ABSTRACT
The role of Multi-Agent System (MAS) Infrastructure is to support agents while they create more complex virtual communities. As agents dynamically appear and disappear from the environment (or move from one host or network to another) they may find that their infrastructure services have also dynamically become transient, creating potential disruptions to the activity of the MAS. Therefore agents should be provided with tools that allow them to dynamically reconfigure and adjust their operation, based on these changes. We describe the protocols of a multicast-based discovery mechanism, that allows agents to rapidly track the changes in the infrastructure of their local networks in order to maintain a consistently high level of interoperation with other agents. In addition, we show how the discovery protocols can be extended over the Internet through a Peer-to-Peer connectivity model based on the Gnutella architecture to create Agent-to-Agent (A2A) communities. The combination of local and global discovery allows agents to maintain survivable connectivity to both local infrastructure and to services dispersed throughout the Internet. The use of this wide-area discovery service allows the agent systems to ultimately participate in more globally diverse and complex agent communities.

1. INTRODUCTION
The promise of Multi-agent technology is for the automatic creation of virtual communities of heterogeneous agents that dynamically collaborate, compete, form teams and coalitions, enter into auctions and negotiate on compensation for services. Yet, all of this does not happen by magic. It is not enough to implement a set of intelligent agents, start them, and wait for them to spring into action and create a community. Rather, agents need a set of infrastructure services to support their interactions, as well as knowledge and conventions that allow them to understand and communicate with each other. The infrastructure must provide white-page services that allow agents to locate each other, and yellow-page services (or middle agents [1, 12]) that identify which agents provide what services. Examples of middle agents include the OAA Facilitator [2], the RETSINA Matchmaker [6] and the Infosleuth Broker [3]. One often neglected aspect of MAS is how agents find their required infrastructure components. The solution usually adopted, to solve this problem in implemented systems, is to handcraft the location of the infrastructure into configuration files, or to hardcode the information in the program code of the agent. The use of static configuration information allows agents to connect to common infrastructure, but it also creates brittle systems which fail to adapt to any change of the infrastructure. The result is that whenever some infrastructure component fails, or changes, the overall MAS is affected: agents are no longer able to find each other and most likely stop functioning until a programmer updates the configuration files.

In this paper we claim that information about infrastructure should be available dynamically rather than through static configuration files. By gathering information about infrastructure dynamically, agents can quickly and intelligently react to changes in the infrastructure components while they continue with their work.

We propose A2A, a new framework for infrastructure discovery based on Multicast and P2P protocols, that allows agents to locate infrastructure components even in presence of drastic changes. Agents that use A2A are not hindered by the problems of configuration files: they always know what alternative infrastructure components are currently available in the system, so they can make a decision of which infrastructure components to use, and eventually switch when the components fail. In addition, when the desired type of infrastructure component is not currently available in their local network, the agent can look for a substitute on the global network, or just wait until it is notified through the discovery mechanism that the desired service has become available on the local or wide area network.

Upon entering the MAS, agents use A2A to transmit requests for desired infrastructure services; each service that matches the request will reply to make itself known. The requesting agent will then select the services with whom to register its own capabilities and location. Symmetrically, Infrastructure components upon entering the MAS, use A2A to advertise their presence. Any agent on A2A will be notified of the existence of the new component, and may decide whether to make use of it. The result of this schema is that any agent has detailed knowledge of what infrastructure services are available at any time.
A2A is composed of two discovery mechanisms to satisfy the different requirements of local area networks (LAN) and wide area networks (WAN). A2A discovery within the LAN is based on Simple Service Discovery Protocol [5] (SSDP), it uses multicast to locate infrastructure services rapidly and reliably. Still, multicast is very sensitive to local network boundaries and Internet routing infrastructure, and it is very unreliable on the WAN. This is a serious limitation for MAS technologies that aspire to operate across the Internet. On the WAN, A2A takes advantage of the fabric of the peer-to-peer (P2P) network, and augments the P2P protocol to guide and direct peer connectivity to form A2A communities. These augmentations fit within the existing protocol structure and guidelines, and therefore maintain complete interoperability with existing (non-agent) p2p clients. Furthermore, we will show how we’ve enhanced this A2A architecture to maximize the probability for appropriate connectivity and to minimize extraneous traffic.

