peer group. Nodes that create groups with the same name will implicitly be members of the same peer group. Peer groups may also be protected with a password.

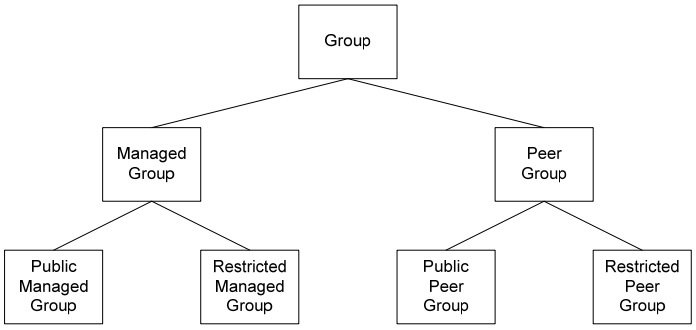

Figure 1: Supported Types of Groups

The group manager does not use a centralized registry but instead maintains node membership independently from the perspective of each node. This design choice also implies that there is no attempt at maintaining a consistent group view for all nodes that are members of a group.

Consider the example in Figure 2. Nodes 1 and 2 create peer group A. Node 3 creates peer group B. Node 4 creates peer groups B and C. Node 5 creates peer group A, B, and C. Node 6 creates peer group C. The resulting groups and members are shown in the figure. Note that since these are peer groups, the nodes did not have to explicitly join the groups. When Node 6 creates peer group C, nodes 4 and 5 automatically become members of group C. Likewise, nodes 4 and 5 are informed that node 6 has joined as a member of group C. Note that this example assumes that all nodes can reach each other via the network.

Group membership is influenced by network visibility. As described earlier, there is no attempt to maintain a consistent view of a group from the perspective of each of the members. Figure 3 shows an example with four nodes that all create a peer group named XYZ. Since Node 1 cannot see Nodes 3 and 4, the group membership list for group XYZ at Node 1 consists of only Nodes 1 and 2. On the other hand, Node 2 can see all the nodes, so from the perspective of Node 2, the membership list for group XYZ contains Nodes 1, 2, 3, and 4. Finally, the membership list for group XYZ on Node 3 consists of Nodes 2, 3, and 4.

Nodes may attach additional application-specific data to the groups they create. In the context of agile computing, this mechanism is used to propagate information about resource and/or service availability. Applications may change this information at any time and the changes are propagated to the neighboring nodes.

