On the Discovery of Brokers in Distributed Messaging Infrastructures

Shrideep Pallickara, Harshawardhan Gadgil and Geoffrey Fox

{spallick, hgadgil, gcf}@indiana.edu

Community Grids Laboratory, Indiana University

Abstract

Increasingly messaging infrastructures are being used to support the communication requirements of a wide variety of clients, services, and proxies thereto. Typically, for various reason this messaging infrastructure is a distributed one with multiple constituent brokers. In the paper we present our scheme for the discovery of brokers in distributed messaging infrastructures based on the publish/subscribe paradigm. We also include empirical results from our experiments related to the implementation of our scheme.

1 Introduction

Increasingly messaging infrastructures are being used to support the communication requirements of a wide variety of clients, services, and proxies thereto. Typically, the messaging infrastructure is a distributed one with multiple constituent *broker*s, where we avoid the term *server*s to distinguish them clearly from application servers. While a distributed messaging infrastructure addresses issues such as scaling and availability, while eliminating single point of failures it is also important for an entity to be able to discover the broker which maximizes its ability to leverage services provided by the broker network. Simple solutions which rely on an entity accessing a certain known remote broker, can sometimes lead to bandwidth degradations and poor utilizations of newly added brokers. In the paper we present our scheme for the discovery of brokers in distributed messaging infrastructures based on the publish/subscribe paradigm.

The publish/subscribe paradigm is a powerful one and one in which there is a clear decoupling of the message producer and consumer roles that interacting entities/services might have. The routing of messages from the publisher to the subscriber is within the purview of the message oriented middleware (MOM), which is responsible for routing the right content from the producer to the right consumers. In publish/subscribe systems a subscriber registers its interest in events by subscribing to topics. In its simplest form these topics are typically "/" separated Strings, these have sometimes also been referred to as *subjects*. When a publisher issues events on a specific topic the middleware substrate routes the events to the subscribers that have registered an interest in this topic.

In this paper we also include empirical results from our experiments related to the implementation of our scheme. We have performed these experiments in the context of our system NaradaBrokering [1-7] which is a distributed messaging infrastructure based on the publish/subscribe paradigm. The scheme presented here can also be leveraged by other systems that are based on the publish/subscribe paradigm.

NaradaBrokering [1-7] is a distributed messaging infrastructure and provides two closely related capabilities. First, it provides a message oriented middleware (MoM) [1,2] which facilitates communications between entities (which includes clients, resources, services and proxies thereto) through the exchange of messages. Second, it provides a notification framework [4] by efficiently routing messages from the originators to only the registered consumers of the message in question. Communication within NaradaBrokering is asynchronous and the substrate places no constraints either on the size, rate or scope of the interactions encapsulated within events or the number of entities present in the system. Events encapsulate expressive power at multiple levels (transport, protocol, service and application). Where, when and how these events reveal their expressive power is what constitutes information flow: the NaradaBrokering substrate manages this information flow. NaradaBrokering includes services such as reliable delivery [5], replays, (de)compression of large payloads, fragmentation and coalescing of large datasets [6], support for the timestamps based on the Network Time Protocol [7] and support for Grid/Web Services [2, 3]. Additional information regarding these capabilities is available at the project web site (http://www.naradabrokering.org) or in the works that have been referenced.

1.1 Problem Statement

A broker network comprises of a large number of cooperating brokers facilitating and mediating access to hosted services and functionalities. Once connected to a broker an entity has access to a wide variety of services. Similarly, an entity may wish to add a broker to this network. In both these cases it is essential for the entity to *discover* a broker. The problem of discovering brokers has two key components. First, the discovery process must return a valid and non-empty set of brokers. Second, the brokers contained within this subset should maximize the added client or broker node's ability to access and mediate accesses to services hosted on the broker network.

1.2 Why a solution is needed

The brokering environment, that we see NaradaBrokering operate in, is a very dynamic and fluid system where broker processes may join and leave the broker network at arbitrary times and intervals. It is thus not possible for any entity to assume that a given broker may be available indefinitely or at all times. Static solutions to this problem might result in a certain known remote broker being accessed over and over again. This in turn causes degradations due to poor bandwidth utilizations. Brokers added to assuage availability and scaling constraints may not be utilized efficiently within this framework.

1.3 The characteristics that should be part of the solution

The discovery of broker thus needs to be a dynamic process which operates on the current state of the broker network. This would ensure that newly added brokers within the system are assimilated faster since the discovery process would operate on them. The discovery process itself might be subjected to the failures and other faults that exist within the brokering system. The discovery process should perform its function in such environments. The broker discovery process should result in a better utilization of the network and networked resources. This is predicated on the fact that the broker returned is the "nearest" available broker, where "nearest" corresponds to network distances or latencies.

The remainder of this paper is organized as follows. In section 2 we describe specialized nodes — broker discovery node (BDN) — that are incorporated into the substrate to facilitate discovery. We then discuss the process of issuing discovery requests (section 3), the processing of these discovery requests (section 4) and subsequently the generation of responses (section 5). In section 6 we then describe the process of broker selection based on the discovery responses. Section 7 outlines the fault-tolerant aspects of this approach and security considerations within the system. Section 8 presents the advantages of our approach. We present the experimental results in Section 9 and the related work in Section 10. Section 11 is the conclusion.

2 Broker Discovery Nodes (BDNs)

Broker Discovery Nodes are registered nodes that facilitate the discovery of brokers within the broker network. BDNs maintain information regarding broker nodes within the system. Here we note that in this scheme not all brokers need to register their information with the BDN. Our section on experimental results includes scenarios where some of the brokers maintain information with the BDN while some do not. In our scheme we have multiple broker discovery nodes (BDNs)(gridservicelocator.org, gridservicelocator.com etc.). We do not place any constraints on the number of BDNs within the system. However, in most cases the number of BDNs will be significantly smaller than the number of available brokers within the network. A given BDN may maintain active connections to one or more broker nodes within the broker network.

2.1 Brokers and Broker Advertisements

Brokers – when they are configured within the broker network – advertise and register their presence with one or more of these BDNs. In our scheme it is not necessary for every broker to be registered with at least one BDN, nor is it necessary for multiple BDNs to maintain identical information about brokers within the broker network. Our scheme will work even if a single broker is registered with a given BDN.

2.2 The anatomy of a Broker Advertisement

When a broker advertises itself with the BDN the advertisement contains information regarding the hostname, transport protocols supported and communication ports, NB logical address and, if provided, geographical and institutional information.

2.3 Dissemination of Broker Advertisements

Broker Advertisements could be disseminated in two forms. First, by sending this advertisement directly to the BDNs that are listed in the broker's configuration file. Second, the broker may send this information to a topic that other BDNs might be subscribed to. Or the broker might send this advertisement over a public topic such as Services/BrokerDiscoveryNodes/BrokerAdvertisement which all BDNs within the substrate subscribe to. Upon receipt of an advertisement at the BDN, this BDN may choose to store the advertisement or ignore it if the BDN is interested in specific advertisements: for e.g. a BDN in the US may be interested only in broker additions in North America.