A2A has been completely implemented and deployed to find infrastructure components, such as yellow and white page services in the RETSINA Multi-Agent System [8], using simple discovery protocols. In the rest of this paper we discuss A2A in great detail. In section 2, we discuss the multicast-based discovery infrastructure utilized in the Local Area Networks. Section 3 augments the A2A paradigm to operate over wide area networks, such as the Internet utilizing the fabric of a peer to peer architecture. In section 4, an analytical evaluation of the different discovery protocols is presented. Section 5 discusses some directions for future work to unify and expand the discovery mechanisms. Finally, we draw our conclusions.

2. LOCAL DISCOVERY

Agents in the RETSINA system use Simple Service Discovery Protocol [5] (SSDP) to find the infrastructure services for their environment. SSDP was created in conjunction with the Universal Plug-and-Play [11] (UPnP) initiative driven by numerous vendors, including Microsoft and Intel. It was developed as a lightweight mechanism to discover service providers in an ad-hoc and dynamic environment. Two modes of this protocol’s operation support the notification of a service’s availability status, and the search for services.

SSDP specifies three types of messages to be multicast. Alive messages are sent out to communicate that a new service has come online. Byebye messages are sent to communicate that the service is leaving the system, and Search Requests are multicast to locate services that are currently available. Responses are only sent to the sender of a Search request, and they are transmitted via unicast. All SSDP message packets are implemented using UDP (User Datagram Protocol) [10] which is a more lightweight protocol than TCP; but because UDP implements fewer checks and balances, it carries no guarantee of delivery. The resulting discovery process is asynchronous where multicast messages are publicly accessible, and unicast responses (if any) may be singular, multiple, or repetitive; and possibly time-delayed.

Upon entering the RETSINA MAS, every agent and infrastructure service multicasts an Alive message and a sequence of search requests for services they need. The Alive message, shown in figure 1, is a communication to all the other agents of the existence of the new agent. The message specifies the type of message sent using the NOTIFY header and the ssdp:Alive indicator, the type of agent that sent out the message (in this case a retsina:Agent); where the agent can be found (<kriton.cimds.ri.cmu.edu:5555>), and finally, the unique name of the agent (SECstock). Alive messages are cached in agents by the appropriate service’s client modules, with location information for future points of contact. Each piece of cached information is only retained for a limited duration, since agents or the interconnecting network may fail or become unavailable. A value of 15 minutes is used in the example below (900 seconds). At 75% of this interval, the agent (if it is still operational) will automatically send a fresh Alive packet to extend its entry’s “lease” in other agents’ discovery caches.

**Figure 1: An Alive message from agent SECstock**

Next, a RETSINA agent will multicast one or more Search Request messages to look for services that it needs. Service requests can specify a specific service, or that all available service providers should respond. An example of an infrastructure service request is shown in figure 2. The agent with a unique name of SECstock, is seeking a service of type retsina:AgentNameServer, and has provided its discovery return address Host: 128.2.179.228:1082 to indicate where the unicast response should be sent. This message is ignored by all agents and services whose type doesn’t match the requested type (i.e. all non-ANS services.)

All ANSs in the MAS recognize that their service is being sought and respond to the request. To avoid network “flooding” problems that might occur if numerous services attempted to simultaneously unicast responses to the same agent, services will stagger their replies by choosing a random delay time with an upper limit as indicated by the MX parameter. In the example, each service will delay between 0 and 2 seconds before transmitting their service response.

**Figure 2: A Search Request from agent SECstock**

Similar search messages are transmitted to look for Matchmakers, Brokers, and other infrastructure services like Loggers, MAS visualization tools, and Multi-Agent Launchers [8]. When an agent or infrastructure service receives an appropriate multicast search request, it will reply directly to the sender via a unicast response. As opposed to the multicast message that is transmitted to all the agents, a unicast response message is a direct point-to-point communication. For example, several ANS servers may reply to an agent request by each providing their own network address so they
may be contacted. No other system on the network will see the responses returned to the original requester; but they all had an opportunity to see the request.

Finally, Bybye messages are multicast when the agent leaves the system. Every agent that had previously learned location information about this service, can then remove the reference from its cache.

Discovery is also used as an event-based trigger to facilitate higher level coordination and community interactions within the MAS infrastructure. For example, an ANS multicasts an Alive message to announce its presence in the system, followed by a search request to locate other ANSs. Existing ANS servers learn of the new ANS from its Alive message, and the new ANS learns of the existing ANS servers from their responses to its query. Once they know about each other, they will exchange information and registration requests, increasing the reliability and efficiency of the overall MAS by ensuring redundancy of agent registration information, and by providing load balancing for each other.