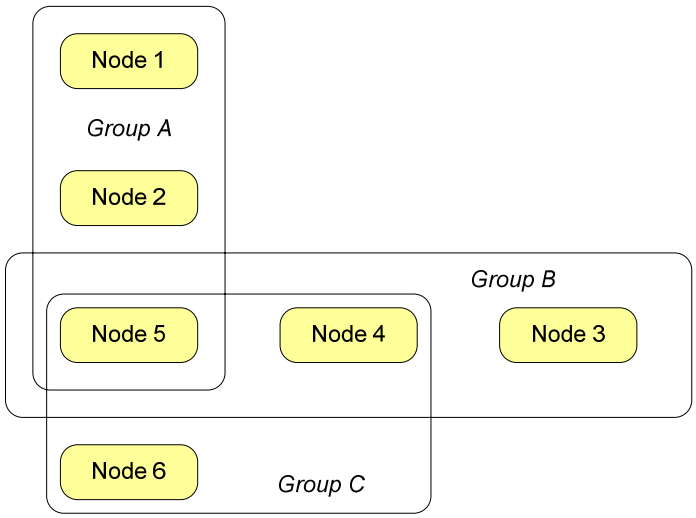

Figure 2: Example of Overlapping Peer Groups

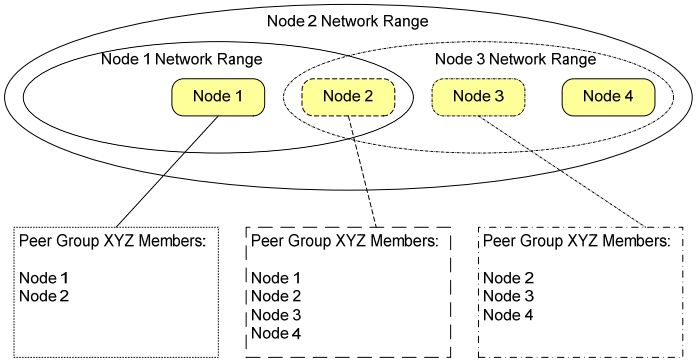

Figure 3: Peer Group Membership and Network Range

Nodes may also initiate a search to find other nodes that satisfy their requirements. The semantics of the search request and reply are application defined – the group manager simply treats it as an array of bytes. Applications may use this mechanism to look for services or to find other nodes that have spare resources. The search algorithm is described in the next section.

PEER-TO-PEER SEARCH AND INFORMATION PROPAGATION

The search algorithm implemented by the group manager is a peer-to-peer search that is loosely based on the Gnutella search protocol [3]. The search request is qualified by a group name, which limits the search to be processed only by other nodes that are also part of the specified group. For flexibility, the search term is a byte buffer and the content is opaque to the group manager.

The search term is forwarded by the group manager to neighboring group managers, which pass the term onto the applications on those nodes. If a peer application wishes to respond to a search request, the application provides a response term to its local group manager. The response term is also an opaque byte buffer that is transmitted back to the original node that initiated the search.

The hop count and flood probability are additional parameters that are used to constrain the search operation. The hop count limits the number of hops through which the search request is propagated. When a node receives a search request and the hop count is greater than zero, the node decrements the hop count and retransmits the search request with the probability specified by the flood probability parameter. A probability of 100 implies that the node will always retransmit the request (provided the hop count is greater than zero). In parallel, the node will process the search request locally.

The peer search also provides an option for performing persistent searches. When initiating a search, the application may specify a time to live parameter. A value of 0 indicates that the search should be performed once and then discarded. A negative value indicates that the search should persist indefinitely. Finally, a positive value specifies the number of milliseconds for which the search should continue.

Persistent search requests are kept at nodes until their time to live expires. While a persistent search is valid, if a new node appears on the network, its neighbors will forward any active persistent search requests on to the new node (provided the hop count parameter allows the request to propagate to the new node). When the local state of an application changes, it can notify the group manager via a method call, which will cause the group manager to re-execute any persistent searches that are still valid.

At the implementation level, persistent search requests are implemented using a lease mechanism. If the application on a node starts a persistent search with a long or indefinite lifetime, the neighboring nodes will only keep the search for a predefined maximum lease time. The original node has to periodically renew the persistent query with the other nodes. This prevents orphaned persistent queries beyond the lease expiration time.

The group manager also supports proactive propagation of information. This capability can be combined with the peer-to-peer search, which is reactive. Applications can attach data to any group they create, which will be propagated to all other nodes that are visible to the node. If the local information changes, the application can provide updates to the group manager, causing the new updated information to also be propagated. A configuration parameter limits the maximum frequency with which updates are transmitted.

Network visibility depends on the configuration of the group manager. In the simplest case, the visibility is limited to nodes that can be directly reached via a network broadcast. The group manager also supports relaying information received from one node onto another node, thereby increasing the number of nodes that become aware of the existence of a node and the groups created by the node. Each node can control the number of hops over which its information should be transmitted, thereby controlling the visibility of the node. Resource rich nodes can increase their hop count, thereby making it easy for other nodes in the network to locate them. Figure 5 shows the network visibility of Node 4 with respect to other nodes. In 4 (a), the information from Node 4 is only propagated for one hop. Therefore, only Nodes 2, 5, 6, and 7 are aware about Node 4. In 4 (b), the information about Node 4 is retransmitted by Nodes 2, 5, 6, and 7, thereby reaching every node in the network except Node 10.