2.4 Adding private BDNs within the broker network

A need may sometimes arise to setup private BDNs within the broker network. In this case the private BDN must advertise its services to brokers within the broker network. Individual brokers may have the option to re-advertise their information at this newly added BDN. Of course different BDN implementations may have different policies regarding the management of broker advertisements and the responses to discovery requests. A private BDN must also require the presentation of appropriate credentials before it decides whether it will disseminate the broker discovery request.

3 Initiating Discovery

When a new node arrives into the system it issues a broker discovery request. The broker discovery request signifies that the entity is interested in connecting to the nearest available broker. The broker discovery request includes information regarding the requesting node process such as hostname, ports and transports protocols for facilitating communications and sometimes also includes credentials for authorized accesses. A broker discovery request also contains a UUID which uniquely identifies the request.

The simplest way to issue a broker discovery request is to forward this request to a publicly known BDN. A node' configuration file contains information regarding a set of BDNs that can manage its broker discovery request. This includes BDN's running at locations such as gridservicelocator.org (.com, .net and .info). A client can add information regarding any other privately run BDN within its configuration file too. It should be noted that our scheme for broker discovery can work even if there are no BDNs up and running. This would be discussed in a subsequent section. The broker discovery request is generally issued to only BDN.

The transport protocol for communications with the BDN could be reliable connection oriented protocol such as TCP or it could be based on UDP. A BDN is expected to acknowledge the receipt of a discovery request in a timely manner. Multiple requests forwarded to the same BDN would be idempotent.

4 Processing Broker Discovery Requests

As mentioned earlier a BDN maintains active concurrent connections to one or more brokers within the broker network. Upon receipt of a receipt of a broker discovery request from the requesting node, the BDN will propagate this broker discovery request through the broker network. To facilitate faster disseminations within the broker network the BDN may inject this request from multiple broker nodes within the broker network. An efficient scheme to do so may be based on the network distances, which separates the BDN from the brokers, within the broker network. This information could easily be constructed by issuing ping request to brokers and computing the delays from the issued responses. Once this information is available, the broker discovery request would be issued simultaneously to the brokers that are closest and farthest from the BDN. This would ensure that the broker discovery request propagates faster through the broker network.

Every broker discovery request has a UUID associated with it. Every broker keeps track of the last 1000 (this number can be configured through the broker configuration file) broker discovery requests so that additional CPU/network cycles are not expended on previously processed requests.

5 Broker Discovery Response

Timestamps in NaradaBrokering are based on the Network Time Protocol (NTP) which ensures that every node in NaradaBrokering is within 1-20 msecs of each other. NTP services at nodes are initialized during node initializations and generally take between 3-5 seconds before the local clock offsets are computed by the NTP service initialization.

It should be noted that not every broker within the broker network needs to respond to broker discovery requests. A broker's response policy may predicate responses based on the presentation of appropriate credentials. Furthermore the policy may also dictate that responses be issued only if the request originated from within a set of pre-defined network realms.

5.1 Constructing a discovery response

Once a broker decides to respond to a broker discovery request it constructs a broker discovery response. The broker discovery response includes the UUID associated with the discovery request. The discovery response also includes three other key pieces of information

(a) The current timestamp – This is based on the NTP protocol

(b) The broker process information – This includes information regarding the hostname/IP-address of the responding broker, the communication protocols supported and port information associated with each of these supported protocols.

(c) Usage metric information – This indicates the load currently at the broker. This includes information regarding the total number of active concurrent connections to the broker, the CPU and memory utilizations at the broker.

The timestamp information is used to give an idea regarding the network delays that would be associated with communications between the broker and the requesting node. The broker process information facilitates communications with the broker. Finally, the usage metric facilitates selection based on usage and dynamic real time load balancing.

5.2 Issuing the discovery response

The responding broker gleans information regarding the requesting node and issues the constructed response to the requesting node. The communication protocol used for transporting this response is UDP. The reasons for this are two fold

a) Since it is based on a connectionless protocol the network resources utilized by the requesting node remain low and invariant irrespective of the number of responding brokers. If a connection oriented protocol such a TCP were used for communications a local socket (conceivably managed in a separate thread) would need to be setup for each response. Since these sockets wouldn't live beyond the issued response there would also be a garbage collection cost associated with disposing these resources.

b) Since UDP packets can be lost, the response's arrival or the lack thereof provides a good indicator of the underlying response. If the responses were to traverse over multiple router hops the chances that the packets would be lost would be higher. This would ensure that requesting node wouldn't be aware of such remote brokers which would be a prevent it from initiating a connection to the responding remote broker.

6 Processing the discovery response

A requesting node will receive responses from one or more brokers within the broker network. From this set of broker discovery responses the requesting node creates a subset of target brokers, from which it would select the broker to connect to. We first describe the process of creating the target set and then the process of arriving at the target broker.

As mentioned earlier, every node in NaradaBrokering computes offsets to its underlying clock to arrive at the UTC time based on the NTP protocol. Since every response received at the broker also includes the UTC time at which it was issued, we can have a very good estimate of the network latencies to the responding brokers by subtracting the current UTC time from the UTC time contained in the discovery response. This is performed over all responses received within a certain period of time typically 4-5 seconds (this can be configured depending on the accuracy that we seek to achieve).

Based on this information we can now sort the discovery responses according to the delays associated with their responses. The discovery responses also contain usage metrics within them. Based on the computed delays and usage metrics we arrive at our targeted set of brokers. This targeted set of broker typically comprises of around 10 brokers. For the discovery process to complete we need to arrive at the target broker.

Note that since the UTC time would be within 1-20 msecs the network delays that we computed where a very good estimate. In order to arrive at the target broker we need to know the precise network delay. To compute this we send ping requests to individual brokers (based on the information supplied in the broker responses). This ping request contains the timestamp at the requesting node at the instant the ping request is sent. The delay in the responses can easily be computed by the subtracting the timestamp contained in the ping response. The ping requests and responses will also be sent using UDP for the reasons outlined earlier in the discovery response section. The requesting node decides on the target node based on the lowest delay associated with the ping requests.

7 Fault tolerant aspects of this approach and Security Considerations

The approach we have described needs only 1 functioning BDN to work. In fact the approach could work even if none of the BDNs within the system are functioning. This can be achieved by sending the discovery request using multicast. Even if one of the brokers within the broker network is accessible, from the requesting node, using multicast the discovery request would be propagated through the system.

Every node keeps track of the its last target set of brokers. If the requesting is arriving after a prolonged disconnect, and if none of the BDNs are available, the requesting node can issue a broker request to one or more of the nodes in the target set and then process discovery responses as described in the earlier section to arrive at the best available broker. The scheme outlined sustains loss of both the discovery requests (retransmission after predefined period of inactivity) and discovery responses. Similarly, broker advertisements may also be lost in transit to the BDNs.

In our scheme a broker can setup its response policy. This includes requiring that a client present the right credentials in its requests to responding only to those requests that originate within specific network realms. It may be argued if the knowledge of brokers within the broker network could facilitate a denial of service attack. However, the security of any system cannot be predicated on keeping the information about broker processes secret. Means to thwarting denial of service attacks are discussed in a companion paper.

8 Advantages of this approach

There are several advantages to the scheme that we have outlined in this paper. We enumerate these below.

1. Efficient discovery: Brokers are discovered very efficiently. Furthermore, this approach ensures that the broker will be connected to one of the closest available brokers if it does indeed possess the right credentials that these brokers might need.
2. No single point of failure: There is no single point of failure within the system. The system can sustain a variety of failures (outlined in previous section) and can also sustain a variety of message losses.