The transmission of Bybye messages from ANS servers who leave the system will prevent the future occurrence of broken connections and socket timeouts during the systems’ intercommunications.

Agents react to Alive messages from infrastructure components by registering with the new services if deemed appropriate, or by ignoring these service discovery events. In the case of the Agent Name Service, an Agent will automatically send a registration request to the new ANS as it comes alive. This action by the Agent will help to make the Agent’s name registration survive system outages, and automatically refresh an environment where a single ANS server crashes and recovers, but has lost its registration database.

2.1 Alternative Discovery protocols

SSDP was created as a lightweight discovery protocol for the Universal Plug-n-Play (UPnP)[11] initiative, and defines a minimal protocol for multicast-based discovery. Because it is so minimal and lightweight, it was advantageous to utilize SSDP in the RETSINA architecture, which supports agents on many platforms, including handheld computers. Nevertheless, we considered other discovery protocols such as SLP and Jini.

Service Location Protocol (SLP)[4] defines three types of systems that can participate in the discovery process, and the interactions between these entities. SLP deals with User Agents (UA), Service Agents (SA), and Directory Agents (DA). SAs have a service that they can provide (a producer.) UAs want to find a SA to perform some task (consumers.) DAs store SA advertisements as a “central clearing house” for finding services (yellow-pages.) The SLP protocol states that a UA should try to “discover” a DA using multicast when coming online. If found, all further discovery will take the form of a lookup process between the DA and the DA using directed unicast conversations. Likewise all SAs should have found the DA, and registered or advertised themselves with it. If a DA is not available, the UA can directly multicast a request for a type of service and any SAs that receive the message and can provide the specified service should respond to the UA.

Java’s Jini is similar to SLP, except that it requires the a priori presence of a DA. The direct discovery of SAs by UAs is not allowed. Otherwise the protocols are similar. In fact, SSDP, SLP and Jini map the current spectrum of permutations available for multicast-based discovery. The minimal case is where there is no discovery. At the next level, SSDP allows any client (UA) to discover any service (SA) without the assistance of any other infrastructure component. Jini has UA and SA entities that discover DA systems which provide a mediated lookup service. Finally, SLP implements the DA to provide a mediated lookup service, but a UA can still discover an SA directly, in the absence of a DA.

Compared with SLP, the RETSINA infrastructure already has the equivalent of a DA service in the form of the Agent Name Service (ANS) [7]. The use of SLP within RETSINA would therefore create redundant services and duplication of functionalities. A similar problem would be created with Jini, with the addition that Jini is typically (but not necessarily) a Java-specific solution, while the RETSINA MAS infrastructure supports heterogeneous agents running on a number of platforms and implemented in a number of programming languages (e.g., Java, C, C++, Lisp, etc.)

2.2 Limits of Multicast

Multicast performs a controlled broadcast on the network that prevents traffic from being propagated onto segments of the network where the packets are not required or desired. In addition, multicast discovery packets are limited in their transmission by a Time-To-Live (TTL) value that indicates the number of routing devices over which a packet should be allowed to propagate over. However, not all routers or firewalls are able (or administratively configured) to route multicast packets, which prevents discovery protocols from being used pervasively and globally.

Since the spread of multicast messages is controlled at the level of packet routing in the network, it is very sensitive to the topology of the overall network. The implementation of multicast results in difficult tradeoffs between spreading the message to all the expected recipients and controlling the message so that it does not needlessly consume excessive bandwidth. The result is that, typically, Multicast cannot be reliably used outside the LAN of the sender. Our dilemma is that if we want to extend Multi-Agent systems across the whole Internet, the exclusive use of multicast for discovery imposes a serious limitation; and using static configuration of remote infrastructure addresses returns the MAS to a brittle and vulnerable state.

3. GLOBAL DISCOVERY

Multi-Agent Systems exemplify the paradigm of an Internet of services in which sites (agents) interoperate and exchange services rather than just static pages. In order for MAS to expand to the whole Internet, the infrastructure of MAS should be available in the same venue. We envision an Internet-wide MAS to be served by distributed registries providing white and yellow page information services. While each registry may directly serve agents in a private community, they may also form an integrated network of services that exchange information to support the communication of agents across the Internet. For example, if a white page service, such as the RETSINA ANS, cannot resolve a request for an agent in its own LAN, it can ask peer ANS servers across the Internet to resolve the reference. In turn, ANSs forward the request to other ANSs that they know, until either the agent is found or the request has traveled too far or timed-out.

