A STUDY OF MEMBERSHIP MANAGEMENT PROTOCOLS FOR GROUPS IN WIRELESS SENSOR NETWORKS

BY

KANWAR HARKIRAT SINGH
B.TECH., Punjab Technical University, 2001

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

ABSTRACT

Over the past few years, Wireless Sensor Networks have attracted quite a lot of research interest and have quickly evolved from the initial phases to the stage where practically deployable applications for sensor networks are being written and being put to use. As applications get more complex and more generic, the old paradigm of writing the whole application from scratch becomes cumbersome; rather, a layered abstractions-based architecture seems to be evolving. Keeping in tune with this layered or stovepiped architecture, we envision the need of a membership management substrate for wireless sensor networks. In this thesis, we develop such a membership substrate for wireless sensor networks, evaluate different membership management strategies and *match* membership management protocols with various types of membership requirements that wireless sensor network applications may have.

Traditionally, membership management algorithms use some variation of the heartbeating technique, but we prefer to consider random pinging techniques over heartbeating since heartbeat protocols are not scalable to large sized groups. In this thesis, we evaluate a total of 195 combinations for matching application requirements to membership protocols, eliminating the bad combinations among these, and then provide enough analysis and experimental data for the other combinations so that an application designer can make an informed choice.

ੴ

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere regards and gratitude for my advisor, Dr. Indranil Gupta. My experience at UIUC would not have been as rewarding and stimulating without his guidance. His original ideas, constant encouragement, patient support and belief in me (much more than I had in myself) have been instrumental in the completion of this work. I thank him for all the pains he has taken to ensure that this thesis reaches a just conclusion and he shall forever be a source of inspiration for me.

Thanks are also due to my friends and colleagues at UIUC, particularly Nikhil and Mithun, for all their suggestions and help. I'd also like to thank my dear buddy, Rahul, without whose interference this thesis would definitely have been finished faster, but would have been a lot less interesting. I am also grateful to all my housemates, including Kunal, Manmeet and Tarun for tolerating me for the last two years.

Lastly, and not in the least, none of this would ever have been possible were it not for the love, support and encouragement of my mom, my dad and my siblings, Ravi and Tashi. They were always there in all the ups and downs of my life and it's because of them that I have been able to achieve anything worth achieving.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

UAR	- Uniformly At Random
CLU	- Clustered
SPA	- Spatial
LOCn	- Local within *n* hops
KLOC	- Nearest *K* local neighbors
HIER	- Hierarchical
UUuar	- UAR Geographic distribution, UAR Membership graph, UAR failure detection and Broadcast-TTL1 update dissemination
CUuar	- CLU Geographic distribution, UAR Membership graph, UAR failure detection and Broadcast-TTL1 update dissemination
UUspa	- UAR Geographic distribution, UAR Membership graph, SPA failure detection and Broadcast-TTL1 update dissemination
UUloc*n*	- UAR Geographic distribution, UAR Membership graph, LOC*n* failure detection and Broadcast-TTL1 update dissemination
CHuar	- CLU Geographic distribution, HIER Membership graph, UAR failure detection and Broadcast-TTL1 update dissemination
USspa	- UAR Geographic distribution, SPA Membership graph, SPA failure detection and Broadcast-TTL1 update dissemination
CSspa	- CLU Geographic distribution, SPA Membership graph, SPA failure detection and Broadcast-TTL1 update dissemination

1 INTRODUCTION

Recent advances in fabrication technology have allowed the integration of sensors, processors and wireless communication capabilities into small embedded systems called sensor nodes. This has led to active research in large-scale distributed systems composed of wireless sensor networks. The need for unobtrusive and remote monitoring is the main motivation for deploying a sensing and communication network (sensor network) consisting of a large number of these battery-powered nodes. The vision for this kind of technology is to have 'anywhere, anytime' monitoring for indoor and outdoor areas, buildings and structures, and interactive spaces.

Wireless Sensor Networks are being developed for a wide range of civil and military applications, such as object tracking [12, 13, 17], infrastructure monitoring [14, 15, 18, 19], habitat sensing [16, 20], disaster relief and battlefield surveillance [27]. We would like a wireless sensor network to consist of hundreds or thousands of tiny sensor nodes that communicate over wireless channels and perform distributed sensing and collaborative data processing.

As applications for wireless sensor networks grow in complexity, so do the services they provide and need. Surveillance applications such as those for vehicle tracking may need to identify a group of sensors that is geographically close to a particular sensor. A multi-resolution image processing application may want to know of all nodes that have the color 'red'. An aggregation application such as an environmental habitat monitoring application may want to know of a group of sensors that are currently

receiving light that is more than a specified threshold. Thus there would seem to be candidate applications that would require a membership protocol that provides the application with a list of currently active nodes that are part of a specified group.