3. Incorporation of brokers: The scheme that we have outline can incorporate new broker nodes efficiently into the system. Since broker discovery responses include the usage metric, a newly added broker within a cluster would be preferentially utilized by the discovery algorithms.
4. Private BDNs: It is very easy to set up private BDNs within this scheme.

9 Experimental Results

To test the performance of the system we consider various topologies formed using five distributed brokers. We have selected nodes which are separated by significant network distances. Based on the results we can fairly conclude that broker discovery in local networks (brokers separated by very small network distance such as in the same institution) would be as worse as the tests indicate or better.

The various machines used are summarized in Table 1.

Machine	Location	Machine Specification (using uname –a)	JVM Version
complexity.ucs.indiana.edu	Indianapolis, IN, USA	SunOS complexity 5.9 Generic_112233-03 sun4u sparc SUNW,Sun-Fire-880	Java HotSpot(TM) Client VM (build 1.4.2-beta-b19, mixed mode)
webis.msi. umn.edu	UMN, MN Univerity of Minnesota, Minneapolis, MN, USA	Linux webis 2.6.9-gentoo-r4 #1 SMP Mon Dec 6 23:31:06 CST 2004 x86_64 AMD Opteron(tm) Processor 240 AuthenticAMD GNU/Linux	Java HotSpot(TM) 64-Bit Server VM (build Blackdown-1.4.2-01, mixed mode)
tungsten. ncsa.uiuc. edu	NCSA, UIUC IL, USA	Linux tuna 2.4.20-31.9smp_perfctr_lustre #2 SMP Thu Jun 24 21:02:14 CDT 2004 i686 i686 i386 GNU/Linux	Java HotSpot(TM) Client VM (build 1.4.1_01-b01, mixed mode)
pamd2.fsit.fsu.edu	FSU, FL Florida State University, Tallahassee, FL, USA	Linux pamd2 2.4.25 #1 SMP Mon Mar 29 11:09:25 EST 2004 i686 unknown unknown GNU/Linux	Java HotSpot(TM) Client VM (build Blackdown-1.4.1-beta, mixed mode)
bouscat.cs.cf.ac.uk	Cardiff, UK Cardiff Univerity, Cardiff, UK	Linux bouscat.cs.cf.ac.uk 2.4.2-2smp #1 SMP Sun Apr 8 20:21:34 EDT 2001 i686 unknown	Java HotSpot(TM) Client VM (build 1.4.1_01-b01, mixed mode)

Table 1 Summary of Machines used in testing process

To test our system we consider three broker network topologies. The first topology is unconnected broker network (Figure 1). Here there exist brokers registered in the Broker Discovery Node (BDN), however they may or may not be connected with each other.

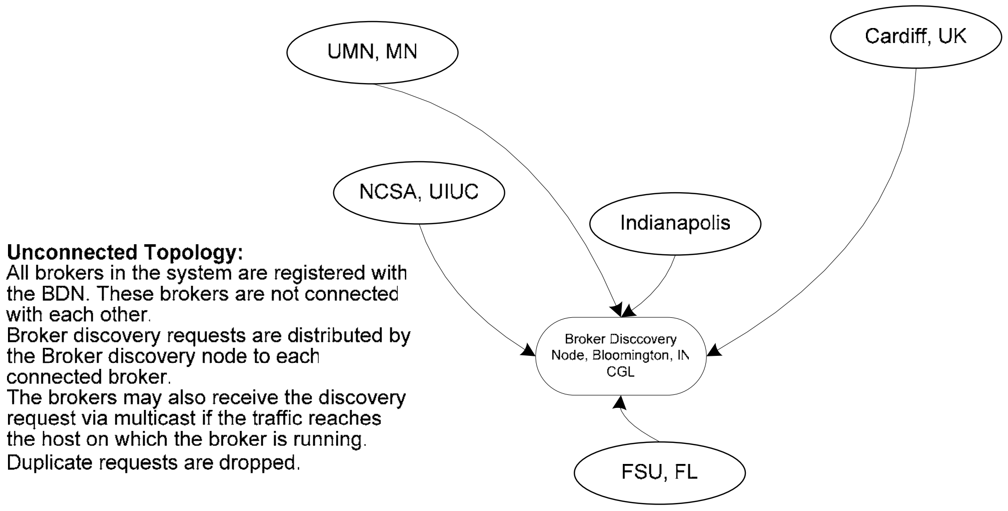

UMN, MN
Cardiff, UK
NCSA, UIUC
Indianapolis
Unconnected Topology:
All brokers in the system are registered with the BDN. These brokers are not connected with each other.
Broker discovery requests are distributed by the Broker discovery node to each connected broker.
The brokers may also receive the discovery request via multicast if the traffic reaches the host on which the broker is running.
Duplicate requests are dropped.
Broker Disccovery Node, Bloomington, IN CGL
FSU, FL