To extend the MAS infrastructure to Wide Area Net-
works, discovery should also be extended to the Internet so that infrastructure services can find each other and collaborate. In addition, agents should be allowed to look for infrastructure services across the entire Internet when they cannot find the service in their local network. In this section we describe the Agent-to-Agent (A2A) global discovery and lookup infrastructure based on the Gnutella P2P Network. A2A discovery and lookup allows agents and infrastructure components to dynamically expand their search across network boundaries, to find ANSs, Matchmakers, and other Middle Agents. A2A makes it possible for communities of agents to be automatically constructed, even though the group members are separated across wide expanses of the Internet.

3.1 The Gnutella P2P Network

Gnutella is both a file sharing mechanism and an asynchronous message passing system that allows its users to locate and share (typically MP3 music) files across the Internet. Each Gnutella application is both a “SERVer” and a “ciENT”, so the protocol’s creators coined a new term to describe a participant of the Gnutella Network as a “servent”. Other than the rapid and global deployment of this application, the exciting part of Gnutella is that it functions purely on Peer-to-Peer (P2P) connections between the servents without a centralized server.

Gnutella servents use their message passing system to perform two types of operations. First, they exchange messages to other servents that are available on the network so that they can maintain, or increase their level of connectivity to the overall Gnutella Network. Secondly, they exchange messages to search for specific files that might be available from other servents. This messaging system is primarily composed of binary packets of information, and text strings that represent search requests. File exchange is based on the HTTP protocol, and uses the same mechanisms used in the retrieval of content from web servers.

Gnutella Network discovery, used by traditional servent applications, is “boot-strapped” by connecting to a few well-known servents that the human operator running the application expects to be there. Several servents have been set up on the Internet to cache connection information of available servent systems, so this initial phase is now typically automated. Agent applications can start with this model, and augment it with a priori information from previous interactions, and by utilizing local discovery as described above.

As each servent starts, it will connect to a few remote systems. In order to discover other servents in the Gnutella Network, it starts a PING/PONG process. PING messages are sent in hopes of receiving PONG messages that contain host, port, number of files, and kilobytes shared from other servents on the Gnutella Network. As shown in Figure 3, each servent that receives a PING performs two operations: first it sends a PONG back along the same path from which the message came, so that eventually the PONG will reach the originating servent; second it forwards the PING to other servents with a reduced TTL. As soon as the TTL reduces to 0, the message is no longer forwarded and the search ceases to propagate. Because of the high degree of connectivity between servents on the Gnutella Network, a PING may hit up to an exponential quantity of servents in its travels from servent to servent. As an example depicted in Figure 4, if all servents were maintaining 5 active connections, and had a TTL of 7, a PING message could reach as many as 109,225 other servents on the network.

Also, as time goes on, a servent will be pinged by other servents as they connect to the Gnutella Network; and the servent will “overhear” PONG messages from even more servents that it is routing. Many of the PONG messages that a servent routes will be from systems that were not online when the servent first sent its own PING message. These additional PONG messages can be harvested, and used to curtail the over-utilization of PING messages. The messaging mechanism of Gnutella is based on the voluntary action of every servent to actually route the messages that they receive to other servents. The routes constructed will have more of a resemblance to graphs than to trees, so even if one servent does not route messages, there is a high likelihood that the original message will still be transmitted via an alternate route.

The search mechanism of Gnutella uses the same message passing process utilized to PING other servents except that message routing occurs first, before a servent attempts to perform a more time-consuming local search. A QUERY message that is sent to the Gnutella Network contains a number representing the minimum acceptable communications link speed for file downloads, and a string representing the content that is being sought. In typical Gnutella ser-
vents, the search string will be tokenized before a servant’s local file system searches for filenames that match any of the string’s keyword tokens. If a local file exists that matches one or more of the words in the query string, its information will be formed into a response to the QUERY packet. If more than one file matches a pattern, the servant can reply with multiple responses embedded in the same message. The QUERY–HIT message that is sent back contains information about the system’s link speed and the name and size of each matching file. It also contains an integer index value to help map the request into the local file system’s storage.

### 3.2 The A2A Network LookUp and Discovery

Gnutella provides a basic connectivity schema that allows its servants to discover other servants over wide area networks. By enabling Agents and infrastructure components of the RETSINA MAS to act as servants on the Gnutella Network, we can take advantage of a fabric of wide-area connectivity that is already in existence and widely deployed. The result is that whenever an agent needs to locate an infrastructure component, such as an ANN server, it can send both a multicast discovery message to the local network, and a QUERY request to the Gnutella Network. Since the speeds of the two processes are quite different (Gnutella being slower than multicast) the agent will connect first to the local service but still learn of other remote services as they become visible.

