Society for Science and Education United Kingdom
TNC Transactions on Networks and Communications
Volume 3, Issue 2
ISSN: 2054 - 7420

Membership Protocols for the iTrust Network

[1]Yung-Ting Chuang, [2]Peter M. Melliar-Smith, [3]Louise E. Moser, [4]Isai Michel Lombera
Department of Electrical and Computer Engineering, University of California, Santa Barbara
Santa Barbara, USA
[1]ytchuang@ece.ucsb.edu, [2]pmms@ece.ucsb.edu, [3]moser@ece.ucsb.edu, [4]imichel@ece.ucsb.edu

ABSTRACT

The iTrust system is a decentralized and distributed system for information publication, search and retrieval over the Internet, which is designed to make it difficult to censor or filter information. In this paper, we present four membership protocols for the iTrust network, namely, the non-adaptive, retry, adaptive and combined adaptive membership protocols. We compare the performance of these membership protocols, with respect to four performance metrics, namely, membership accuracy, match probability, response time and message cost, for various parameter values when the membership churn is high and when the membership is stable.

Keywords: Membership protocol, Membership churn, Information publication, search and retrieval, Peer-to-peer network

1 Introduction

Our modern society depends on uncensored publication and retrieval of information over the Internet. Currently, for reasons of efficiency and economy of scale, centralized search engines dominate Internet search and retrieval. However, centralized search engines can easily censor, filter or bias the information they provide. An effective decentralized and distributed search system, as an alternative to conventional centralized search, can help to ensure the free flow of information over the Internet.

The iTrust system [5, 15, 16] is a decentralized and distributed system for information publication, search and retrieval over the Internet, which makes it difficult to censor or filter information. iTrust is based on a peer-to-peer network that is deliberately unstructured to reduce the risk of censorship. The communication cost for iTrust is greater than that for centralized search engines or structured peer-to-peer systems. However, if users suspect that information important to them is being censored or suppressed, they should be willing to incur that extra cost. Nevertheless, we try to minimize any additional cost.

In iTrust, a source node with information to share distributes metadata and a URL for that information, to a subset of the nodes in the membership of the iTrust network, chosen at random. A requesting node, seeking information, generates and distributes its query to a subset of the nodes in the membership, chosen at random. Nodes, that receive metadata and a matching query, send the URL for the information to the requesting node, which retrieves the information from the source node. If the metadata and the requests are distributed to enough nodes, the probability of a match and the consequent retrieval of information are very high [15].

The iTrust system is designed to protect against malicious nodes that attempt to disrupt the search for information. A potential attack might insert, into the membership of the iTrust network, nodes

DOI: 10.14738/tnc.32.1166
Publication Date: 30th April, 2015
URL: http://dx.doi.org/10.14738/tnc.32.1166

that behave normally except that, for sensitive information, they do not match metadata and requests, thus reducing the probability of a match. To guard against such an attack, each node computes, locally and independently, an estimate of the proportion of non-matching nodes in the membership. The node then increases the number of nodes to which it distributes metadata and requests, to maintain the desired high probability of a match.

Another potential attack on iTrust might distribute metadata and URLs for misleading information. iTrust itself does not attempt to detect or downrank such information. If it were to do so, iTrust itself would be engaging in a form of censorship. Rather, as described in [18], a requesting node downloads, from the source node, a table of frequencies of terms in the document. The requesting node matches those terms against its query to generate a relevance ranking for the document. Whether the document is useful or misleading is determined by the user.

iTrust does not attempt to provide anonymity for its users. Anonymity is quite distinct from censorship. Anonymity attempts to hide the identities of users who publish, or search for, information. Censorship attempts to hide the information itself. iTrust could be coupled with an anonymity service [8, 20], so that users could publish or retrieve information without disclosing their identities. We are investigating combining one of those anonymity services with iTrust.

An extensive literature on membership exists, but most of that work is not relevant to iTrust. Previous membership protocols [3] aim to achieve an agreed accurate membership based on a consensus algorithm [2]. The membership protocols for iTrust are simpler and less costly than previous membership protocols, because iTrust does not need to achieve an agreed accurate membership but can operate effectively with an approximate membership for a rapidly changing network. As large-scale peer-to-peer networks become more common, effective approximate membership protocols will become more important.

In iTrust, the membership consists of the nodes that participate in the iTrust network (also referred to as the participating nodes). Each node has a local view of the membership, which the membership protocols aim to keep close to the actual membership. A node can join the membership at any time; likewise, a node can leave the membership at any time, either voluntarily or because it is faulty or disconnected.

In [5, 15, 16], we presented an overview of the iTrust system and the principles on which iTrust is based. In [1], we presented a version of iTrust in which nodes maintain a randomized subset of the membership, a neighborhood. In large networks, containing millions of nodes, the cost of maintaining the entire membership is excessive. iTrust can operate effectively with the membership protocols presented in this paper using smaller neighborhoods.

In [4], we described a non-adaptive membership protocol for iTrust. In this paper, we present three additional membership protocols for iTrust, namely, the retry, adaptive and combined adaptive membership protocols. We compare these membership protocols with respect to four performance metrics, namely, membership accuracy, match probability, response time and message cost, for various parameter values when the membership churn is high and when the membership is stable.

The novel contributions of this paper are:

- Distributed membership protocols for unstructured peer-to-peer networks, such that each node locally determines its view of the membership without communicating additional messages.

- Approximate membership protocols for unstructured peer-to-peer networks, in which each node maintains its own local view of the membership, which it tries to keep close to the actual membership.
- Distributed local estimators for the size of the membership and the membership churn.
- Non-adaptive, retry, adaptive and combined adaptive membership protocols.

The interested reader can find the source code, the user manuals and additional papers on iTrust at http://itrust.ece.ucsb.edu.

2 Related Work

Mischke and Stiller [17] have characterized peer-to-peer networks for distributed search and retrieval as structured or unstructured. The iTrust system is based on an unstructured network, like Gnutella [10], with random distribution of the metadata and the requests to $\lceil 2 \sqrt{N} \rceil$ nodes, like the system of Lv et al. [14].

Prior work on membership has focused on forming an agreed accurate membership, and is typically based on a consensus algorithm. Chandra et al. [2] have shown that it is impossible to achieve an agreed accurate membership in the presence of unreliable processors and unreliable communication. Chockler et al. [3] provide a comprehensive survey of membership protocols and group communication systems, and of their formal specifications. Schiper and Toueg [22] provide an elegant formalization of the membership problem that distinguishes between maintaining and agreeing on a set of members and determining which processes are working and should be members. The iTrust membership protocols do not aim to achieve an agreed accurate membership based on a consensus algorithm. Rather, they allow each member to have its own local view of the membership, and they aim to keep that local view close to the actual membership, with a much lower message cost than consensus-based membership protocols.

Ganesh et al. [9] present a membership service, named SCAMP, for gossip-based protocols that operates in a decentralized and self-configured manner, where no peer has global knowledge of the membership. A node that wishes to join (leave) the membership notifies some nodes in the network to add (remove) it to (from) their views. To prevent a node from becoming isolated, a node periodically tries to discover new nodes if it does not receive any messages for a given period of time. Compared to SCAMP, the iTrust membership protocols place more emphasis on maintaining a node's local view of the membership when the membership churn is high.

Zage et al. [25] present a network-aware and distributed membership protocol that biases neighbor selections towards beneficial nodes, based on multiple system metrics and social network patterns. They demonstrate the effectiveness of their protocol for a network with a high churn rate, through emulation. In the iTrust membership protocols, the nodes do not maintain their views of the membership through biased neighbor selections, which might allow malicious nodes to subvert the membership. Rather, they discover newly joining nodes and detect leaving nodes through the normal course of distributing metadata and requests, which reduces the message cost.

Liu et al. [13] describe a novel age-based membership protocol with a conservative neighbor maintenance scheme under churn, to retain desirable properties such as a low network diameter and a low clustering coefficient. Thus, a bootstrapping node recommends, to a newly joining node, only the nodes that have remained in its view for a long period of time. However, with their protocol, a newly joining node might not discover other nodes very quickly, whereas an older node might have knowledge of a larger number of other nodes. In the iTrust membership protocols, a boostrapping node sends its entire membership to a newly joining node, regarding all nodes as equals.

Voulgaris et al. [24] present a membership management protocol, named CYCLON, for unstructured peer-to-peer networks, in which each node maintains a small and fixed-size neighbor list. They describe a shuffling protocol for large networks and provide an experimental analysis in which they examine the clustering coefficient and the node degree distribution. The iTrust membership protocols differ from CYCLON in that each node tries to discover as many nodes as possible to include in its local view. In other work [1], we have also investigated the use of neighborhoods and de-clustering (shuffling) for very large iTrust networks.

In BubbleStorm [12, 23], when a node joins the network, it finds an existing connection between two peers and interposes itself between them. When a node leaves the network, it re-connects those two peers before it leaves. If a node crashes, a neighboring peer adds a connection to the other peer when it discovers the crashed node. Thus, BubbleStorm aims to maintain a fixed node degree at all of the nodes in the network. The iTrust membership protocols do not try to maintain a fixed node degree but, rather, allow each node to maintain its own local view of the membership.

PlanetP [7] uses a global index that describes all of the peers and their metadata in a Bloom filter, which it replicates throughout the network using gossiping. iTrust does not use gossiping to distribute the entire membership but, rather, allows each node to maintain its own local view by discovering newly joining nodes and detecting leaving (non-operational) nodes through the normal course of operation of the iTrust messaging protocol.