Figure 1 Unconnected Topology

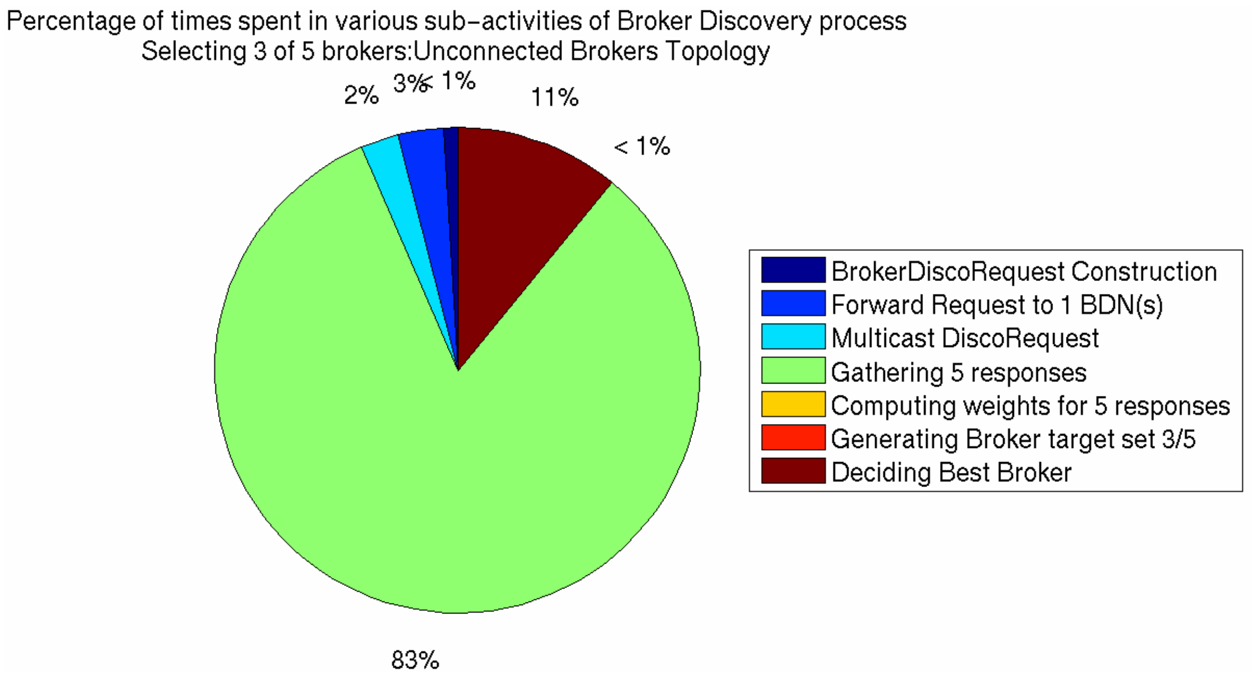
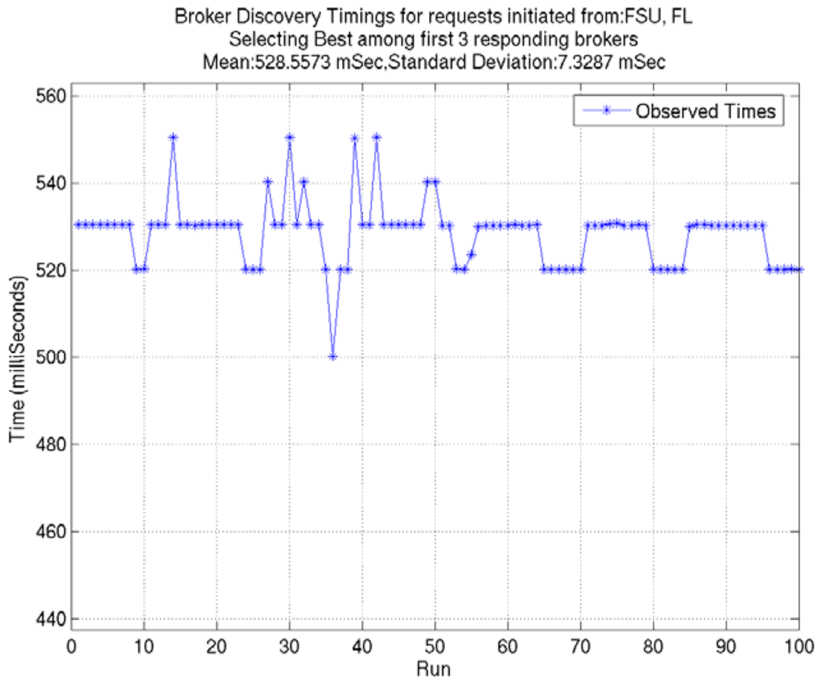


Figure 2 Percentage of time spent in various sub-activities of Broker Discovery in Unconnected Topology

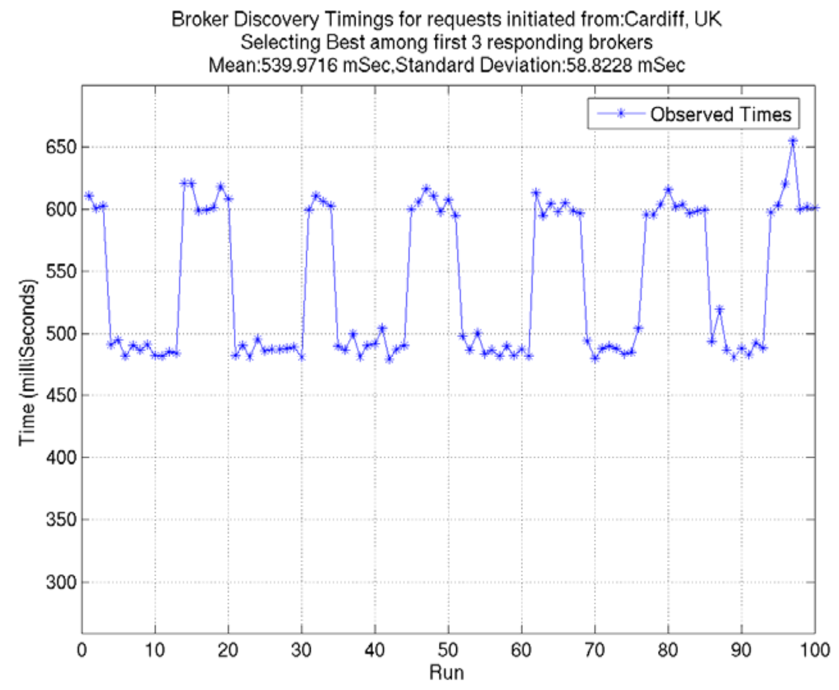
Note that in this case the BDN has to distribute the discovery request to each registered broker. This scenario implies O (N) distribution and would be inefficient in disseminating broker discovery requests. We observe (Figure 2) that maximum time (about 83 %) is spent by the client in waiting for the initial responses. This test was carried out by running the broker discovery client in Bloomington.

We also noted the total discovery times required to find the best broker for the above topology when the discovery process is initiated from different sites. Figure 3 thru Figure 7 show the corresponding results. The discovery process was carried out 120 times and the first 100 results were selected after removing outliers.



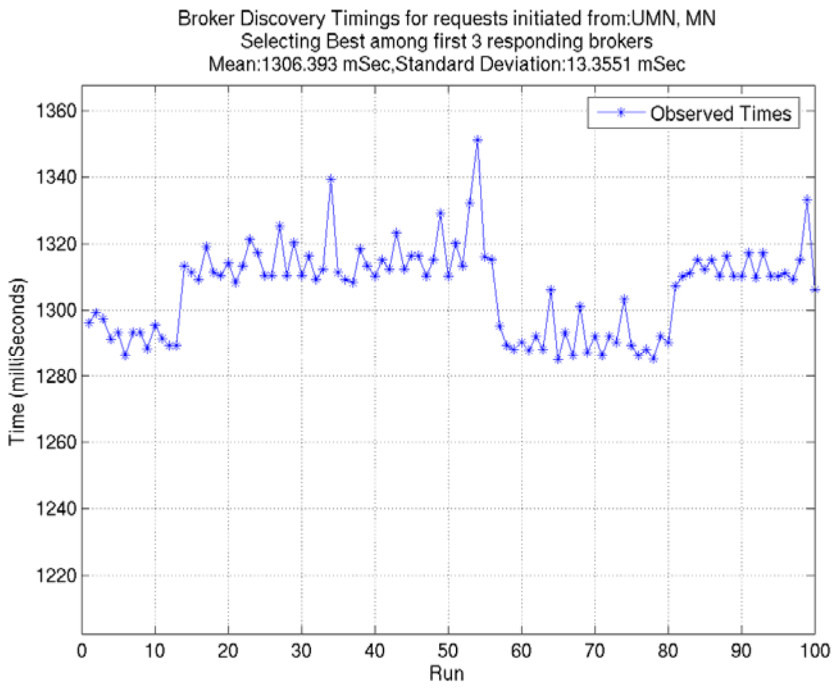
Metric	Time (MilliSec)
Mean	528.5573
Standard deviation	7.3287
Maximum	550.421
Minimum	500.283
Standard Error	0.73287