As QUERY requests fan-out over the Gnutella Network, being sent from servant to servant (like any other MP3 music file search request,) A2A servants providing RETSINA infrastructure functionality will recognize a request for a service that they provide, and reply with a QUERY–HIT message. However, these QUERY–HIT responses do not provide information about where to find a file, but the address where infrastructure component can be reached.

To create A2A servants, we augmented the standard Gnutella servants in several ways. A2A servants monitor their connections to develop confidence levels for remote servants, and use these values to manage their connectivity. They categorize remote servants based on numerous attributes, to increase connectivity to communities that are most beneficial. A2A servants quickly focus incoming queries to agent tasks that can respond, and bypass inappropriate requests. And finally, they implement the automatic answering of requests that can facilitate the congregation of agents into communities. In the following sections we explain further details about the confidence, categorization, search, and discovery mechanism within A2A servants.

#### 3.2.1 Confidence

The A2A architecture implements a ranking operation for connections by using several counters to indicate the confidence that an agent has in the usefulness of a connection. The counters provide a metric to indicate a positive or negative confidence in the connection’s worthiness to remain connected, or be reestablished at a future time. The counters track repetitive messages, consecutive messages of the same type, repeated queries, and malformed or unknown message types.

The ranking of servants is used by an agent to help decide which servants to remain connected to. A2A servants do not keep their connections open forever: a connection is open only for a set number of messages. As an agent interacts with other servants, it learns from their interactions and modifies its confidence level. The more information the agent gets from another servant the better it ranks it and the longer it keeps a connection open to it. On the other hand, if the information provided by the servant is of no interest, or the agent suspects that the other servant is being disruptive, it will use the decreasing confidence level close the connection prematurely. The ranking is also used to decide with which servants to connect next, so that the agent is more likely to open a connection to a servant with whom it has a high confidence. The result of the ranking is that the agent is more likely to be connected to other agent/agents that provide useful information.

#### 3.2.2 Categorization

Gnutella accumulates all connectivity candidates into one large pool. The RETSINA A2A software refines these selections and categorizes hosts into several prioritized groups.

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIME</td>
<td>preferred, agent specified</td>
</tr>
<tr>
<td>ALT</td>
<td>high calculated confidence levels</td>
</tr>
<tr>
<td>LOCAL</td>
<td>nearby on the agent’s current network</td>
</tr>
<tr>
<td>HOME</td>
<td>on agent’s home network or system</td>
</tr>
<tr>
<td>OTHER</td>
<td>miscellaneous unclassified servants</td>
</tr>
<tr>
<td>CACHE</td>
<td>system bootstrap cached PONG servers</td>
</tr>
<tr>
<td>BAD</td>
<td>servants or networks to be avoided</td>
</tr>
<tr>
<td>NEW</td>
<td>staging area for discovered servants</td>
</tr>
<tr>
<td>USED</td>
<td>servants active on current connections</td>
</tr>
<tr>
<td>COMMUNITY</td>
<td>focused on a specific Task or Service</td>
</tr>
</tbody>
</table>

Table 1: A2A Servant Categories

The agent tries to keep a set number of each category of servants connected. The total connectivity count must also fit within the minimum and maximum allowed connections settings. If there are insufficient “appropriate” connections in place to perform a lookup or other agent communications task, an ordered selection from these categories will be utilized to increase connectivity. If the total number of connections exceeds a maximum limit, the A2A Agent will go into a host-cache mode, where no new outgoing connections will be attempted, existing connections will eventually expire and be disconnected, and incoming connections will be limited to a very few interactions (PING/PONG and immediately disconnect.) The PONGs that the host-cache mode A2A Agent will respond with, are the PONG packets of the servants it is currently attached to. This way, if Agent A knew it needed to talk to Agent B, but Agent B could not support any additional connections, Agent A could be connected to Agent C who was already participating in a conversation with Agent B. Any QUERY's sent to Agent C would fan-out and reach Agent B. Scalability is enhanced while maintaining required connectivity and intercommunications capabilities.