Richardson and Cox [21] provide an overlay of indices to achieve search and ranking in an unstructured peer-to-peer network. A malicious attack might distort the ranking, by not reporting relevant documents (censorship), by increasing the rank of selected documents, or by not reporting highly ranked documents. The authors discuss the distributed local estimation of system-wide metrics, such as average document length, which are expected to follow a normal distribution. Attempts to distort the metrics introduce skew into the distribution, which they can detect. Their work is somewhat similar to our work [18] on search and ranking in iTrust.

Reiter et al. [20] and Freedman et al. [8] describe anonymous communication layers, with associated membership algorithms. Freenet [6] maintains a semi-structured network with a small world topology for better routing that also provides anonymity. We plan to investigate whether such anonymity schemes are appropriate for iTrust. However, anonymity by itself is not enough to prevent censorship.

Gramoli et al. [11] and Pruteanu et al. [19] use a strategy for estimating churn similar to the strategy we use for iTrust. To refine their churn estimates, they exchange churn estimates between neighbors, which exposes those estimates to malice. In contrast, iTrust does not exchange churn estimates and, thus, is more resistant to malice; even so, the churn estimates of iTrust are quite accurate.

3 iTrust Messaging Protocol

First, we briefly describe the iTrust messaging protocol, because the iTrust membership protocols are dependent on it. Some of the nodes in the membership, referred to as the source nodes, produce information, and make that information available to other nodes. Other nodes in the membership, referred to as the requesting nodes, make requests (queries) and retrieve information from the source nodes.

The steps involved in the iTrust messaging protocol are given below and are illustrated in Figure 1.

1. A source node produces metadata that describes its information, and distributes the metadata, along with the URL of the matching document, to a subset of nodes randomly chosen from its local view of the membership.
2. A requesting node generates a request (query) that contains keywords, and distributes its request to a subset of nodes randomly chosen from its local view of the membership.
3. When a node receives a request containing keywords that match metadata it holds (referred to as an encounter or a match), the node returns, to the requesting node, the URL of the matching document at the source node.
4. The requesting node then uses the URL, provided by the matching node, to retrieve the document from the source node.

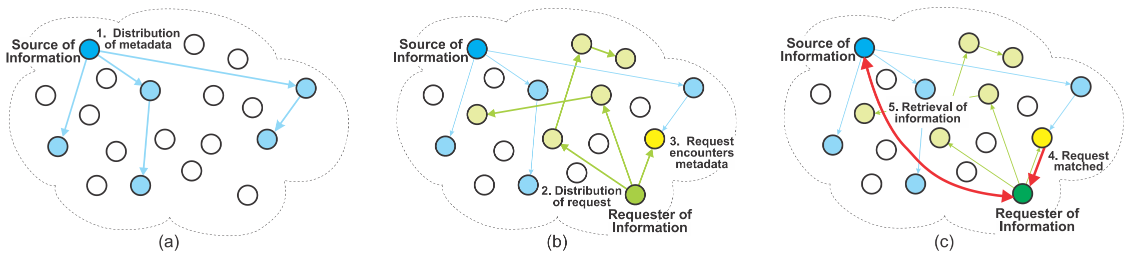

Figure 1. (a) A source node distributes metadata, describing its information, to a subset of nodes randomly chosen from its local view of the membership. (b) A requesting node distributes its request to a subset of nodes randomly chosen from its local view of the membership. (c) One of the nodes matches the metadata and the keywords in the request, and reports the match to the requesting node, which then retrieves the information from the source node

A match between the keywords in a request received by a node and the metadata held by a node might be an exact match or a partial match, or it might correspond to synonyms.

Distribution of the metadata and the requests to relatively few nodes suffices to achieve a high probability of a match. In an iTrust membership with N nodes, distribution of the metadata to $M = \lceil 2\sqrt{N} \rceil$ nodes and distribution of the requests to $R = \lceil 2\sqrt{N} \rceil$ nodes results in a probability of a match that exceeds 0.9817, derived in [15] from the hypergeometric formula for the probability of a match. Distribution of the metadata and the requests to more nodes would result in a higher message overhead, with little improvement in the match probability.

4 iTrust Menbership Protocols

The iTrust messaging protocol, described in Section 3, for metadata and request distribution and for matching and document retrieval depends on a membership, but iTrust does not require an agreed accurate membership, as do some other distributed systems [3]. Rather, iTrust allows each member to have its own local view of the membership, but aims to keep that local view close to the actual membership. We present below the basics of the iTrust membership protocols and, in Sections 6-9, we consider four specific membership protocols, namely the non-adaptive, retry, adaptive and combined adaptive membership protocols. The two adaptive membership protocols utilize the Exponential Weighted Moving Average (EWMA) method. We compare the effectiveness of the four membership protocols when the membership is subject to churn and when the membership is stable.

4.1 Joining the Membership

To join the membership, a node must first obtain the address of a bootstrapping node. To obtain the address of a bootstrapping node, the node uses mechanisms outside the iTrust network, such as conventional Web search, e-mail, Twitter, printed publications, etc.

The steps involved when a node joins the membership are given below, and are illustrated in Figure 2(a).

1. A joining node contacts a bootstrapping node to obtain the bootstrapping node's current view of the membership. A node typically uses a bootstapping node that is known to, and trusted by, the user. Using a malicious bootstrapping node can lead to a seriously distorted membership.
2. The joining node then publishes its joining the membership to a subset of nodes randomly chosen from the view of the membership it obtained from the bootstrapping node.
3. The randomly chosen nodes then add the new node to their local views of the membership.

Another node learns about a new node when it receives a response from a node that is aware of the new node.

4.2 Leaving the Membership

A node may leave the membership either voluntarily, or because it is faulty or disconnected. The steps involved in leaving the membership are simple:

1. To leave the membership, a node simply leaves, without publishing its leaving.

Over time, each node individually discovers the departure of nodes when it sends requests to nodes that do not respond. It is not appropriate to allow a node to publish the departure of another node, because doing so might enable a malicious node to cause the removal of many nodes from the local views of other nodes.

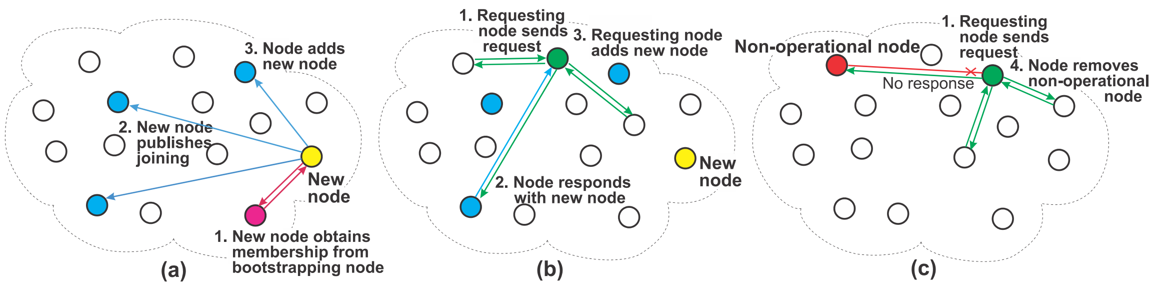
3. Node adds new node
2. New node publishes joining
New node
1. New node obtains membership from bootstrapping node
(a)
1. Requesting node sends request
3. Requesting node adds new node
2. Node responds with new node
New node
(b)
Non-operational node
1. Requesting node sends request
No response
4. Node removes non-operational node
(c)

Figure 2. (a) A node joins the membership by first contacting a bootstrapping node to obtain that node's current view of the membership, and then publishing its joining to randomly chosen nodes in that local view. (b) A requesting node distributes a request to nodes randomly chosen from its view of the membership. A node that receives the request returns the identifier(s) of the node(s) that it recently added to its view. A matching node also returns the URL of the document to the requesting node. (c) The requesting node does not receive a response to its request from a node. The requesting node sees a timeout expire or receives an error code from TCP, and then removes that node from its view of the membership.

4.3 Updating the Membership

In the iTrust messaging protocol described in Section 3, each requesting node expects to receive response messages from only the matching nodes. Other nodes that don't have a match are not required to send a response to the requesting node. We have modified that messaging protocol to enable a requesting node to detect non-operational (leaving) nodes and to discover newly joining nodes, from the responses to its requests.

Now, the requesting node expects each node to which it sent a request to respond with its recently joined member(s), regardless of whether or not that node has a match. Thus, a matching node sends in its response to the requesting node not only the URL of the document at the source node but also the identifier(s) of its recently joined member(s). If it does not have a match, the node responds to the requesting node with the identifier(s) of its recently joined member(s). Thus, the requesting node discovers not only the URLs of the documents, but also newly joined nodes, from the responses to its requests.

If the requesting node doesn't receive a response from a node within a timeout or it receives an error code from TCP, then the non-responding node is considered to have left the membership voluntarily or to be faulty or disconnected, and the requesting node removes that node from its local view of the membership.

The steps involved in updating a requesting node's local view of the membership are given below, and are illustrated in Figure 2(b) and 2(c).

1. A requesting node distributes its request to a subset of nodes randomly chosen from its local view of the membership.
2. A node that receives a request compares the keywords in the request with the metadata it holds. If it finds a match, the node responds to the requesting node with a message that contains the URL of the matching document and also the identifier(s) of its recently joined member(s). A node that doesn't find a match responds to the requesting node with a message that contains the identifier(s) of its recently joined member(s).
3. When the requesting node receives the responses, it adds the new members obtained from the other nodes to its local view of the membership.
4. If the requesting node does not receive a response to its request before a timeout occurs, or if it receives an error code from TCP, then the non-responding node is considered to have left the membership voluntarily or to be faulty or disconnected, and the requesting node removes that node from its local view of the membership.