Figure 3 Time required for discovery with Client in FSU, FL



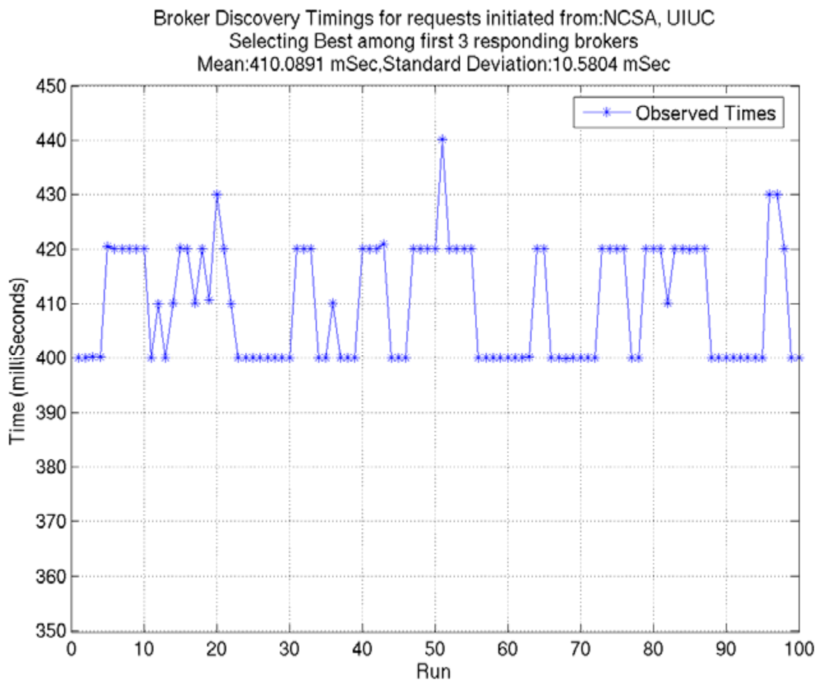
Metric	Time (MilliSec)
Mean	539.9716
Standard deviation	58.8228
Maximum	654.811
Minimum	478.899
Standard Error	5.8823

Figure 4 Time required for discovery with Client in Cardiff, UK



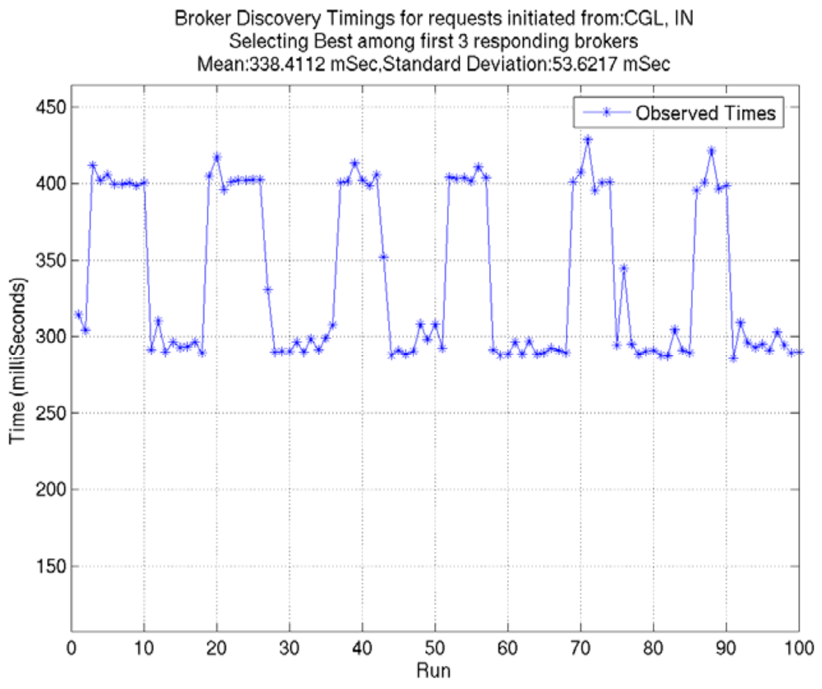
Metric	Time (MilliSec)
Mean	1306.393
Standard deviation	13.3551
Maximum	1351.214
Minimum	1285.102
Standard Error	1.3355

Figure 5 Time required for discovery with Client in UMN, MN



Metric	Time (MilliSec)
Mean	410.0891
Standard deviation	10.5804
Maximum	440.121
Minimum	399.973
Standard Error	1.058

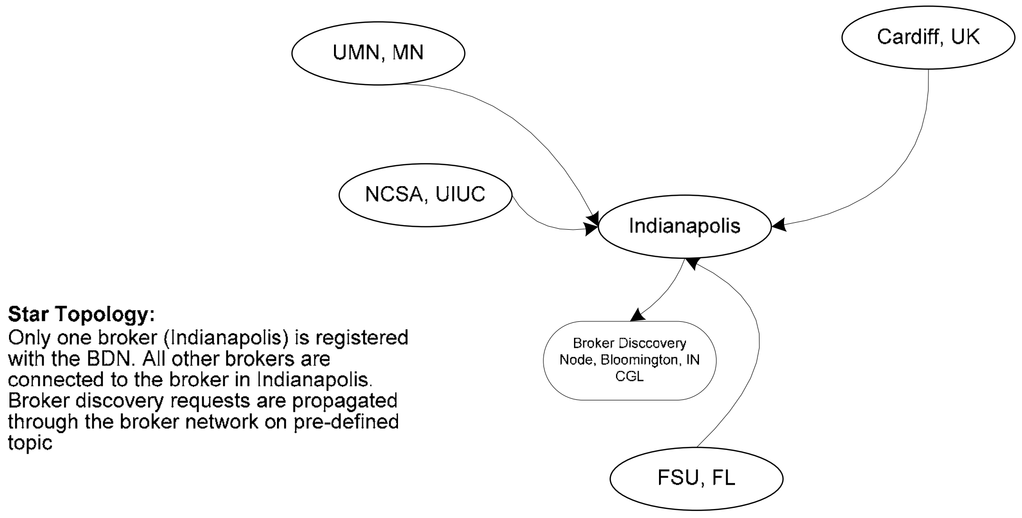
Figure 6 Time required for discovery with Client in NCSA, UIUC, IL



Metric	Time (MilliSec)
Mean	338.4112
Standard deviation	53.6215
Maximum	428.884
Minimum	286.026
Standard Error	5.3622

Figure 7 Time required for discovery with Client in Bloomington, IN

Our next topology shows the behavior of the system in presence of a star topology. The system is illustrated in Figure 8. Here the broker network is responsible for disseminating the broker discovery requests to other existing brokers in the network.



UMN, MN
Cardiff, UK
NCSA, UIUC
Indianapolis
Star Topology:
Only one broker (Indianapolis) is registered with the BDN. All other brokers are connected to the broker in Indianapolis. Broker discovery requests are propagated through the broker network on pre-defined topic
Broker Disccovery Node, Bloomington, IN CGL
FSU, FL

Figure 8 Star Topology

It was observed that the time required for waiting for the initial set of responses decreased significantly. The results are shown in Figure 9.