#### 3.2.3 Community-based A2A Search

The Community structure of A2A enhances the basic QUERY process provided by Gnutella. All servants in Gnutella are assumed to be working on the same type of process: file sharing. When answering a query, they provide information that facilitates the location and downloading of a file. MAS, however, can utilize this same network transport to
locate service providers (with a response using a URL to describe how and where a service could be contacted), or to actually have work done (with the response being the results of the process performed.) An impediment to using the public Gnutella Network is that the agent systems should be able to ignore non-Agent messages, or any query that relates to a task that they cannot perform. Agent communities optimize their communications over the Gnutella Network since they rapidly and precisely determine which of the many QUERY messages flowing through the network are appropriate to work with. The A2A architecture does this by explicitly categorizing conversations, and providing a short-cut to expedite the sifting of messages.

To become part of a community with similar interests, an agent will initiate a new Task (referred to as a “Community”) to which it can pose “Questions” and receive “Answers”. Joining a Community requires that the Agent specify a textual description of the task-type. For example, a Community might be created to participate in a group like retsina:AgentNameServer, retsina:matchmaker, auction:automobile, store:shoes, or weather. This string will be encoded into a numeric value that is included with the outgoing question as the Gnutella “minimum connection speed” value. As questions arrive at an A2A servant, each of the agent’s Community tasks can quickly examine this value to see if the new Question’s special code matches the topic that it services. If it doesn’t match, the QUERY message might be from a normal Gnutella servant searching for “Britney Spears” or another agent seeking some other information. This is a “quick-check”, and is further validated by examining the actual query from the remote Agent.

As a question is posed to a Community object, the question’s text will be automatically prefixed with the text that identifies the Community’s actual type. For example, a query moving across the network, might appear as: stock:quote:IBM or retsina:AgentNameServer:lookup:brent. After passing the quick-check, described above, a receiving agent will verify that the query actually begins with the Community task’s identification string (e.g. stock: or retsina:AgentNameServer:). Once it verifies that the incoming request was directed to this task, the Community object will process and respond to the query with an appropriate answer. Otherwise the query is ignored. The encoded Community task identifier always represents a value that would be interpreted as a speed greater than 10Mbps for the minimum download bandwidth, so normal Gnutella servants should ignore the A2A QUERY. To increase privacy and further reduce the probability of false hits being generated by non-agent servants, the questions and answers (QUERYs and QHITs) can have their payload automatically encrypted and decrypted.

3.3 A2A Discovery

Many questions will take the form of “who can do service X” or “where is Agent Y”. This is especially true of “Look Up questions”. These types of questions will have responses that are composed of host/port information, so the originator of the query can then contact that system. Since the type of query is so common, the A2A Community object can support Agent-specified auto-answer questions. When these Questions arrive at the Agent, the underlying A2A architecture will see the common Question and immediately reply with the predefined Answer, without disturbing the Agent to process the request. An Auto-Answer Question that is initiated whenever a new Community object is created, is for discovery (e.g. retsina:AgentNameServer:discover). The answer to this question would be the hostname and port where this service is listening for incoming connections.

When a new Community object begins operation, it will send one of these discovery questions to the currently established Gnutella P2P Network. Any responses that come back are added to the Community’s list of discovered hosts. This list is similar to the Prime, Alt, Local, and Home category lists mentioned previously. However, every time an Agent attempts to ask a Question to the network, the Community object will endeavor to ensure that a minimum number of Community/Task-specialized connections are active before actually transmitting the question. The Community Object will periodically retransmit the discovery query in an attempt to maintain a preset number of Community-specific hosts. Over time, connectivity of an A2A agent will automatically shift from the public Gnutella Network, to communities of similarly focused agent populations.

3.4 Scaling

A distinction between SSDP (or multicast-based discovery protocols in general) and this Agent-to-Agent model, is that with SSDP, systems that provide a specific service respond to the multicast discovery messages. P2P and A2A blur the differentiation between systems that are exclusively client-like or server-like in nature, and treats everyone as peers. The systems discovered in any P2P network might be clients or servers of that service, or both. If required, an A2A Community Task can be further augmented with an auto-answer indicating that it provides a specific service and clients of that type of service can ask a question explicitly seeking servers of that specific type. Since, typically, there will be many more clients in existence than there are services (and services may have to limit their number of concurrent connections) discovery of another system that is either a specific service provider, or (more likely) connected to a provider of that service, allows that agent to connect in a more opportunistic way. Scalable services that operate on the Internet will need to maintain strong connectivity to communities of clients, and not just to massive amounts of directly-connected individual clients.