As an example of an application that would require membership information of other sensors and nodes, consider an interactive space, such as a smart room. The devices in the room can be set up to identify people in the room based on the Active Badge Location System [1] and grant them access based on their access privileges. Furthermore, devices can also be configured to grant access only if one or more supervisors are present in the room. For this, the sensor network would have to keep track of people entering and leaving the room as well as to maintain a list of people currently present in the room. Any changes in the membership of the room (in effect, changes in the membership of the group that is allowed access to the devices) would need to be rapidly propagated to all the other nodes for them to be able to configure themselves to reflect the changes in the membership of the room.

Another application where it would be useful to have such membership information is an application that monitors and controls the temperature settings in a building. The application could monitor various rooms to keep track of people in those rooms and control the temperature of the building as well as of the individual rooms based on the number of people in each room and the amount of time each room has been occupied. Furthermore, to conserve energy and resources, the system could also control the temperature to be set in a gradient or a spatial manner. In such a case, the temperature of the rooms that are infrequently used and are some distance away from where most

people are at a certain point in time can be controlled as a function of the distance of the room from a specific reference point.

In their current version, sensor motes are constrained in terms of resources such as having limited power supplies, short radio ranges, limited on-board memory and limited on-board processing capabilities but we envision that with further technological advances, such limitations would be overcome and sensor networks would become commonplace in everyday life. Instead of being application specific, as most sensor networks are these days, future sensor networks would be generic and reprogrammable [33]. Already, work is being done on reprogramming sensor motes over the sensor network itself by using a virtual machine to run the applications rather than burning the application onto the mote [2]. Thus, in future, we expect more generic architectures to be available to the application developer who wants to develop an application for a sensor network. To that end, we propose a membership substrate for wireless sensor networks. At each participating mote, this substrate could provide a complete or partial list of currently active members of a specified group.

Welsh and Mainland in [26] have proposed programming sensor networks using abstract regions that are defined as "a family of spatial operators that capture local communication within regions of the network, which may be defined in terms of radio connectivity geographic location, or other properties of nodes." These abstract regions are deemed to make easier the work of application developers by abstracting the lower-level details. Our work, in comparison, is more geared towards the actual lower-level architecture required to maintain such abstract regions. Thus a middleware that provides

abstract regions to the application developers can use our membership substrate to maintain those abstract regions.

Our work is also somewhat similar to the middleware service proposed by Blum, et al. in [31] but differs in the implementation details and the fact that we explore and evaluate different failure detection techniques and update dissemination schemes as well as try to match wireless sensor network application requirements to membership protocols. They also tend to do 'Entity Management' within a local, bounded area near a leader whereas our protocols do not have any such restrictions.

1.1 Thesis Overview

In this thesis, we investigate membership management for wireless sensor networks. We take a look at some typical sensor network applications and how they differ from applications in a wireless ad-hoc network. We also look at different geographic distributions and membership models relevant to sensor networks.

We implement our membership protocol as a nesC [3] library so that sensor network application developers can use it to create distributed applications. nesC [3] is a component-oriented variant of C that is used by the TinyOS [4] operating system. The protocol is evaluated through simulations using TOSSIM [5]. We also simulate the behavior of different failure detection strategies for various geographical distributions and membership models using our own simulator developed in C and explore their tradeoffs with respect to network traffic and completeness in failure detection.

Overall, this thesis makes the following contributions. First, it presents an evaluation of different membership management schemes for wireless sensor network applications and attempts to match applications with varied membership requirements to schemes that match their requirements. Second, it makes recommendations to developers regarding the choices and tradeoffs involved in choosing between the various membership management protocols. Finally, it provides an implementation of the membership protocol as a library for application development.

The rest of this thesis is organized in the following manner. In Chapter 2, we give an overview of wireless sensor networks. Chapter 3 discusses various applications and the kind of membership requirements they might have. Chapter 4 gives an overview of various membership management and failure detection schemes and analyses them. Chapter 5 attempts to match differing application requirements to the membership protocols and also discusses the implementation details and the simulation results. We conclude the thesis in Chapter 6 along with recommendations to application developers and future work.

2 WIRELESS SENSOR NETWORKS

Wireless sensor networks have been the focus of considerable research interest during the past few years. Advances in MEMS technology along with the development of low-cost and low-power radio transmission circuits and processors have resulted in the emergence of such networks of sensors. The prowess of wireless sensor networks comes from the ability to deploy a large number of tiny sensor nodes over a geographic area. While the capability of each sensor node, by itself, is limited, the composition of hundreds of sensor nodes offers avenues for reliable, parallel and large-scale data acquisition.