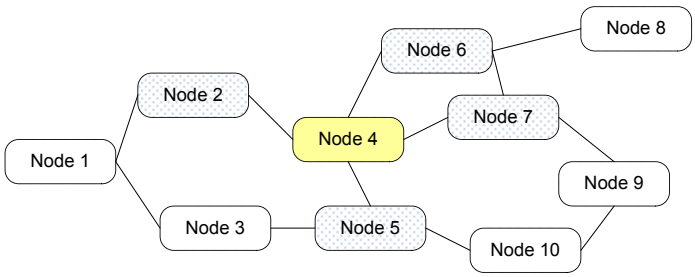

Figure 4 (a): One Hop Information Propagation

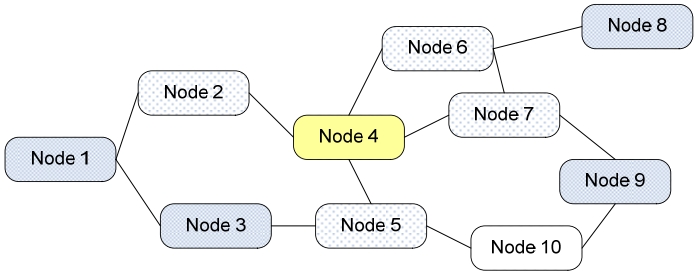

Figure 4 (b): Two Hop Information Propagation

It is important to note that the information propagated and searched for is application defined and opaque to the group manager. Moreover, the information does not need to be the same. For example, an application can choose to propagate resource availability information and use the search to find services (or vice versa). The group manager's ability to support both modes provides flexibility to the application to pick and choose between proactive versus reactive modes of operation at the same time. For example, often needed information can be

proactively pushed whereas information that is needed rarely can be reactively queried.

APPLICATION PROGRAMMING INTERFACE

While the group manager was conceived to support the agile computing middleware, it is an independent component that can be used by other applications as well. The API for the group manager is simple and flexible and consists of two sets of methods. The first set of methods allows an application to invoke operations on the group manager. These are detailed in Figure 5 (a). The second set of methods allows the group manager to call back into the application. These are detailed in Figure 5 (b).

The createPublicPeerGroup() and createRestPeerGroup() methods allow the application to create a group. The group is identified by a name, which is unique for the node. An optional parameter may be attached that allows the application to provide additional information. The hopCount specifies the number of hops through which the information about this peer group should be propagated. In the case of the restricted peer group, the password is used to ensure that a node only accepts others nodes with the same password as members of the group. The removeGroup method deletes the specified group

Applications may attach any additional information they wish to the group using the data argument. In the agile computing middleware, this parameter is used to propagate resource and service availability information. The updatePeerGroupData() method can be used by the application to update the data at any time, which is propagated to other nodes.

The last four methods are related to the peer search mechanism. The startPeerSearch() is used by an application to start a search. Each search request is identified by a UUID that is returned to the application as a result of the startPeerSearch() method. The parameters are discussed in the previous section on peer-to-peer search. An application can terminate a persistent search by invoking the stopPeerSearch() method and passing in the UUID that identifies the search.

An application that receives a search request (via the callback API) may choose to respond to the request by using the respondToPeerSearch() method. The response is a byte array that is passed back to the original application that initiated the search.

Finally, an application may notify the group manager about a change in status with respect to a specified group via the statusChanged() method. This will cause the group manager to re-execute any active persistent queries so that the application can generate replies if appropriate.

```
createPublicPeerGroup (groupName, data,
                       hopCount)
createRestPeerGroup (groupName, password,
                     data, hopCount)
removeGroup (groupName)

updatePeerGroupData (groupName, data,
                     hopCount)

startPeerSearch (groupName, hopCount,
                 floodProbability,
                 searchParam, timeToLive)
stopPeerSearch (searchUUID)
respondToPeerSearch (searchUUID,
                     responseParam)

statusChanged (groupName)
```

Figure 5 (a): Group Manager API

```
newPeer (uuid)
deadPeer (uuid)

groupListChange (uuid)

newGroupMember (groupName, memberUUID, data)
groupMemberLeft (groupName, memberUUID)

conflictWithRestPeerGroup (groupName,
                           peerUUID)

peerGroupDataChanged (groupName, peerUUID,
                      data)

peerSearchRequestReceived (searchUUID,
                           searchParam)
peerSearchResultReceived (searchUUID,
                          responseParam)
```

Figure 5 (b): Group Manager Callback API

The callback API is used by the group manager to notify the application when events occur. The newPeer() and deadPeer() methods are used when a new peer is detected and when a previously detected peer is deemed to be dead. The groupListChange() method is used to indicate that one of the peers has changed its list of groups.