If the requesting node is also a source node then, after receiving the responses to its request, it distributes its metadata with the URL of the corresponding document to additional nodes, according to the following steps:

1. The requesting node (which is also a source node) calculates the number of nodes to which to distribute its metadata, based on its current view of the membership.
2. Next, the requesting node subtracts the number of nodes to which it previously distributed metadata from the calculated number.
3. Finally, the requesting node distributes its metadata to that many additional nodes, randomly chosen from its current view, but to which it had not sent the metadata previously.

For example, suppose that a requesting node currently has N = 1024 nodes in its local view. It distributes its request to $R = \lceil 2\sqrt{1024} \rceil = 64$ randomly chosen nodes. Suppose further that only 58 nodes reply to the requesting node. From these responses, the requesting node detects that there are 64-58 = 6 non-operational nodes. Suppose that, as a result of receiving the responses from the 58 nodes, the requesting node adds 40 new nodes to its local view. Consequently, the requesting node now has N = 1024-6+40 = 1058 nodes in its view. If the requesting node is also a source node, then it distributes its metadata to 2√1058 - 2√1024 ~ 65-64 = 1 more node randomly chosen from its new view of the membership.

5 Foundations and Experimental Method

5.1 Environmental Variables

Membership churn refers to nodes joining and leaving the membership, and is represented by the following rates:

- JR: The Joining Rate, the number of nodes that join the membership per time unit. For example, JR = 50 means that 50 nodes join the membership per time unit.
- LR: The Leaving Rate, the number of nodes that leave the membership per time unit. For example, LR = 50 means that 50 nodes leave the membership per time unit.

When the membership has a lot of churn, both *JR* and *LR* are high. When the membership is stable, both *JR* and *LR* are low. These rates are an important consideration for the membership protocols. A node can't control or alter *JR* or *LR*, but it can adjust its requesting rate.

5.2 Parameters for the Membership Protocols

The parameters for the membership protocols are:

- *N*: The number of nodes in a node's local view of the membership.
- *LastJ*: The Last Joined members, the number of recently joined members that a node may report to the requesting node. For example, *LastJ* = 2 allows a node to report its two most recently joined members.
- *Try*: The number of times that a requesting node sends its request message, in an attempt to receive responses from $\lceil 2\sqrt{N} \rceil$ nodes. Because some request messages might be sent to non-operational nodes, a requesting node might need to try several times before it receives responses from $\lceil 2\sqrt{N} \rceil$ nodes.
- *TryMax*: The Maximum Try value, i.e., the maximum number of times that a requesting node is allowed to try to send its request message.
- *RR*: The Requesting Rate, the number of times a node sends a request message to $R = \lceil 2\sqrt{N} \rceil$ nodes per time unit. For example, *RR* = 10 means that a node sends 10 distinct request messages per time unit, each of which is sent to $R = \lceil 2\sqrt{N} \rceil$ nodes.
- *RRMin*: The Minimum Requesting Rate, the minimum rate at which a node is allowed to make requests.
- *RRMax*: The Maximum Requesting Rate, the maximum rate at which a node is allowed to make requests.
- *c*: The weighting factor of the Exponential Weighted Moving Average algorithm used by the adaptive membership protocols.

When a node joins the membership, it obtains the values of LastJ, TryMax, RRMin, RRMax and c. These parameters are tunable for the particular network environment.

5.3 Performance Metrics

The performance metrics for the membership protocols are defined in terms of the following quantities:

- L: The number of leaving nodes that a requesting node hasn't detected.
- J: The number of joining nodes that a requesting node hasn't discovered.

- I: The number of nodes in the intersection of the requesting node's current view of the membership and the actual membership.

The requesting node's current view of the membership consists of I+L nodes, whereas the actual membership consists of I+J nodes. Figure 3 illustrates the quantities I, L and J.

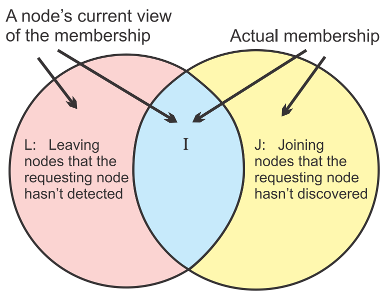
A node's current view of the membership
Actual membership
L: Leaving nodes that the requesting node hasn't detected
I
J: Joining nodes that the requesting node hasn't discovered

Figure 3. A node's current view of the membership vs. the actual membership

The performance metrics for the membership protocols are:

- *LND*: The Leaves Not Detected, the proportion of leaving (non-operational) nodes in its local view that a requesting node has not detected at a particular time, defined by:

$$LND = \frac{L}{I+L} \tag{1}$$

- JND: The Joins Not Discovered, the proportion of newly joined nodes in the actual membership that a requesting node has not discovered at a particular time, defined bv:

$$JND = \frac{J}{I+J} \tag{2}$$

- *MA*: The Membership Accuracy, the number of nodes in a node's current view that are in the actual membership, divided by that number of nodes plus the number of leaving nodes not detected plus the number of joining nodes not discovered, defined by:

$$MA = \frac{I}{I+L+J} \tag{3}$$

Note that *MA* = *I*/(*I*+*L*+*J*) = 1 - (*L*+*J*)/(*I*+*L*+*J*), where *L*+*J* is the number of leaving nodes that the node hasn't detected plus the number of newly joining nodes that the node hasn't discovered and, therefore, (*L*+*J*)/(*I*+*L*+*J*) represents the inaccuracy in the node's local view of the membership.

- *MP*: The Match Probability of one or more responses for a request, averaged over all requesting nodes.
- *RT*: The Response Time for a request, from the time a node starts sending a request to other nodes until it has received all responses, including responses for multiple tries, averaged over all requesting nodes.
- *MC*: The Message Cost per node per time unit, calculated as an average over all nodes over time.

5.4 Measured Values

The membership protocols for iTrust use the following measured values:

- *Left*: The number of nodes that a requesting node has detected to have left the membership since its last request.
- *Joined*: The number of nodes that a requesting node has discovered to have joined the membership since its last request.

- *NumNodes*: The number of nodes to which the requesting node sent its request.

Using these measured values, for each of the two adaptive membership protocols, a node calculates a *Churn Estimator* for each request when it finishes receiving the responses to that request, defined as follows:

- CE: The Churn Estimator, an estimate of the leaves and joins (churn) obtained by random sampling, given by:

$$CE = \frac{Left + Joined}{NumNodes} \tag{4}$$

The Churn Estimator is used by the adaptive membership protocols to adapt the Requesting Rate *RR*. The values of the Churn Estimator are determined using the Exponential Weighted Moving Average algorithm, described below.

5.5 Exponential Weighted Moving Average Algorithm

The adaptive membership protocols for iTrust uses the Exponential Weighted Moving Average (EWMA) algorithm to process a sequence of estimated values of the Churn Estimator *CE*, to smooth the estimated values and to reduce the noise inherent in the individual samples.

A requesting node issues requests (queries), collects responses, detects non-operational nodes, and discovers newly joined nodes. It then computes the estimated value *CE*, using the EWMA algorithm defined by:

$$\begin{aligned} s_1 &= v_1 \\ s_t &= c \text{ x } v_t + (1\text{-}c) \times s_{t\text{-}1} \quad \text{if } t > 1 \end{aligned} \tag{5}$$

where v_t is the measured value at time *t* and s_t is the smoothed value at time *t*. The constant *c* is a smoothing factor, $0 < c < 1$. Larger values of *c* place more emphasis on the most recently measured values and a faster response to changes. Smaller values of *c* provide more smoothing and less vulnerability to random fluctuations and noise.

The pseudocode for the EWMA algorithm is given in Figure 4.

```
EWMA(v, c, s)
1   if (t = 0) then s ← v
2   else s ← c × v + ( 1 – c ) × s
3   return s
```

Figure 4. Pseudocode for the EWMA algorithm

In our experiments, we used *c* = 0.7. However, iTrust offers the user the option to choose a value of *c* for the user's particular network environment. Different users, who operate in different network environments with different objectives, may choose different values of *c*, which iTrust allows.

5.6 Experimental Method

To evaluate and compare the four membership protocols for iTrust, described in Sections 6-9, we performed experiments using an emulation of iTrust. In the emulation, we can control the Leaving Rate *LR* and the Joining Rate *JR*, which a real-world deployment would not allow us to do. Moreover, in the emulation, we can compare a node's current view of the membership with the actual membership.

The emulation of iTrust is based on our HTTP implementation of iTrust using the Apache Web server, running on Debian Linux 6.0 on an Intel Quad Core 3.4 GHz processor with 4 GB memory and 1 TB hard drive. In the emulation, we have multiple virtual hosts installed on a single Apache Web server, where each virtual host represents a node in the iTrust network. Each node has a separate SQLite database that resides on the Apache Web server, where it stores queries and resource information.

Before we start the emulation program, we set the value of *N*, the number of nodes in the initial membership. The program clears the node's resources and databases, and then adds all of the nodes to each node's view of the membership, so that each node has the complete initial membership. At each time step, nodes might join the membership, leave the membership, and make requests. Different nodes might have different views of the membership, and different nodes might make requests at different rates.