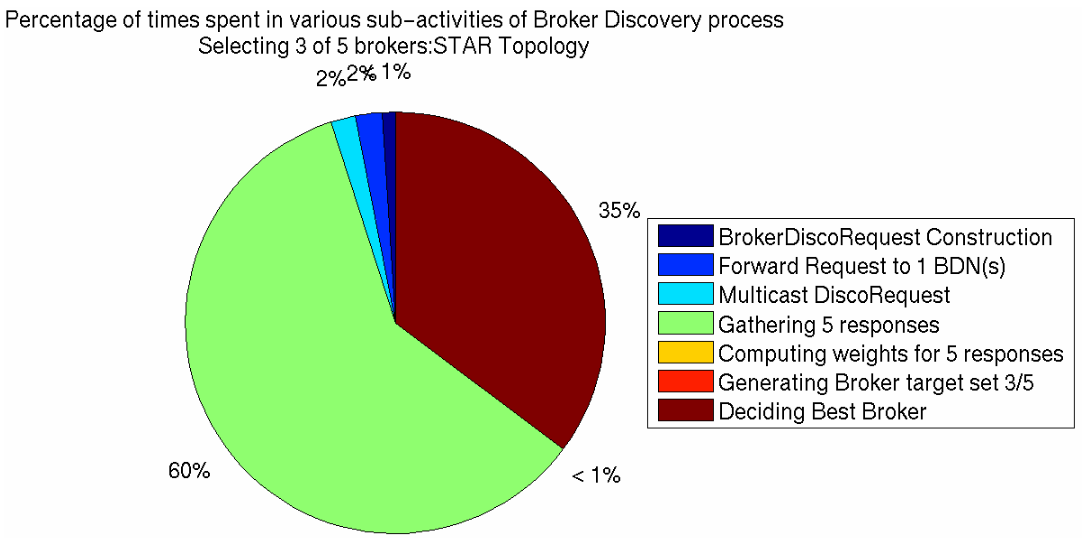


Figure 9 Percentage of time spent in various sub-activities of Broker Discovery in Star Topology

Our final topology is the linear topology wherein only one broker is registered with the BDN. All other brokers are connected to each other in a linear fashion. Our choice for selecting the links for this topology was arbitrary and we believe that any other variation of the scheme would produce similar results. The topology is shown in Figure 10.

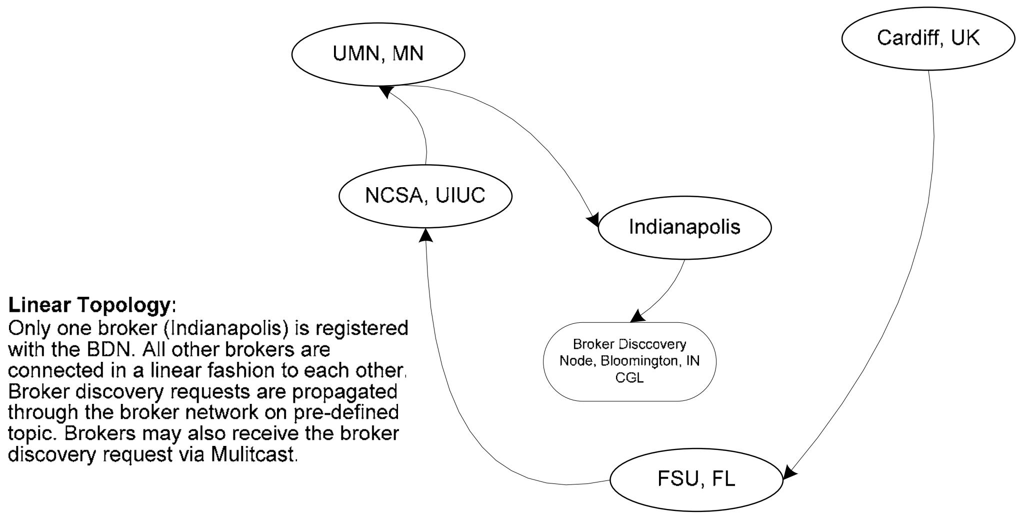

UMN, MN
Cardiff, UK
NCSA, UIUC
Indianapolis
Linear Topology:
Only one broker (Indianapolis) is registered with the BDN. All other brokers are connected in a linear fashion to each other. Broker discovery requests are propagated through the broker network on pre-defined topic. Brokers may also receive the broker discovery request via Mulitcast.
Broker Disccovery Node, Bloomington, IN CGL
FSU, FL

Figure 10 Linear Topology

It was observed that the time spent in waiting for the initial set of responses although better than the first case was still poor compared to the second case. We conclude that the performance is improved over the unconnected topology because the brokering network uses optimized routing to disseminate request within the brokering network, however it still takes finite amount of time for the request to reach the last broker in the chain. Figure 11 shows the relative percentage of the times required for various steps involved in the discovery process.

From the above results we note that in each case, the maximum time is spent in waiting for the initial responses. As the number of brokers increases we face the problem of scalability as waiting for more brokers would badly affect the total time in making a decision on the best broker to connect to. To address this problem we introduce a timeout period before proceeding to make the decision. The timeout period is a configurable value and specifies the amount of time a client is willing to wait to gather discovery responses. A small timeout period would decrease the total time in arriving at a decision, however we risk collecting only few broker responses depending on when the discovery request reaches the broker and the broker decides to reply. A large timeout value implies more time is spent waiting for responses to arrive. This might unnecessarily increase the time of discovery process if fewer brokers exist in the system or only a few brokers respond to the discovery request than the minimum number of responses specified.