If an application is only interested in nodes that are members of local groups, then they can use the newGroupMember() and groupMemberLeft() callbacks. These identify the local group name and the UUID of the peer that has joined or left the group. In the case of a restricted peer group, it is possible that another node created the group with a different password. This will result in the application being notified via the conflictWithRestPeerGroup() method.

If a peer node changes the data associated with a group (and the peer node is a member of the group on the local node), the changed data is passed to the application via the peerGroupDataChanged() method. This facilitates receiving information that is being proactively propagated by other nodes.

When a node receives a peer search request from another node, the application is notified via the peerSearchRequestReceived() method. The application is passed the search parameter, which is a byte buffer. Finally, when other applications respond to a peer search request, the response is provided back to the original node via the peerSearchResultReceived() callback.

The group manager provides several additional functions that can be used to configure the default behavior. This includes changing the frequency with which the group manager transmits packets of different types (see next section) and the number of missing packets before a node is considered to be no longer reachable.

IMPLEMENTATION DETAILS

The current implementation of the group manager operates over UDP. Both broadcast and unicast messages are sent by the group manager to communicate with other group mangers. Six different packets are transmitted at various points in time and at various frequencies. The PING packet is the most basic packet and contains the UUID and IP address for the node, and hop count and flood probability fields.

The INFO packet contains additional information about a node and the groups present at the node. It a user-friendly node name, the public key, a state sequence number, and the ping interval.

Receiving an INFO packet allows a node to obtain all the information it needs about a remote node. Receiving the PING packet allows a node to determine that the remote node is still alive and reachable over the network. By default, the INFO packet is transmitted every 10 seconds and the PING packet is transmitted every 2 seconds between the INFO packets. However, receiving an INFO or PING packet from a new node causes other nodes to immediately transmit an INFO packet. This allows the new node to learn quickly about existing nodes.

When an application changes the data associated with a group, the group manager generates and transmits a GROUP_DATA packet, which extends the PING packet. It contains the group name and the group data. By default, this packet is transmitted at most once a second, even if the application changes the data more rapidly. Transmitting a GROUP_DATA packet subsumes transmitting a PING packet.

If a node initiates a search, the node transmits a PEER_SEARCH packet or a PEER_SEARCH_RPG packet. The former is used for public peer groups and the latter for restricted (password protected) peer groups. The PEER_SEARCH packet contains the basic fields in a PING packet followed by a UUID to identify the search, the group name, and the search parameter. The PEER_SEARCH_RPG packet is similar, except that the search parameter is encoded using the password chosen for the group. This prevents other peers that do not have the same password from accessing the search parameter.

The last packet type is the PEER_SEARCH_REPLY packet. This is used by a node that is responding to a peer search initiated by another node. This packet contains the UUID of the node replying, the UUID of the search request, and the search reply, which is a byte buffer.

The group manager uses UDP broadcast to send PING, INFO, GROUP_DATA, PEER_SEARCH, and PEER_SEARCH_RPG packets. When a node receives one of these packets, the node may rebroadcast the packet based on the hop count and flood probability parameters in the packet. The PEER_SEARCH_REPLY packet is sent via a UDP unicast message back to the node that initiated the search.

PERFORMANCE ANALYSIS

Bandwidth utilization is one measure of performance of the group manager, which transmits several different packet types periodically and when requested by the application. Most of the packets are variable length and their size depends on application-defined data. Table 1 lists the packet types and their minimum sizes.

Packet Type	Size
PING	78
INFO	272
GROUP_DATA	86
PEER_SEARCH	320
PEER_SEARCH_RPG	324
PEER_SEARCH_REPLY	140

Table 1: Packet Types and Sizes

Except for the PING packet, all of the other packets are variable length. The size of the INFO packet depends on the length of the user-friendly node name, the number of groups, the length of the group name(s), and the group

data attached to the groups. The size of the GROUP_DATA packet depends on the length of the group name and the size of the attached data. The sizes of the PEER_SEARCH and PEER_SEARCH_RPG packets depends on the length of the group name as well as the length of the search parameter. Finally, the size of the PEER_SEARCH_REPLY packet depends on the length of the reply parameter.