For each source node, iTrust creates metadata for a document that the node wishes to share, and distributes the metadata to *M* randomly chosen nodes in the membership set. Then, iTrust distributes requests to *R* randomly chosen nodes in the membership set. Finally, the program compares each node’s view of the membership against the actual membership. The program computes the four performance metrics at each time step.

6 Non-Adaptive Membership Protocol

The Non-Adaptive Membership Protocol implements the membership protocol, described in Section 4, which involves requesting nodes updating their local views of the membership, as other nodes join and leave the membership.

The pseudocode for the Non-Adaptive Membership Protocol is given in Figure 5. The inputs for the Non-Adaptive Membership Protocol are *N* and *RR*, where *N* is the number of nodes in the node's local view and *RR* is the node's requesting rate.

The Non-Adaptive Membership Protocol comprises an infinite loop, at the beginning of which *nextRequestTime* is set to the current *time* plus *timeunit*/*RR*, which is the time when the node sends its next request. As time passes, *time* is automatically incremented (not shown in the pseudocode); *timeunit* is the length of the time unit. The protocol waits (line 3) until the current *time* reaches the *nextRequestTime*.

```
NonAdaptive(N, RR)
1  while true do
2      nextRequestTime ← time + (timeunit/RR)
3      wait until (time = nextRequestTime)
4      R ← ⌈2×√N⌉
5      responses ← makeRequests(view, R)
6      responded ← 0
7      for ( j ← 0 to R) do
8          if (responses[j].noResponse) then
9              removeNode(view, responses[j].node)
10             N ← N – 1
11         else
12             responded ← responded + 1
13             for (k ← 0 to LastJ) do
14                 if (responses[j].recent[k]) then
15                     isNew ← addNode(view, responses[j].recentNode[k])
16                     if (isNew) then
17                         N ← N + 1
```

Figure 5: Non-Adaptive Membership Protocol

The protocol then sets the number *R* of nodes to which the node sends its request message to $\lceil 2\sqrt{N} \rceil$, where *N* is the number of nodes in the node's current local view. The node then sends its request to *R* nodes, and waits for responses from those nodes (line 5).

Next, the protocol iterates through the *responses* array (line 7). The protocol checks whether the node received a response from node *j*. If not, it removes the non-responsive node from the node's local view and then decrements the number *N* of nodes in that view. Otherwise the protocol increments *responded*.

The protocol then iterates (line 13) through the *responses*[*j*].*recent* array and the *responses*[*j*].*recentNode* array, which provides the identifiers of up to the *LastJ* recently joined nodes. It checks whether *responses*[*j*].*recent*[*k*] is true. If so, it invokes *addNode*() to add the recent node *responses*[*j*].*recentNode*[*k*] to its local view. The procedure *addNode*() returns a boolean *isNew* to indicate whether or not the recent node is already present in the node's local view. If the recent node is indeed new, the protocol increments the number *N* of nodes in the node's local view.

Control then returns to continue the iteration through the *recentNode* array (line 13) or the *responses* array (line 7). When the protocol finishes iterating through the *responses* array, it goes back to the beginning, and repeats these steps indefinitely.

6.1 Investigation of the Non-Adaptive Membership Protocol

We investigate the Non-Adaptive Membership Protocol, in particular, the number *LastJ* of newly joined nodes that a responding node may report to a requesting node, and its effect on *LND* (the proportion of leaving nodes that the requesting node has not detected) and *JND* (the proportion of newly joined nodes that the requesting node has not discovered).

In the Non-Adaptive Membership Protocol, a requesting node distributes its request to $R = \lceil 2\sqrt{N} \rceil$ nodes chosen at random from its local view. Initially, we required those *R* nodes to return their entire views to the requesting node, and the requesting node to update its local view accordingly. The problem with that approach is that the requesting node adds to its view non-operational nodes obtained from other nodes that haven't yet detected that those nodes are non-operational. Thus, the requesting node adds to its local view non-operational nodes, including nodes that it recently removed.

Several possible solutions to this problem exist. One solution is that, once a requesting node has obtained the views of the other nodes, it sends a “verify” message to confirm that those nodes are indeed operational. Such a solution consumes too much network bandwidth. An alternative, less costly solution is to require the $\lceil 2\sqrt{N} \rceil$ nodes to return, to the requesting node, their “most recently joined members,” rather than their entire views. We adopt the latter solution and investigate how *LastJ* affects *LND* and *JND*.

Consider a scenario where *N* = 1024 nodes with a high leaving rate (*LR* = 300), a high joining rate (*JR* = 300) and a low requesting rate (*RR* = 10). Figure 6 shows the graphs for *LND* and *JND* over time for this scenario. Increasing *LastJ* from *LastJ* = 1 in the left graph to *LastJ* = 2 in the middle graph results in a decrease in *JND* but an increase in *LND*. Increasing *LastJ* from *LastJ* = 2 in the middle graph to *LastJ* = 3 in the right graph results in a slight decrease in *JND* and little change in *LND*.

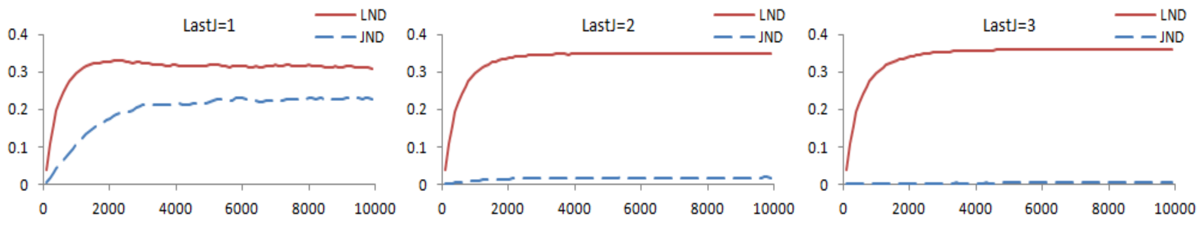

Figure 6: Graphs for the Non-Adaptive Membership Protocol, showing *LND* and *JND* over time for *LastJ* = 1, *LastJ* = 2 and *LastJ* = 3, where *N* = 1024 initially, *LR* = 300, *JR* = 300 and *RR* = 10

Thus, increasing *LastJ* definitely helps the requesting node to discover more joining nodes as it issues more requests. However, increasing *LastJ* also causes the requesting node to add back into its local view too many non-operational (leaving) nodes. Setting *LastJ* = 2 or *LastJ* = 3 increases *LND* with worse results than setting *LastJ* = 1. We conclude that *LastJ* = 1 is an appropriate value to use in our further experiments.

7 Retry R Membership Protocol

As we have seen, increasing *LastJ* does not help to detect leaving (non-operational) nodes. Thus, we investigate other methods to reduce the proportion *LND* of leaving nodes that a requesting node has not detected.

When a node distributes a request message to $\lceil 2\sqrt{N} \rceil$ nodes, it might not receive $\lceil 2\sqrt{N} \rceil$ responses for its request, because non-operational (leaving) nodes do not respond. Thus, we investigate a retry method that allows the requesting node to distribute its request to more than $\lceil 2\sqrt{N} \rceil$ nodes until it receives $\lceil 2\sqrt{N} \rceil$ responses. We call this protocol the Retry *R* Membership Protocol.

The pseudocode for the Retry *R* Membership Protocol is given in Figure 7. The inputs for this protocol are *N*, *TryMax* and *RR*, where *N* is the number of nodes in the node's current local view, *TryMax* is the number of times a node is allowed to try to obtain $\lceil 2\sqrt{N} \rceil$ responses and *RR* is the node's requesting rate.

The Retry *R* Membership Protocol comprises an infinite loop, at the beginning of which *nextRequestTime* is set to the current *time* plus the time *timeunit*/*RR* until the next request. The protocol waits (line 3) until the current *time* reaches the *nextRequestTime*.

The protocol then sets the number *R* of nodes to which the node sends its request message to $\lceil 2\sqrt{N} \rceil$. It sets the number *resRec* of responses received to 0 and the number *Try* of tries to 1, and then starts the while loop (line 7). The node sends its request message to *R* - *resRec* nodes, and waits for responses from those nodes (line 8).

Next, the protocol iterates through the *responses* array (line 10). It checks whether the node received a response from node *j*. If not, it removes the non-responsive node from the node's local view and then decrements the number *N* of nodes in that view. Otherwise, it increments *responded*.

The protocol then checks whether *j*'s response contains a recent node. If so, it invokes *addNode*() to add the recent node to the node's local view. The protocol then checks whether the recent node is indeed new and, if so, it increments the number *N* of nodes in the node's view. Control then returns to continue the iteration through the *responses* array (line 10).

```
RetryR(N, TryMax, RR)
while true do
    nextRequestTime ← time + (timeunit / RR)
    wait until (time = nextRequestTime)
    R ← ⌈2×√N⌉
    resRec ← 0
    Try ← 1
    while ((Try ≤ TryMax) and (resRec < R)) do
        responses ← makeRequests(view, R – resRec)
        responded ← 0
        for (j ← 0 to (R – resRec)) do
            if (responses[j].noResponse) then
                removeNode(view, responses[j].node)
                N ← N – 1
            else
                responded ← responded + 1
                if (responses[j].recent) then
                    isNew ← addNode(view,responses[j].recentNode)
                    if (isNew) then
                        N ← N + 1
        resRec ← resRec + responded
        Try ← Try + 1
```

Figure 7: Retry *R* Membership Protocol

After it has finished iterating through the *responses* array, the protocol increases *resRec* by *responded*, the number of responses in the *responses* array and then increments *Try*. Control then returns to the while loop (line 7) to determine whether *Try* is less than or equal to *TryMax* and *resRec* is less than *R*. If both of these conditions are satisfied, the protocol goes through the while loop again. Otherwise, the protocol goes back to the beginning, and repeats these steps indefinitely.