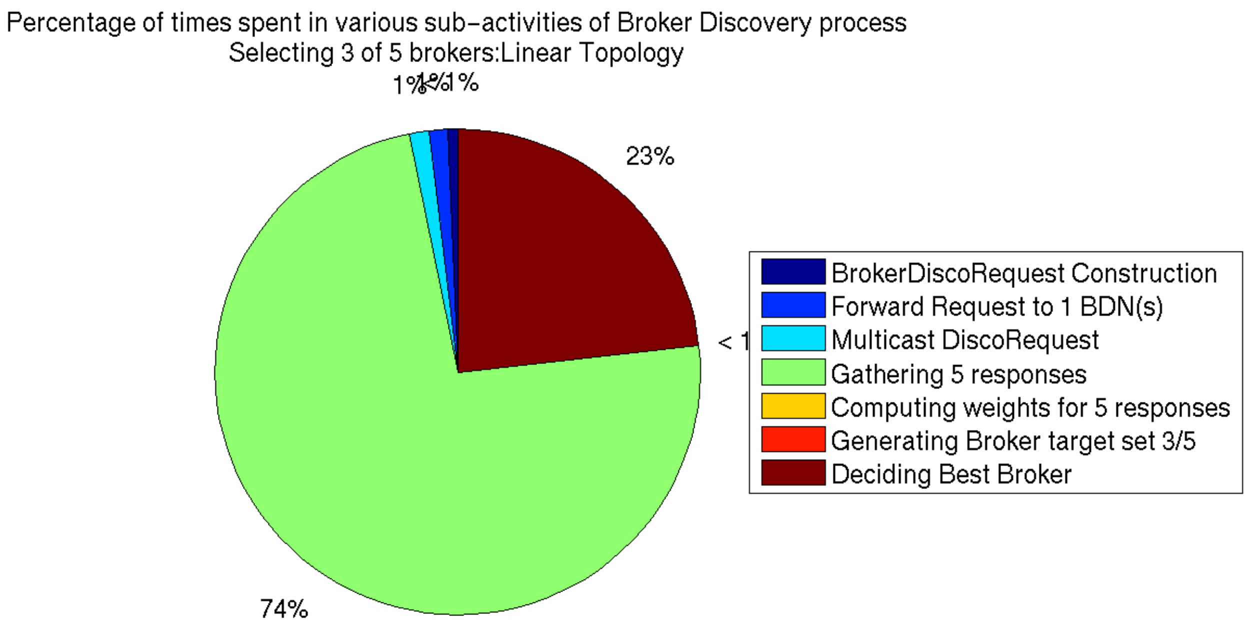
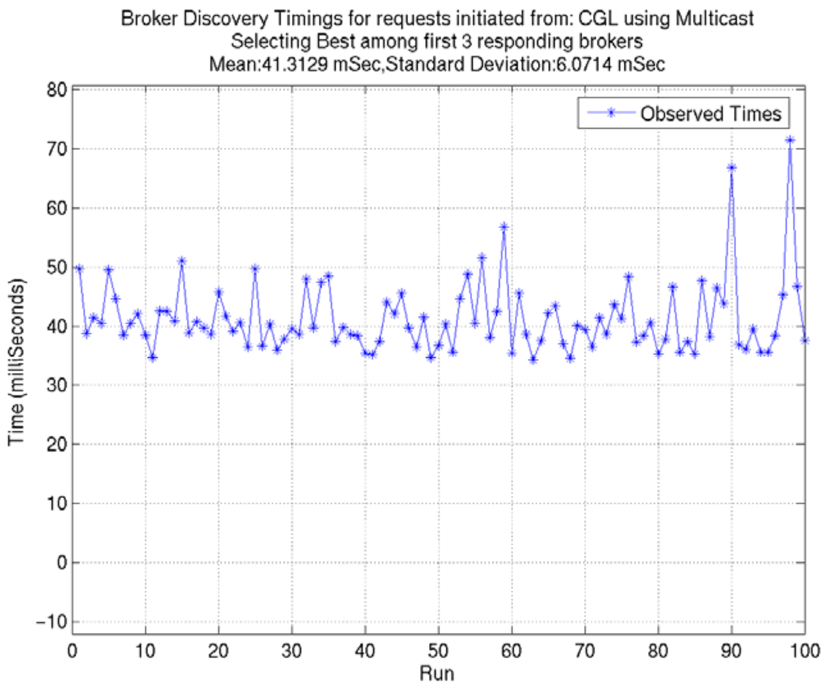


Figure 11 Percentage of time spent in various sub-activities of Broker Discovery in Linear Topology

This problem may be partially solved by also multicasting the request so that the brokers in the local networks would get the request directly rather than being forwarded through some broker discovery node. The multicast scheme may not work if some brokers choose to listen on a different multicast address or port or if the multicast service is disabled for a particular set of brokers. We carried out tests to see how the system works with multicasting. Figure 12 shows the results. However, since multicast was disabled for network traffic outside the lab, the multicast requests could only reach to those brokers which were in the lab. Thus, these results do not reflect multicast results for brokers at outside the lab realm.



Metric	Time (MilliSec)
Mean	41.3129
Standard deviation	6.0714
Maximum	71.416
Minimum	34.303
Standard Error	0.60714

Figure 12 Broker Discovery times using ONLY multicast

We also introduce the maximum responses a client is willing to consider and the broker target set size. Usually a client might be willing to risk more timeout period but specify that only the first N responses must be considered. We then shortlist the set of size N to arrive at a broker target set, T, such that `size(T) <= size(N).`

The target set is decided based on the BrokerDiscoveryResponse. The BrokerDiscoveryResponse contains the total memory available to the broker, the total amount of used memory, the number of links the broker is connected to and possibly the CPU load at the broker. We weigh each of these metrics and compute the weight associated in selecting a particular broker. The values for weighing various factors are configurable and will help the client to give preference for a specific metric with respect to other factors. For instance the weighing may be done as follows

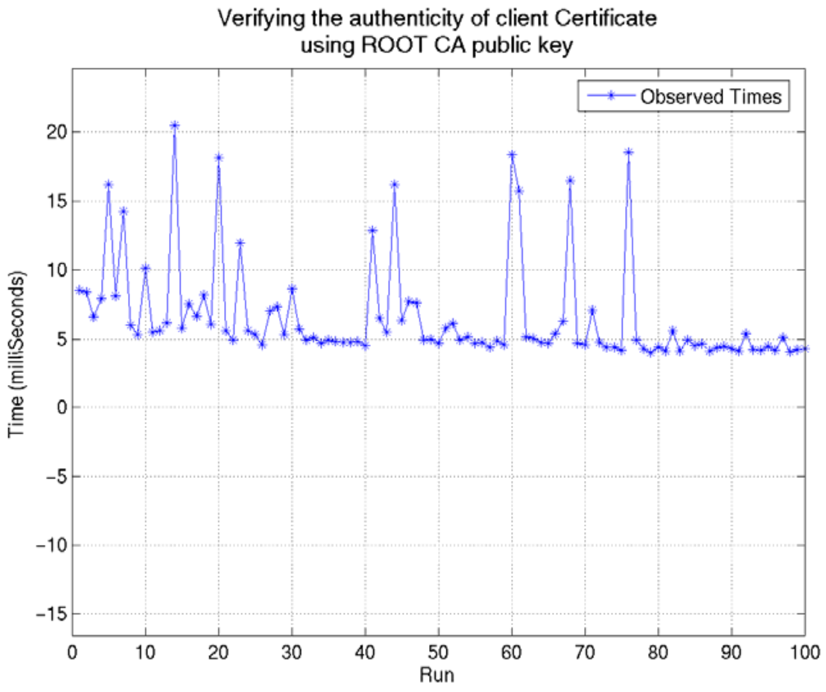
```
// Initialize weight
double weight = 0.0;

// Higher the better
```

```
weight += (freeMem / totalMem) * WEIGHTAGE_FREE_TO_TOTAL_MEMORY;
weight += (totalMem / (1024 * 1024)) * WEIGHTAGE_TOTAL_MEMORY;

// Lower the better
weight -= numLinks * WEIGHTAGE_NUM_LINKS;

// OTHER factors may be similarly added to compute the weight
```

The received results are then sorted using the weights and we select the first `size(T)` brokers to arrive at the broker target set.

9.1 Security Considerations

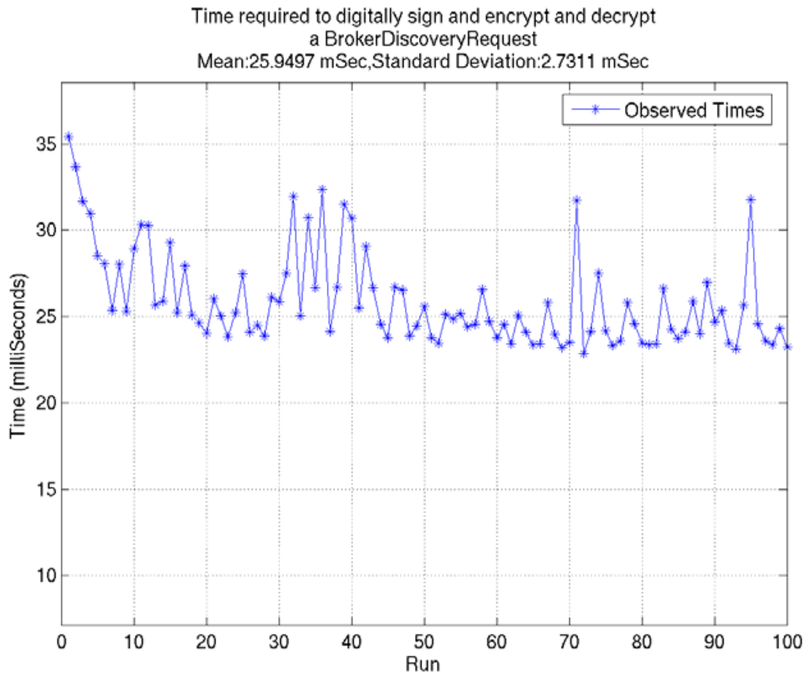
Discovery operations usually entail giving broker details which might be used by a malicious user to launch attacks against the broker network. In order to secure the discovery process against such attacks, a discovery request and response may be secured by sending credentials verifying the authenticity of the clients and also encrypting the discovery request and response.