The use of a form of managed broadcast, by utilizing point-to-point fan out of messages to reach a multitude of systems, can help Agent infrastructure communications scale to larger levels of connectivity. Rather than maintaining individual connections to many different resources, an Agent can keep a few connections active into the P2P network fabric. A2A communications riding on top of this network can moderate the connections to increase the probability that a system of the desired type is within the TTL of any Agent’s transmissions. Then, active communities of Agents working with similar interests will congregate together to allow the formation of robust interconnectivity behaviors.

Transmitting one request that reaches tens of thousands of systems can increase the reach of an Agent to global proportions. Tuning the selection of peers can dramatically improve the appropriateness of message transmission. Customizations via configuration parameters can allow limited capability Agents to choose whether to ignore (just relay), drop, or process specific types of Gnutella, or A2A messages. “Harvesting” of information seen passing a system as it routes messages can proactively discover information.
needed in advance of actually requiring a lookup message being sent. The added ability to harvest ALL Questions (regardless of there being a matching Community object currently active at the Agent) allows an Agent to monitor and fully interact with all messages flowing over the Gnutella Network. This feature could be used to monitor or categorize traffic loading, anticipate message storms, and automatically adjust connectivity to avoid denial of service attacks or system misuse. Private conversation groups could also be created by restricting the PING/PONG behavior from specific communications links and only allowing out-bound queries and responses to pass.

4. QUALITATIVE EVALUATION

In this paper we have been advocating the need for a discovery mechanism as part of the MAS infrastructure. Here we make a qualitative evaluation of the various discovery mechanisms that we proposed in regards to: what is gained with their use; when they are adequate; and when, instead, they are inadequate. In the evaluation we will compare the base condition no discovery against the two discovery mechanisms described: multicast and A2A.

We will compare the three environments on 4 dimensions: soundness, completeness, adaptability, and speed of discovery. The summary of our analysis is displayed in table 2.

- **Soundness** measures whether the agent finds what it seeks. All three methods are sound. The base condition no discovery is trivially sound because the system administrator points the agents directly to the service that they need; the two discovery mechanisms only allow a reply from infrastructure that matches a specified type.

- **Completeness** measures whether the agents will find all services that they are looking for. The base condition no discovery is trivially complete in the sense that the system administrator decides which agents should connect with which infrastructure; so there is no need for the agent to develop a picture of what services are available. The two discovery mechanisms are not complete. Multicast’s UDP packets can be dropped or lost, and its discovery is typically limited to LAN segments. A2A has a broader discovery range, but does not guarantee a totally non-segmented network and so some systems may be unaccounted for.

- **Adaptability** measures whether the agents can adapt to changes in dynamic infrastructure landscapes. No discovery is not adaptable because the agents have hard-coded references to the infrastructure components, which is the problem we were forced to address when we looked at discovery. Both multicast and A2A detect changes and are able to recover from loss of connectivity to infrastructure devices. (Again, multicast discovery is restricted to a localized operation.)

- **Speed of Discovery** measures the speed at which agents discover the infrastructure services they seek. In the no discovery condition the information is already provided, so there is no point with measuring speed. Multicast is extremely fast and efficient; services are hit in a few milliseconds and any delay of their discovery process is due to the built-in packet-flood-prevention mechanisms of the protocol. A2A is much slower since requiring the message and response to be propagated across the network by the servants.

This analysis identifies some of the tradeoffs behind the different configurations. No discovery is useful only when the infrastructure is carefully pre-planned and never modified or perturbed in any way. In such cases, the System Administrator can confidently hard-code the references to infrastructure components and decide, in advance, where the agents are going to connect. The analysis also shows that multicast is very efficient and it can perform almost as efficiently as hard-coding, with the additional benefit of allowing automatic reconfiguration of the agents. The limit of multicast is scalability to the whole network; which is solved by the A2A discovery that has the potential to scale to the Internet. However, it suffers from increased discovery latency. Within the Retsina project we implemented discovery in such a way that we use multicast as the primary discovery mechanism, and supplement it with A2A discovery for Wide Area implementations.

The last column of the table reports how other distributed systems are placed with respect to discovery. OAA and Infosleuth did not report of any discovery mechanism in their papers; UDDI[9] is supported by a directory service run by IBM and it does not report of any automatic discovery of lookup services. Jini supports discovery of lookup services (LUS), but it does not support discovery to interconnect distributed LUS. UPnP supports multicast to locate services within the local area network using the same SSDP protocol used by Retsina; finally Gnutella based systems do not distinguish between local and wide area networks and support discovery only through the Gnutella PING / PONG protocol described above.