7.1 Investigation of the Retry R Membership Protocol

We investigate the Retry *R* Membership Protocol, in particular, the number *Try* of times that a requesting node tries to send its request message, in order to receive $\lceil 2\sqrt{N} \rceil$ responses. For example, *Try* = 1 means that a requesting node sends its request message to $\lceil 2\sqrt{N} \rceil$ nodes regardless of the number of responses that it receives to its request. *Try* = 2 means that a requesting node tries a second time and sends its request to *Left* nodes, where *Left* nodes didn't respond on the first Try and, similarly, for *Try* = 3. *Try* = ∞ means that a requesting node sends its request repeatedly until it receives responses from $\lceil 2\sqrt{N} \rceil$ nodes for that particular request.

Table 1 shows the membership accuracy, match probability, response time and message cost for *Try* = 1, 2, 3, ∞, where *N* = 1024 initially, *LastJ* = 1, *LR* = 300, *JR* = 300 and *RR* = 10 for the Retry *R* Membership Protocol.

Table 1: Retry *R* Membership Protocol with *Try* = 1, 2, 3, ∞

Try	1	2	3	∞
Membership Accuracy	0.5966	0.6821	0.6986	0.7048
Match Probability	0.9345	0.9817	0.9865	0.9864
Response Time	6.0	11.9274	17.6573	24.1682
Message Cost	3.8552	5.1538	5.4527	5.5598

In Table 1, we see that, as *Try* is increased from *Try* = 1 to *Try* = 2, both the membership accuracy and the match probability are greatly increased but, when *Try* is further increased to *Try* = 3, there is not much increase in either the membership accuracy or the match probability. We also see that both the response time and the message cost increase as *Try* is increased. To obtain a substantial increase in the membership accuracy and the match probability with a reasonable increase in the response time and the message cost, we use *Try* = 2 in our further experiments.

8 Adaptive RR Membership Protocol

The next membership protocol we consider uses the Churn Estimator *CE* to control the Requesting Rate *RR*. The Churn Estimator *CE* provides an estimate of the churn (leaves and joins) in the network; it is initialized to 0, and the *CE* values are averaged using the Exponential Weighted Moving Average (EWMA) algorithm as time progresses. We call this protocol the Adaptive *RR* Membership Protocol.

The pseudocode for the Adaptive *RR* Membership Protocol is given in Figure 8. The inputs for the Adaptive *RR* Membership Protocol are *N*, *RR*, *RRMin*, *RRMax* and *c*. Here *N* is the number of nodes in the node's current local view, and *RR* is the node's initial requesting rate.

```
AdaptiveRR(N, RR, RRMin, RRMax, c)
1  CE ← 0
2  while true do
3      nextRequestTime ← time + (timeunit/RR)
4      wait until (time = nextRequestTime)
5      Left ← 0
6      Joined ← 0
7      R ← $\lceil 2\times\sqrt{N}\rceil$
8      responses ← makeRequests(view, R)
9      for (j ← 0 to R) do
10         if (responses[j].noResponse) then
11             removeNode(view, responses[j].node)
12             N ← N - 1
13             Left ← Left + 1
14         else
15             if (responses[j].recent) then
16                 isNew ← addNode(view, responses[j].recentNode)
17                 if (isNew) then
18                     N ← N + 1
19                     Joined ← Joined + 1
20     currentCE ← (Left + Joined) / R
21     CE ← EWMA(CE, currentCE, c)
22     if CE > RRMin / RRMax then
23         RR ← RRMax × CE
24     else
25         RR ← RRMin
```

Figure 8: Adaptive *RR* Membership Protocol

The Adaptive *RR* Membership Protocol comprises an infinite loop, at the beginning of which *nextRequestTime* is set to the current *time* plus *timeunit*/*RR*, until the next request. The protocol waits (line 4) until the current *time* reaches the *nextRequestTime*.

The protocol initializes the variables *Left* and *Joined* (which count the changes in the node's view) to 0, and sets the number *R* of nodes to which the node sends its request message to $\lceil 2\sqrt{N}\rceil$. The node then sends its request to *R* nodes, and waits for their responses (line 8).

Next, the protocol iterates through the *responses* array (line 9). It checks whether the node received a response from node *j*. If not, it removes the non-responsive node from the node's local view and then decrements the number *N* of nodes in that view and increments *Left*, the number of nodes that have left. Otherwise, the protocol checks whether *j*'s response contains a recent node. If so, the protocol invokes *addNode*() to add the recent node to the node's local view. The protocol then checks whether the recent node is indeed new and, if so, it increments the number *N* of nodes in the node's local view and *Joined*, the number of nodes that have recently joined. Control then returns to continue the iteration through the *responses* array (line 9).

After processing the *responses* array, the protocol calculates *currentCE* (line 20) and then applies the EWMA algorithm to obtain the smoothed value of the Churn Estimator *CE* (line 21). The protocol then calculates the value of the Requesting Rate *RR* for the next time unit, corresponding to the smoothed value of the Churn Estimator *CE*. It then goes back to the beginning of the loop and repeats these steps indefinitely.

8.1 Investigation of the Retry R Membership Protocol

Figure 9 shows the graphs for the Leaves Not Detected (*LND*) and the Joins Not Discovered (*JND*) over time for the Non-Adaptive (left graph), the Retry *R* (middle graph) and the Adaptive *RR* (right graph) Membership Protocols. Here *N* = 1024 initially, *LastJ* = 1, *LR* = 300, *JR* = 300 and *c* = 0.7. For the Non-Adaptive and the Retry *R* Membership Protocols, *RR* = 10. For the Adaptive *RR* Membership Protocol, *RRMin* = 1, *RRMax* = 100 and *RR* = 10 initially.

In the middle graph of Figure 9, we see that, for the Retry *R* Membership Protocol, *LND* decreases to about 0.26 and *JND* decreases to about 0.12. In the right graph, we see that for the Adaptive *RR* Membership Protocol, *LND* greatly decreases to about 0.14 and *JND* decreases to almost 0. Thus, from these graphs, we see that increasing *RR* is more effective than increasing the number of tries, in decreasing both *LND* and *JND*.

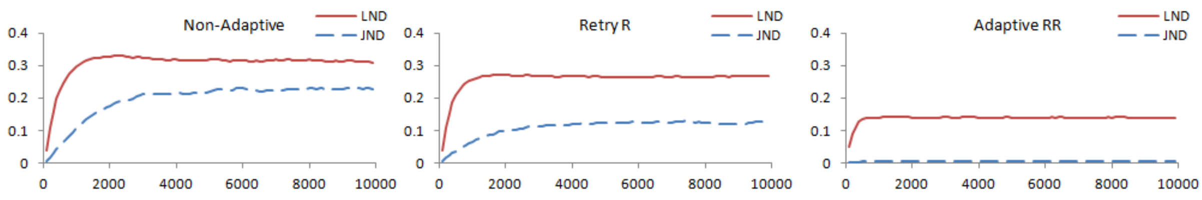

Figure 9: Graphs for the Non-Adaptive, the Retry *R* and the Adaptive *RR* Membership Protocols, showing Leaves Not Detected (*LND*) and Joins Not Discovered (*JND*), over time, where *N* = 1024 initially, *LastJ* = 1, *LR* = 300 and *JR* = 300

Table 2 presents the membership accuracy, match probability, response time and message cost for the Non-Adaptive, the Retry *R* and the Adaptive *RR* Membership Protocols.

In Table 2, we see that the Adaptive *RR* Membership Protocol has the highest membership accuracy, whereas the Non-Adaptive and the Retry *R* Membership Protocols have a much lower membership accuracy. We also

Table 2: Non-Adaptive vs. Retry *R* with *Try* = 2 vs. Adaptive *RR* with *Try* = 1 and *RRMax* = 100

	Non-Adaptive	Retry *R*	Adaptive
Membership Accuracy	0.5966	0.6821	0.8581
Match Probability	0.9345	0.9817	0.9728
Response Time	6.0	11.9274	6.0
Message Cost	3.8552	5.1538	12.5690

notice that the Retry *R* Membership Protocol achieves the highest match probability, whereas the Non-Adaptive Membership Protocol has the lowest match probability. Both the Non-Adaptive and the Adaptive *RR* Membership Protocols have a low response time, whereas the Retry *R* Membership Protocol has a much higher response time. The Adaptive *RR* Membership Protocol has a message cost that is about 3 times that of the Non-Adaptive Membership Protocol and about 2.5 times that of the Retry *R* Membership Protocol.

Thus, the Adaptive *RR* Membership Protocol has the highest membership accuracy, which is achieved by increasing the Requesting Rate *RR*, resulting in a high message cost. In contrast, the Retry *R* Membership Protocol has the highest match probability with a lower message cost but it also has a lower membership accuracy and a higher response time.

Therefore, we continue our investigations into an adaptive membership protocol that is intermediate between the Retry *R* Membership Protocol and the Adaptive *RR* Membership Protocol, in order to reduce the Requesting Rate *RR* and thus the message cost.

9 Combined Adaptive Membership Protocol

Now, we consider a membership protocol that not only adapts a node's Requesting Rate *RR* based on the Churn Estimator *CE* but also tries a second time (*Try* = 2) to obtain $\lceil 2\sqrt{N} \rceil$ responses to a node's request. We call this protocol the Combined Adaptive Membership Protocol.