In the Agile Computing middleware, the size of the INFO and GROUP_DATA packets are 492 bytes and 151 bytes respectively. These will be used to estimate bandwidth utilization in the calculations below.

In the quiescent state, with default settings, each node transmits one INFO packet and four PING packets within a 10 second interval. Therefore, the minimum bandwidth utilized by each node is 804 bytes per 10 seconds, or 81 bytes/sec. If the resource information at a node is changing rapidly, the node will transmit GROUP_DATA packets at a maximum rate of one every two seconds, effectively replacing the PING packets. Therefore, the maximum bandwidth utilization by each node is 1096 bytes per 10 seconds, or 110 bytes/sec. This scales linearly with respect to the number of nodes that share the wireless medium, resulting in the graph in Figure 6.

If a node uses a hop count larger than 2, other nodes receiving the packet will retransmit that packet one time. Since every node that receives the packet will retransmit the packet, this will result in n packets, where n is the number of nodes. Note that increasing the hop count further has no impact on the number of packets. Each node will retransmit the packet only one time. If all n nodes use a hop count larger than 2, the maximum number of packets that will be generated is n * n. The typical case is not n^2 because only nodes that have a large number of free resources use a hop count greater than 1. Figure 7 shows the maximum bandwidth utilization as a function of the number of nodes as well as the number of nodes requesting retransmission with a hop count greater than 1.

The peer search request has the same bandwidth utilization pattern – linear with respect to the number of nodes that are being searched. The search reply is linear with respect to the number of replies generated.

Considerable work has been done in the area of service discovery. Efforts range from centralized approaches such as UDDI [4] to hybrid approaches such as SLP [5] to many approaches specifically designed for MANETs. Approaches based on centralized registries are not well suited to MANETs for obvious reasons: they introduce a requirement for centralization in an otherwise distributed approach (not to mention the usual problems of creating bottlenecks and a single point of failure). These approaches also do not work with network partitioning, which is a typical occurrence in tactical environments.

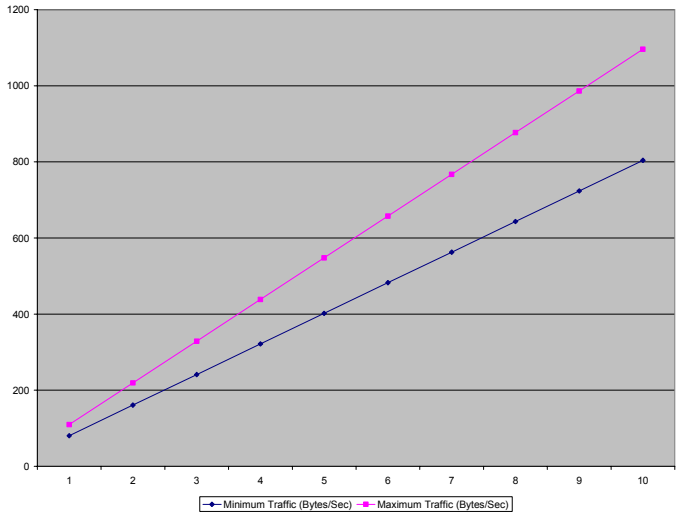

Figure 6: Minimum and Maximum Bandwidth Utilization (Upto 10 Nodes With 1 Hop Information Propagation)

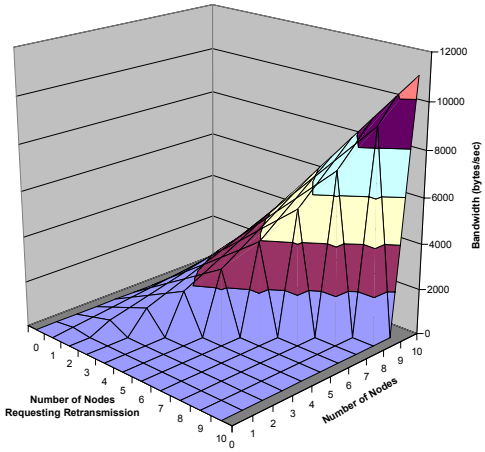

Figure 7: Maximum Bandwidth Utilization (Upto 10 Nodes With > 1 Hop Information Propagation)

RELATED WORK

The Service Locator Protocol (SLP) normally operates in conjunction with one ore more centralized Directory Agents, but could operate in a peer-to-peer mode also. However, SLP was designed for corporate networking environments and does not fit well into the MANET model. For example, there is no ability to incrementally search for services or resources that are farther away from the requestor.