Table 2: The results derived from the query placed to the agent

<table>
<thead>
<tr>
<th>Discovery Method</th>
<th>Sound</th>
<th>Complete</th>
<th>Adaptable</th>
<th>Speed of Discovery</th>
<th>Where to Use</th>
<th>Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Discovery</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>instantaneous</td>
<td>static infrastructure preplanned</td>
<td>OAA, Infosleuth, UDDI, Jini</td>
</tr>
<tr>
<td>Multicast (LAN)</td>
<td>yes</td>
<td>high probability</td>
<td>yes</td>
<td>high</td>
<td>ad-hoc local workgroup</td>
<td>UPnP, Jini, RETSINA</td>
</tr>
<tr>
<td>Agent-to-Agent (LAN &amp; WAN)</td>
<td>yes</td>
<td>high probability</td>
<td>yes</td>
<td>low</td>
<td>ad-hoc global communities</td>
<td>Gnutella, RETSINA</td>
</tr>
</tbody>
</table>

5. FUTURE WORK

5.1 Support for Constrained Platforms

Some Agent platforms with extremely limited resources (cell phones, embedded devices, and palm computers) might
be unable to maintain even 5 concurrent open socket connections. By allowing an outgoing connection of the P2P layer to be a localized (low TTL) multicast connection, these small devices would be capable of concurrently communicating to multiple local systems as if each system had a separate connection. Using this vehicle to connect to larger platforms, each with multiple open connections to other Gnutella servants or A2A Agents, the resource-constrained system could easily participate in the larger expanse of network connectivity.

5.2 Wide and Local Discovery Integration
Using the multicast connectivity described above, the normal discovery operation of Gnutella PING / PONG transactions, together with the higher level discovery of the A2A layer, alleviates the need for a separate local discovery protocol. Tuning the process to better discriminate which packets it routes to this multicast connection would help to maintain the traffic to levels easily supported by agents and networks with connectivity and bandwidth constraints.

5.3 Infrastructure Integration
Beyond the use of basic infrastructure discovery and lookup services, new areas of research can be examined with the A2A enhanced conversations and interactions between Agents. Now an advertisement to a Matchmaker could be overheard, and waiting clients could immediately respond. Also lookups could solicit responses from new (or hybrid) systems that could provide augmented or enhanced services; and provide services that the Agent might not have known to ask for. Learning-systems that observe solutions portrayed by interactions between other Agents could form their own infobase of heuristics that have been seen to work by other systems.

5.4 Standard and Upgraded Technologies
The discovery techniques implemented in the RETSINA agent infrastructure solved many real-world problems that arose while performing software demonstrations, presentations, and integration experiments. Numerous groups are developing similar techniques and strategies for facilitating continued system operation in dynamic environments. The IETF’s ZeroConf and DNS-ext working groups have been pursuing this lofty goal for quite a while. Once formalized techniques become available, and are integrated into the underlying TCP/IP mechanisms, the A2A “wedge” software can be replaced by mainstream tools. Until then, we can continue to develop “intelligent” agent software that is build upon a facility that accommodates the dynamic nature of actual working environments.

Also, Gnutella has continued to thrive as a community, and has seen many wonderful collaborations for extending the basic protocols. Many actions that the A2A code provided to our agents are now available to all Gnutella clients. Standard mechanisms to include XML (or other) meta information have also been added. We feel that upgrading the A2A code to take advantage of the newest features of the expanding Gnutella protocol, and possibly to replace some of the low-level protocol handling with the recently-available open-source Gnutella libraries would provide enhanced capabilities and a shorter lead-time when applying future updates to the protocols.

We are not advocating that SSDP or Gnutella are the end-all mechanisms to achieve the desired results described in this paper; and it is not our role to develop the ultimate products in these areas and to see that they are widely deployed. This will happen with or without us. The specific technologies chosen had features and capabilities that fit well with our existing architecture and research goals, and were sufficiently well defined (and available) so that we could integrate them with our systems. What we did do feel to be important for our group, and all Agent Research, is that we understand the impact that these technologies can have on our infrastructure and how we can use them to expand the depth of our research. Our Agent-to-Agent research can provide valuable real-world feedback to the developers of these new and future protocol standards. We also feel that A2A technologies dramatically increase the ability of Intelligent Agent Systems to dynamically interact with each other and their “environment”; and they will produce new operational and behavioral models for agent interaction.

7. ACKNOWLEDGMENTS
This research has been sponsored in part by the Office of Naval Research Grant N-00014-96-1-1222 and by DARPA grant F-30602-98-2-0138.

8. REFERENCES

6. CONCLUSIONS