The pseudocode for the Combined Adaptive Membership Protocol is given in Figure 9. The inputs for the Combined Adaptive Membership Protocol are *N*, *RR*, *RRMin*, *RRMax* and *c*. Again, *N* is the number of nodes in the node's current view, and *RR* is the node's current requesting rate.

The Combined Adaptive Membership Protocol comprises an infinite loop, in which *nextRequestTime* is first set to the current *time* plus *timeunit*/*RR*, until the next request. The protocol waits (line 4) until the current *time* reaches the *nextRequestTime*.

The protocol initializes the variables *Left* and *Joined* to 0, and sets the number *R* of nodes to which the node sends its request message to $\lceil 2\sqrt{N} \rceil$. It also initializes the number *resRec* of nodes from which it received responses to 0, and initializes the variable *Try* to 1.

The loop commencing at line 10 is potentially executed twice, but only once if the number of responses received (*resRec*) in the first try is equal to the number *R* of nodes to which the node sent its request message. Within the loop, the node sends its request message to *R* - *resRec* nodes, and waits for responses from those nodes (line 11).

The protocol then iterates through the *responses* array (line 13). It checks whether the node received a response from node *j*. If not, the protocol removes the non-responsive node from the node's view and then decrements the number *N* of nodes in its view and increments *Left*, the number of nodes that have left. Otherwise, the protocol increments *responded*. Then, it checks whether *j*'s response contains a recent node. If so, the protocol invokes *addNode*() to add the recent node to the node's view. The protocol then checks whether the recent node is indeed new and, if so, it increments the number *N* of nodes in the node's view and *Joined*, the number of newly joined nodes in that view. The protocol increases *resRec* by *responded*, the number of nodes that responded in this try, and then increments *Try*. Control then returns to the while loop (line 10) to determine whether *Try* is less than or equal to 2 and *resRec* is less than *R*. If both of these conditions are satisfied, the protocol goes through the while loop again.

After the protocol has completed the while loop, it calculates *currentCE* (line 27) using the values of *Left*, *Joined* and *resRec* it obtained in the loop, and then invokes the EWMA algorithm to calculate the smoothed value of the Churn Estimator *CE* (line 28). The protocol then calculates the value of the Requesting Rate *RR* for the next time unit, corresponding to the smoothed value of the Churn Estimator *CE*. The protocol then goes back to the beginning of the loop and repeats these steps indefinitely.

```
CombinedAdaptive(N, RR, RRMin, RRMax, c)
1  CE ← 0
2  while true do
3      nextRequestTime ← time + (timeunit/RR)
4      wait until (time = nextRequestTime)
5      Left ← 0
6      Joined ← 0
7      R ← ⌈2×√N⌉
8      resRec ← 0
9      Try ← 1
10     while ((Try ≤ 2) and (resRec < R)) do
11         responses ← makeRequests(view, R-resRec)
12         responded ← 0
13         for (j ← 0 to (R – resRec)) do
14             if (responses[j].noResponse) then
15                 removeNode(view, responses[j].node)
16                 N ← N – 1
17                 Left ← Left + 1
18             else
19                 responded ← responded + 1
20                 if (responses[j].recent) then
21                     isNew ← addNode(view,responses[j].recentNode)
22                     if (isNew) then
23                         N ← N + 1
24                         Joined ← Joined + 1
25         resRec ← resRec + Responded
26         Try ← Try + 1
27     currentCE ← (Left + Joined) / (R + R – resRec)
28     CE ← EWMA(CE, currentCE, c)
29     if CE > RRMin / RRMax then
30         RR ← RRMax × CE
31     else
32         RR ← RRMin
```

Figure 10: Pseudocode for the Combined Adaptive Membership Protocol

Note that, with *Try* = 2, a requesting node sends its request to more nodes to try to obtain $\lceil 2\sqrt{N} \rceil$ responses to its request. Thus, the Combined Adaptive Membership Protocol does not need to increase the Requesting Rate *RR* as much as does the Adaptive *RR* Membership Protocol. Consequently, the Combined Adaptive Membership Protocol realizes some savings in the message cost, compared to the Adaptive *RR* Membership Protocol.

9.1 Investigation of the Combined Adaptive Membership Protocol

We investigate the Combined Adaptive Membership Protocol, in particular the values of the Maximum Requesting Rate *RRMax*.

Table 3 shows the values of the membership accuracy, match probability, response time and message cost for the Combined Adaptive Membership Protocol for *RRMax* = 100, 50 and 30.

Table 3: Combined Adaptive Membership Protocol with *RRMax* = 100, 50 and 30

RRMax	100	50	30
Membership Accuracy	0.8663	0.8198	0.7579
Match Probability	0.9843	0.9836	0.9822
Response Time	11.9874	11.9883	11.9885
Message Cost	13.4939	9.3156	6.9104

We see that, as *RRMax* is decreased from *RRMax* = 100 to *RRMax* = 50, the membership accuracy decreases and the match probability slightly decreases. When *RRMax* is further decreased to *RRMax* = 30, again the membership accuracy decreases and the match probability slightly decreases, but still remains quite good.

We also see that, as *RRMax* is decreased, the response time remains the same and the message cost decreases substantially. The message cost for *RRMax* = 100 is nearly twice that for *RRMax* = 30. To keep the message cost lower while obtaining good membership accuracy, we chose *RRMax* = 50 for our further experiments.

9.2 Comparison of the Retry R, the Adaptive RR and the Combined Adaptive Membership Protocols

Table 4 shows the membership accuracy, match probability, response time and message cost for the Retry *R*, the Adaptive *RR* and the Combined Adaptive Membership Protocols. Here, *N* = 1024 initially, *LastJ* = 1, *LR* = 300, *JR* = 300 and *c* = 0.7. For the Retry *R* Membership Protocol, *Try* = 2 and *RR* = 10. For the Adaptive *RR* Membership Protocol, *Try* = 1, *RRMin* = 1, *RRMax* = 100 and *RR* = 10 initially. For the Combined Adaptive Membership Protocol, *Try* = 2, *RRMin* = 1, *RRMax* = 50 and *RR* = 10 initially.

**Table 4: Retry *R* with *Try* = 2 vs.
Adaptive *RR* with *Try* = 1 and *RRMax* = 100 vs.
Combined Adaptive with *Try* = 2 and *RRMax* = 50**

	Retry *R*	Adaptive *RR*	Combined Adaptive
Membership Accuracy	0.6821	0.8581	0.8198
Match Probability	0.9817	0.9728	0.9836
Response Time	11.9274	6.0	11.9883
Message Cost	5.1538	12.5690	9.3156

From Table 4, we see that the membership accuracy of the Combined Adaptive Membership Protocol is 0.8198, which is much better than that of the Retry *R* Membership Protocol, but worse than that of the Adaptive *RR* Membership Protocol. We also see that the match probability of the Combined Adaptive Membership Protocol is 0.9836, which is good and better than that of the Adaptive *RR* Membership Protocol and slightly better than that of the Retry *R* Membership Protocol. We see further that the response time of the Combined Adaptive Membership Protocol is about the same as that of the Retry *R* Membership Protocol, which is about double that of the Adaptive *RR* Membership Protocol. We also see that the message cost of the Combined Adaptive Membership Protocol lies between that of the Retry *R* Membership Protocol and that of the Adaptive *RR* Membership Protocol, and is about three-fourths that of the Adaptive *RR* Membership Protocol.

In summary, the Combined Adaptive Membership Protocol balances the message cost against the membership accuracy. The message cost of the Combined Adaptive Membership Protocol is less than that of the Adaptive *RR* Membership Protocol and also the membership accuracy of the Combined Adaptive Membership Protocol is greater than that of the Retry *R* Membership Protocol.

10 Extended Scenario

Now we investigate the effectiveness of the Combined Adaptive Membership Protocol to see how well it handles various combinations of low and high Leaving Rate *LR* and low and high Joining Rate *JR*. In particular, we consider an extended scenario that comprises the following five scenarios:

- Scenario 1: *LR* = 10, *JR* = 10 for time 0 to 3000
- Scenario 2: *LR* = 300, *JR* = 300 for time 3000 to 6000

- Scenario 3: *LR* = 0, *JR* = 300 for time 6000 to 9000
- Scenario 4: *LR* = 300, *JR* = 0 for time 9000 to 12000
- Scenario 5: *LR* = 0, *JR* = 0 for time 12000 to 15000.

For all five scenarios, we set *LastJ* = 1 and *c* = 0.7. Initially, there are *N* = 1024 nodes in the membership, and each node's view is the entire membership. As time progresses, each member changes its local view. The number *M* of nodes to which the metadata are distributed and the number *R* of nodes to which the requests are distributed are both set to $\lceil 2\sqrt{N} \rceil$, where *N* is the number of nodes in the node's current view at a given time step.

We compare the effectiveness of the Combined Adaptive Membership Protocol and the Non-Adaptive Membership Protocol by considering the extended scenario that comprises these five scenarios.

10.1 Non-Adaptive Membership Protocol

Figure 11 shows the graphs of the Leaves Not Detected *LND*, Joins Not Discovered *JND*, Membership Accuracy *MA* and Match Probability *MP* for the Non-Adaptive Membership Protocol. Here, *LastJ* = 1 and the Requesting Rate *RR* = 10 for all five scenarios.