Numerous distributed service lookup approaches have been proposed for ad-hoc networks. These may be broadly

classified into proactive, reactive, or hybrid approaches. In [6], the authors compare proactive and reactive approaches and conclude that reactive approaches are more efficient in terms of the number of messages exchanged, especially when combined with route discovery. In [7], the authors present a reactive protocol based on multicast, but it is not well suited to a MANET environment. In [8], the authors address scalability issues that arise when integrating several networks together, including wired infrastructures. Finally, in [9] and [10], the authors argue for an integrated approach that combines service lookup with ad-hoc routing, which results in a more efficient approach.

The group manager component presented in this paper is designed to be generic and support more than service discovery. In particular, it is used in the agile computing middleware to find resources and services in the network. The group manager is flexible by supporting a combination of proactive and reactive approaches and by delegating the task of matching search requests and generating responses to the applications. Enhancements allow applications to determine the extent to which their information should be propagated, allowing resource rich nodes or core services to increase the ease and likelihood of being found by other nodes.

FUTURE WORK

The current implementation of the group manager operates on top of UDP. While this is convenient given the ubiquity of UDP, we are also investigating integrating the group manager with the underlying routing protocols using a cross-layer design approach. We expect that by integrating the INFO and PING packets with the exchange of routing messages, we can significantly reduce the bandwidth utilization for the group manager. Moreover, by taking advantage of the topology information available at the routing layer, we hope to improve the performance of active propagation of information as well as the peer search operations.

AKNOWLEDGEMENTS

This work is supported in part by the U.S. Army Research Laboratory under Cooperative Agreement W911NF-04-2-0013, by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0009, and by the Air Force Research Laboratory under Cooperative Agreement FA8750-06-2-0064.

REFERENCES

[1] Suri, N., Bradshaw, J., Carvalho, M., Cowin, T., Breedy, M., Groth, P., and Saavedra, R., Agile Computing: Bridging the Gap between Grid Computing and Ad-hoc Peer-to-Peer Resource Sharing. In Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), 2003.

[2] Suri, N., Bradshaw, J., Carvalho, M., Breedy, M., Cowin, T., Saavedra, R., and Kulkarni, S. Applying Agile Computing to Support Efficient and Policy-controlled Sensor Information Feeds in the Army Future Combat Systems Environment. In Proceedings of the Collaborative Technologies Alliance Conference (CTA 2003), 2003.

[3] RFC-Gnutella Team. The Gnutella Protocol Specification. Available online at: http://www.the-gdf.org/.

[4] Ariba Corp., IBM Corp., and Microsoft Corp. UDDI Technical White Paper. Available online at: http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf. September 2000.

[5] Guttman, E. Service Location Protocol: Automatic Discovery of IP Network Services. IEEE Internet Computing. Vol. 3, No. 4, pp 71-80.

[6] Engelstad, P.E. and Zheng, Y. Evaluation of Service Discovery Architectures for Mobile Ad Hoc Networks. In Proceedings of the 2nd Annual Conference on Wireless On-demand Network Systems and Services (WONS 2005).

[7] Helal, S., Desai, N., Verma, V., and Lee, C. Konark – A Service Discovery and Delivery Protocol for Ad-Hoc Networks. In Proceedings of the 3rd IEEE Conference on Wireless Communication Networks (WCNC 2003).

[8] Sailhan, F. and Issarny, V. Scalable Service Discovery for MANET. In Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005).

[9] Chakraborty, D. Joshi, A., Yesha, Y, and Finin, T. Toward Distributed Service Discovery in Pervasive Computing Environments. IEEE Transactions on Mobile Computing. Vol. 5, No. 2, pp 97-112.

[10] Garcia-Macias, J.A. and Torres, D.A. Service Discovery in Mobile Ad-Hoc Networks: Better at the Network Layer? In Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW 2005).