Metric	Time (MilliSec)
Mean	6.6104
Standard deviation	3.7214
Maximum	20.51
Minimum	4.0390
Standard Error	0.3721

Figure 13 Time required in validating a X.509 Certificate

Although our initial prototype does not incorporate any security schemes, the broker and client may be augmented with digital certificates and PKI authentication schemes to secure the request and response messages. We carried out tests to estimate the cost associated with such a scheme. Figure 13 shows the time required to verify a client's identity. We also time the cost associated with signing and encrypting a broker discovery request and decrypting it (Figure 14). These tests were performed on a Pentium M 2.0 GHz machine with 512 MB RAM. The Java virtual machine used was Java Standard Edition HotSpot(TM) Client VM (build 1.4.2_06-b03, mixed mode). We note from the timings that these costs are acceptable in most systems which would require such a feature.



Metric	Time (MilliSec)
Mean	25.9497
Standard deviation	2.7311
Maximum	22.862
Minimum	35.428
Standard Error	0.27311

Figure 14 Time required to digitally sign and encrypt and later extract the BrokerDiscoveryRequest

10 Related Work

In this section we summarize related work pertaining to the discovery of nodes in distributed systems. IDMaps [8] examines the network distance prediction problem from a topological point of view. In this approach special HOPS servers maintain a virtual topology map of the Internet, consisting of end hosts and special hosts called Tracers. The distance between two peers *A* and *B* is then estimated as the distance between *A* and it's nearest Tracer T_1, plus the distance between *B* and it's nearest Tracer T_2, plus the shortest path distance between the Tracers T_1 and T_2 over the Tracer virtual topology. The prediction accuracy improves with the growing number of tracers. This approach however requires Internet-wide deployment of measurement entities. Hotz [9] approach requires limited infrastructure support and uses a small set of measurement reference points called landmarks or beacons. The distance between each application peer and landmarks is measured, and processed to obtain the nearest peer using triangulation methods. The JXTA P2P system [10] uses rendezvous peers to locate peers with matching resource availability constraints. This scheme however assumes knowledge of existence of rendezvous peers in the network and the means to connect to at least one of these peers. The Tiers [11] approach uses hierarchical grouping of peers for improving the scalability of the system. Our approach to locating nearest brokers is different from the above approaches in that, we do not place any constraints on the existence of Broker Discovery Nodes (BDNs). In the absence of BDNs the request may optionally be also propagated using multicast to nearest brokers. The brokers also propagate discovery requests on a predefined topic thus guaranteeing that the request can reach each broker connected in the network. In the Global Network Positioning (GNP) [12] approach the network distances are predicted using a distance function over a set of coordinates that characterizes the location of the peer in the Internet. In our system we only compute the distances using UDP PING to each of the brokers in target broker set. This PING operation may be repeated multiple times to compute the average network Round Trip Time (RTT) between the peer and the broker in question. Usually the broker target set is limited to a very small number, between 5 and 20, and is configurable. In publish/subscribe systems Sienna [13] utilizes schemes so that subscribers are clustered closer to broker nodes that host publishers that those subscribers are interested in. A related work Ref [14] employs strategies to make sure brokers that manage similar subscriptions are clustered together.

11 Conclusions

In this paper we have outlined our strategy for discovering brokers in distributed settings. We have also included results from our experiments in WAN (Wide Area Network) settings. The results have demonstrated the feasibility of using our scheme in distributed settings. The advantages of our scheme have been outlined in section 8.

Acknowledgements

The authors would like to thank Dr. Gordon Erlebacher, Ben Kadlec and Dr. Mary Thomas for granting us computer system accounts in FSU, UMN and SDSU. The authors would also like to thank our colleagues in NCSA and Cardiff for granting us host access.

References

1. Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.
2. Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil. Building Messaging Substrates for Web and Grid Applications. (To appear) in the Special Issue on Scientific Applications of Grid Computing in the Philosophical Transactions of the Royal Society of London 2005.
3. Geoffrey Fox and Shrideep Pallickara. Deploying the NaradaBrokering Substrate in Aiding Efficient Web & Grid Service Interactions. Special Issue of the Proceedings of the IEEE on Grid Computing. Vol 93, No 3. pp 564-577. March 2005.
4. Shrideep Pallickara and Geoffrey Fox. On the Matching Of Events in Distributed Brokering Systems. Proceedings of IEEE ITCC Conference on Information Technology. April 2004. pp 68-76 Volume II.
5. Shrideep Pallickara and Geoffrey Fox. A Scheme for Reliable Delivery of Events in Distributed Middleware Systems. Proceedings of the IEEE International Conference on Autonomic Computing. New York, NY. pp 328-329
6. Geoffrey Fox, Sang Lim, Shrideep Pallickara and Marlon Pierce. Message-Based Cellular Peer-to-Peer Grids: Foundations for Secure Federation and Autonomic Services. Journal of Future Generation Computer Systems. Volume 21, Issue 3, pp 401-415 (1 March 2005). Published by Elseiver.
7. Hasan Bulut, Shrideep Pallickara and Geoffrey Fox. Implementing a NTP-Based Time Service within a Distributed Brokering System. 2004 ACM International Conference on the Principles and Practice of Programming in Java. pp 126-134.
8. P. Francis et. al. *IDMaps: A Global Internet Host Distance Estimation Service*. In IEEE/ACM Transactions on Networking Oct. 2001
9. Steven Michael Hotz. *Routing information organization to support scalable interdomain routing with heterogeneous path requirements* PhD Thesis, University of Southern California, Los Angeles, CA, 1996

10. Sun Microsystems. The JXTA Project and Peer-to-Peer Technology http://www.jxta.org
11. Suman Banerjee, Christopher Kommareddy, Bobby Bhattacharjee *Scalable Peer Finding on the Internet*, To appear in Proceedings of Global Internet Symposium, Globecom 2002, Taipei, Taiwan, November 2002.
12. T.S. Eugene Ng, Hui Zhang *Predicting Internet Network Distance with Coordinates-Based Approaches*, In Proceedings of IEEE INFOCOM, pages 170--179, New York, NY, USA, June 2002.
13. Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pages 219–227, Portland OR, USA, 2000.
14. Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, Antonino Virgillito: Subscription-Driven Self-Organization in Content-Based Publish/Subscribe. Proceedings of the IEEE International Conference on Autonomic Computing. New York, NY.: 332-333.