In the first scenario of Figure 11, the Leaving Rate *LR*, Joining Rate *JR* and Requesting Rate *RR* are low and the same (*LR* = *JR* = *RR* = 10). The values of *LND* and *JND* remain low, because a node detects non-operational (leaving) nodes and discovers newly joining nodes within a short time interval. The Membership Accuracy *MA* remains high at about 0.9873 throughout the first scenario. The Match Probability *MP* is generally higher than the value 0.9817 obtained from the hypergeometric formula [15].

In the second scenario, the values of the Leaving Rate *LR* and the Joining Rate *JR* are much higher than the value of the Requesting Rate *RR* (*LR* = *JR* = 300 and *RR* = 10). The values of *LND* and *JND* increase, because a node can't detect enough non-operational (leaving) nodes and can't discover enough newly joined nodes within a short time interval. The Membership Accuracy *MA* dramatically decreases to about 0.5852. Moreover, the Match Probability *MP* is quite variable, decreasing to about 0.85 and then increasing to about 0.9350.

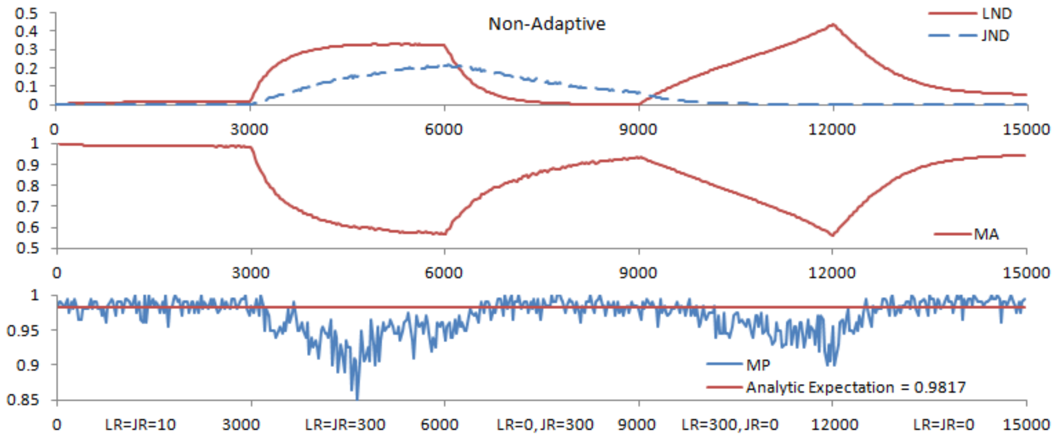

Figure 11: Graphs of *LND*, *JND*, *MA* and *MP* for the Non-Adaptive Membership Protocol where *LastJ* = 1 and *c* = 0.7

In the third scenario, the value of the Leaving Rate *LR* is low, the value of the Joining Rate *JR* is high, and the value of the Requesting Rate *RR* is low (*LR* = 0, *JR* = 300 and *RR* = 10). The values of *LND* decrease because *LR* drops to *LR* = 0. The values of *JND* remain high because *JR* remains high. The

Membership Accuracy *MA* is higher than that in the second scenario (because *LR* = 0), and slowly increases to about 0.9331 at the end of the third scenario. Similarly, the Match Probability *MP* slowly increases from about 0.9350 to about 0.9850.

In the fourth scenario, the value of the Leaving Rate *LR* is high, the value of the Joining Rate *JR* is low, and the value of the Requesting Rate *RR* is low (*LR* = 300, *JR* = 0 and *RR* = 10). The values of *LND* increase to about 0.4384. In addition, the Membership Accuracy *MA* steadily decreases to about 0.5610. The reason is that most of the nodes haven't yet discovered all of the newly joined nodes from the third scenario, but now more nodes are leaving the membership. The Match Probability *MP* fluctuates considerably, decreasing to about 0.9.

Lastly, in the fifth scenario, the values of both the Leaving Rate *LR* and the Joining Rate *JR* are low and the value of the Requesting Rate *RR* is also low (*LR* = *JR* = 0 and *RR* = 10). Thus, the Membership Accuracy *MA* slowly increases to about 0.9426. In addition, the Match Probability *MP* increases and remains high, hovering around the analytic expectation 0.9817.

10.2 Combined Adaptive Membership Protocol

Figure 12 shows the Leaves Not Detected *LND*, Joins Not Discovered *JND*, Requesting Rate *RR*, Membership Accuracy *MA* and Match Probability *MP* for the Combined Adaptive Membership Protocol. Here, *LastJ* = 1, *c* = 0.7, *Try* = 2, *RRMax* = 50 and *RR* = 10 initially.

In the first scenario, the values of the Leaving Rate *LR* and the Joining Rate *JR* are low (*LR* = *JR* = 10). Thus, the values of both *LND* and *JND* are low, because there are not many non-operational (leaving) nodes or newly joined nodes. The value of the Requesting Rate *RR* quickly decreases to 3.6764, in order to reduce the message cost. The Membership Accuracy *MA* remains high throughout the first scenario, and the Match Probability *MP* hovers around the analytic expectation 0.9817.

In the second scenario, the values of the Leaving Rate *LR* and the Joining Rate *JR* are high (*LR* = *JR* = 300), much higher than the values of the Requesting Rate *RR* (*RRMax* = 50). The values of *JND* and *LND* shown in

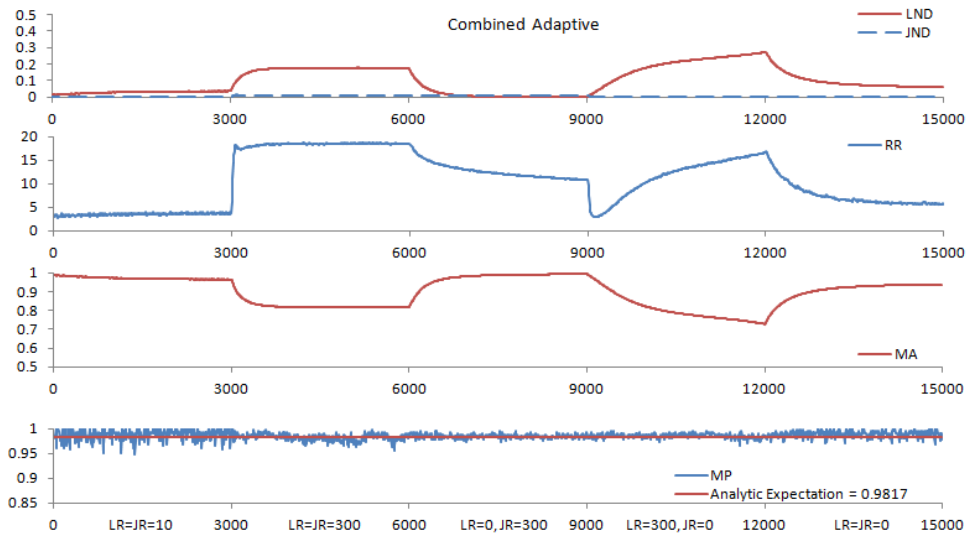

Figure 12: Graphs of *LND*, *JND*, *RR*, *MA* and *MP* for the Combined Adaptive Membership Protocol, where *RR* = 10, *LastJ* = 1, *c* = 0.7, *RRMax* = 50 and *Try* = 2

Figure 12 are much less than the corresponding values for the Non-Adaptive Membership Protocol shown in Figure 11. For the Combined Adaptive Membership Protocol, the value of the Requesting

Rate *RR* is increased to about 18.5161, which results in a Membership Accuracy *MA* of about 0.8251 compared to about 0.5852 for the Non-Adaptive Membership Protocol. Lastly, the Match Probability *MP* remains high throughout the second scenario, hovering around the analytic expectation 0.9817, whereas for the Non-Adaptive Membership Protocol it decreases to about 0.85.

In the third scenario, the value of the Leaving Rate *LR* is low and the value of the Joining Rate *JR* is high (*LR* = 0, *JR* = 300). Because *LR* is low, the values of *LND* remain close to 0. The values of *JND* also remain close to 0. The Combined Adaptive Membership Protocol adjusts the value of the Requesting Rate *RR* to about 12.6922. Joining nodes are discovered relatively quickly, and there are no new leaving nodes to detect because *LR* = 0. Thus, the Membership Accuracy *MA* increases from about 0.8251 to about 0.9739. The Match Probability *MP* still remains high, and hovers around the analytic expectation 0.9817.

In the fourth scenario, the value of the Leaving Rate *LR* is high and the value of the Joining Rate *JR* is low (*LR* = 300, *JR* = 0). Because the value of *LR* is high, the values of *LND* increase to about 0.2669. The values of *JND* remain low, because *JR* = 0. The Combined Adaptive Membership Protocol increases the Requesting Rate *RR* to about 16.5080, in order to detect leaving nodes more quickly. However, leaving nodes are still not detected quickly enough, so *LND* increases and the Membership Accuracy *MA* decreases to about 0.7307 at the end of the fourth scenario. Finally, the Match Probability *MP* remains high, and hovers around the analytic expectation 0.9817, in contrast to the Non-Adaptive Membership Protocol where the Match Probability *MP* fluctuates and decreases to about 0.9.

Lastly, the fifth scenario has a low value of the Leaving Rate *LR* and a low value of the Joining Rate *JR* (*LR* = *JR* = 0). The Combined Adaptive Membership Protocol decreases the value of the Requesting Rate *RR* to about 5.6349, in order to reduce the message cost. The Membership Accuracy *MA* increases to, and remains at, about 0.9375 during most of the fifth scenario. Moreover, the Match Probability *MP* remains high, and hovers around the analytic expectation 0.9817. Note that, with the Combined Adaptive Membership Protocol, the Match Probability *MP* remains high in all five scenarios despite substantial membership churn and substantial changes in the Leaving Rate *LR* and the Joining Rate *JR*.

Finally, we find the averages for each of the membership accuracy, match probability, response time and message cost over all five scenarios. Table 5 shows the overall values of these metrics for the Non-Adaptive, the Retry *R*, the Adaptive *RR* and the Combined Adaptive Membership Protocols, averaged over all five scenarios. As we see from the table, the Combined Adaptive Membership Protocol achieves a membership accuracy of 0.8982, which is quite good. Moreover, the Combined Adaptive Membership Protocol achieves the best match probability 0.9858 of all four protocols. The response time of the Combined Adaptive Membership Protocol is slightly more than that of the Retry *R* Membership Protocol, and is much more than that of the Non-Adaptive and Adaptive *RR* Membership Protocols. The message cost of the Combined Adaptive Membership

Table 5: Non-Adaptive vs. Retry *R* with *Try* = 2 vs. Adaptive *RR* with *Try* = 1 and *RRMax* = 100 vs. Combined Adaptive with *Try* = 2 and *RRMax* = 50

	Non-Adaptive	Retry *R*	Adaptive *RR*	Combined Adaptive
Membership Accuracy	0.8149	0.8542	0.9217	0.8982
Match Probability	0.9704	0.9841	0.9801	0.9858
Response Time	6.0	10.8262	6.0	11.0339
Message Cost	5.0490	5.7335	8.8284	6.3305

Protocol is less than that of the Adaptive *RR* Membership Protocol, and is slightly more than that of the Non-Adaptive and Retry *R* Membership Protocols, as Table 5 shows.

Overall, these experiments demonstrate that the Combined Adaptive Membership Protocol is effective in discovering newly joining nodes and in detecting non-operational (leaving) nodes. When the Joining Rate *JR* and the Leaving Rate *LR* are high, the Combined Adaptive Membership Protocol quickly increases the Requesting Rate *RR* to obtain a high membership accuracy. Moreover, when the Joining Rate *JR* and the Leaving Rate *LR* are low, the Combined Adaptive Membership Protocol decreases the Requesting Rate *RR*, in order to maintain a reasonable response time and a reasonable message cost, while still maintaining a reasonable membership accuracy and a high match probability. As a result, the Combined Adaptive Membership Protocol works well not only when the membership is subject to a lot of churn, but also when the membership is stable.

11 Conclusion

We have presented four membership protocols for the iTrust network, the Non-Adaptive, Retry, Adaptive and Combined Adaptive Membership Protocols. These membership protocols allow each member to maintain its own local view of the membership, and aim to keep that local view close to the actual membership. A node that receives a request sends a response, to the requesting node, that contains newly joined member(s) in its local view. If the keywords in the query match metadata that it holds, the node also sends the URL of the document.

A requesting node discovers newly joining nodes from the responses it receives to its requests. Likewise, a requesting node detects leaving (non-operational) nodes when it does not receive responses from those nodes before a timeout occurs, or when it receives an error code from TCP. Thus, the iTrust membership protocols exploit messages already required by the iTrust messaging protocol for distributing metadata and requests.

As our experiments show, for appropriate values of the parameters, the membership accuracy, the response time and the message cost are reasonable, and the match probability is high, particularly for the Combined Adaptive Membership Protocol. The Combined Adaptive Membership Protocol works well not only when the membership experiences a lot of churn but also when the membership is stable.

ACKNOWLEDGMENTS

This research was supported in part by the U.S. National Science Foundation grant number NSF CNS 10-16193.

REFERENCES

[1]. Badger, C.M., L.E. Moser, P.M. Melliar-Smith, I. Michel Lombera, Y.T. Chuang, *Declustering the iTrust search and retrieval network to increase trustworthiness*. Proceedings of the 8th International Conference on Web Information Systems and Technologies, Porto, Portugal, April 2012: p. 312-322.

[2]. Chandra, T.D., V. Hadzilacos, S. Toueg, B. Charron-Bost, *On the impossibility of group membership*. Proceedings of the 15th ACM Symposium on Principles of Distributed Computing, Philadelphia PA, May 1996: p. 322-330.

[3]. Chockler, G.V., I. Keidar, R. Vitenberg, *Group communication specifications: A comprehensive study*. ACM Computing Surveys, 2001. 33(4): p. 427-469.

[4]. Chuang, Y.T., P.M. Melliar-Smith, L.E. Moser, I. Michel Lombera, *Discovering joining nodes and detecting leaving nodes in the iTrust membership protocol*. Proceedings of the 2013 IAENG International Conference on Computer Science, Hong Kong, China, March 2013: p. 189-194.

[5]. Chuang,Y.T., I. Michel Lombera, L.E. Moser, P.M. Melliar-Smith, *Trustworthy distributed search and retrieval over the Internet.* Proceedings of the International Conference on Internet Computing, Las Vegas, NV, July 2011: p. 169-175.

[6]. Clarke, I., O. Sandberg, B. Wiley, T. Hong, *Freenet: A distributed anonymous information storage and retrieval system.* Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, July 2001: p. 46-66.

[7]. Cuenca-Acuna, F.M., C.Peery, R.P. Martin, T.D. Nguyen, *PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities*. Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Seattle, WA, June 2003: p. 236-246.

[8]. Freedman, M.J. and R. Morris, *Tarzan: A peer-to-peer anonymizing network layer*. Proceedings of the 9th ACM Conference on Computer and Communications Security, Scottsdale, AZ, November 2014: p. 193-206.

[9]. Ganesh, A., A.M. Kermarrec, L. Massoulie, *Peer-to-peer membership management for gossip-based protocols.* IEEE Transactions on Computers, February 2003. 52(2) : p. 139-149.

[10]. Gnutella, http://en.wikipedia.org/wiki/Gnutella (2000).

[11]. Gramoli, V., A.M. Kermarrec, E. Le Merrer, *Distributed churn measurement in arbitrary networks.* Proceedings of the 27th ACM Symposium on Principles of Distributed Computing, Toronto, Canada, August 2008: p. 431.

[12]. Leng, C., W.W. Terpstra, B. Kemme, W. Stannat, A.P. Buchmann, *Maintaining replicas in unstructured P2P systems.* Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies, Madrid, Spain, December 2008: p. 19.

[13]. Liu, H., X. Liu, W. Song, W. Wen, *An age-based membership protocol against strong churn in unstructured P2P networks.* Proceedings of the 2011 International Conference on Network Computing and Information Security, vol. 2, Guilin, China, May 2011: p. 195-200.

[14]. Lv, Q., P. Cao, E. Cohen, R. Li, S. Shenker, *Search and replication in unstructured peer-to-peer networks.* Proceedings of the 16th International Conference on Supercomputing, Baltimore, MD, June 2002: p. 84-95.

[15]. Melliar-Smith, P.M., L.E. Moser, I. Michel Lombera, Y.T. Chuang, *iTrust: Trustworthy information publication, search and retrieval*. Proceedings of the 13th International

Conference on Distributed Computing and Networking, LNCS 7129, Hong Kong, China, January 2012: p. 351-366.

[16]. Michel Lombera, I., Y.T. Chuang, P.M. Melliar-Smith, L.E. Moser, *Trustworthy distribution and retrieval of information over HTTP and the Internet*. Proceedings of the 3rd International Conference on the Evolving Internet, Luxembourg City, Luxembourg, June 2011: p. 7-13.

[17]. Mischke, J., and B. Stiller, *A methodology for the design of distributed search in P2P middleware*, IEEE Network, 2004. 18 (1): p. 30-37.

[18]. Peng, P., L.E. Moser, P.M. Melliar-Smith, Y.T. Chuang, I. Michel Lombera, *A distributed ranking algorithm for the iTrust information search and retrieval system*. Proceedings of the 9th International Conference on Web Information Systems and Technologies, Aachen, Germany, May 2013: p. 199-208.

[19]. Pruteanu, A., V. Iyer, S. Dulman, *ChurnDetect: A gossip-based churn estimator for large-scale dynamic networks*. Proceedings of the Euro-Par 2011 Conference, LNCS 7155, Bordeaux, France, August 2011: p. 289-301.

[20]. Reiter, M.K.. and A.V. Rubin, *Crowds: Anonymity for Web transactions*. ACM Transactions on Information and System Security, November 1998. 2 (1): p. 66-92.

[21]. Richardson, S. and I.J. Cox, *Estimating global statistics for unstructured P2P search in the presence of adversarial peers.* Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, Gold Coast, Queensland, Australia, July 2014: p. 203-212.

[22]. Schiper, A. and S. Toueg, *From set membership to group membership: A separation of concerns*. IEEE Transactions on Dependable and Secure Computing, 2006. 3(1): p. 2-12.

[23]. Terpstra, W.W., J. Kangasharju, C. Leng, A.P. Buchmann, *Bubblestorm: Resilient, probabilistic, and exhaustive peer-to-peer search*. Proceedings of the ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications, Kyoto, Japan, August 2007: p. 49-60.

[24]. Voulgaris, S., D. Gavidia, M. Van Steen, *CYCLON: Inexpensive membership management for unstructured P2P overlays.* Journal of Network and Systems Management, June 2005. 13(2): p. 197-217.

[25]. Zage, D., C. Livadas, E.M. Schooler, *A network-aware distributed membership protocol for collaborative defense*. Proceedings of the International Conference on Computational Science and Engineering, vol. 4, 2009: p. 1123-1130.