Unlike traditional cellular communication, wireless sensor nodes, typically, do not communicate directly with a base station. Rather, they work in a peer-to-peer fashion. Thus data collected by a node is transmitted to the base station by routing it through the sensor network in a multi-hop fashion.

Applications for wireless sensor networks must take care of certain technical challenges that are unique to such sensor networks and are not found in other kinds of wireless networks:

- *Low cost*: Most wireless sensor network applications require the deployment of hundreds or thousands of sensor nodes. To be able to effectively and economically deploy such a large number of nodes, the cost of the components has to be minimized.

- *Low power consumption*: Sensor nodes are usually powered by batteries. Due to the large number of sensor nodes deployed or due to inaccessibility of the nodes themselves, it may not be feasible to replenish the batteries when they run out. So to maximize their lifetimes, sensor nodes must utilize minimal energy. Therefore, protocols and application for wireless sensor networks have to be aware of energy consumption and try to minimize both computation and transmission by the nodes.
- *Communication efficiency*: Sensor networks typically have low communication efficiency; therefore, the communication protocols must be kept as simple as possible.
- *Ad-hoc deployment*: In many cases, sensor nodes are deployed in a region with no infrastructure at all. In such a case, it is up to the nodes themselves to configure the network and identify its connectivity.
- *Unattended operation*: In most cases, wireless sensor networks, once deployed have no human intervention. Thus, such networks have to be able to reconfigure themselves to any changes in the network.

2.1 Wireless Sensor Networks vs. Wireless Ad-hoc Networks

Sensor networks are a new family of wireless networks and are significantly different from traditional wireless networks. In traditional wireless networks, the emphasis is on optimizing QoS and bandwidth efficiency. However, in wireless sensor networks, energy consumption takes on a higher degree of importance. Although both

wireless sensor networks and wireless ad-hoc networks seem similar, there are a few fundamental differences between them. These differences are outlined below:

- A wireless sensor network is typically has a large number of sensor nodes that can be in the range of thousands of nodes. On the other hand, a wireless ad-hoc network has a much lesser number of nodes.
- Sensor nodes in a wireless sensor network are much more highly constrained in terms of memory, computational capabilities and power consumption as compared to nodes in a wireless ad-hoc network.
- Due to their large number, nodes in a sensor network may be more densely packed that those in a wireless ad-hoc network. This results in a degree of redundancy in a wireless sensor network
- Sensor nodes in a wireless sensor networks are more prone to failure than nodes in a wireless ad-hoc network. Such failures can be due to environmental conditions, battery exhaustion or other factors. In many cases, it is not possible for sensor nodes to recover after having failed.

Thus in many cases, protocols and applications that work for wireless ad-hoc networks may not work for wireless sensor networks and new protocols and applications have to be developed from scratch keeping in mind the various constraints of sensor networks.

2.2 Wireless Sensor Network Applications

The recent interest in wireless sensors has given rise to a number of applications based on wireless sensor networks. Modern sensor nodes can have various different types

of sensors on them such as acoustic, seismic, thermal, visual and infrared sensors that are able to monitor a wide variety of phenomena. Based on these sensors, numerous applications have been developed, some of which are described below.

One of the major fields of application for wireless sensor networks has been for military applications. Wireless sensor networks, by virtue of their rapid deployment, self-organization and fault tolerance characteristics, are ideal for military surveillance, targeting and reconnaissance operations. One of the military applications of wireless sensor networks is battlefield surveillance in which critical paths, routes and other terrain can be covered and monitored for enemy forces. Monitoring of friendly forces, ammunition and supplies as well as damage assessment are some of the other uses to which such wireless sensor networks can be put to.

Wireless sensor networks have been put to considerable use in studying and monitoring environmental phenomena. One of the most famous environmental applications of such a network was the GDI system [16] deployed in August 2002 by researchers from UCB/Intel Research Laboratory. They deployed a mote-based sensor network on Great Duck Island, Maine, to monitor the behavior of the storm petrel. Other environmental applications include flood and forest fire detection systems [37], seismic monitors, pollution and pesticide monitoring systems.

Another area of wireless sensor network applications is structural monitoring. Sensor nodes installed in building and bridges and other structures monitor vibrations and other displacement of the structures [15]. These sensors can detect cracks or other anomalies in the structure as soon as they occur and prove to be very helpful to perform the necessary repairs. Usually hidden cracks appear in buildings after minor earthquakes

or tremors but are not easily detected. Sensor network systems can detect such small structural variations and can also be used to monitor the effects of various natural phenomena on the structure.

Traffic monitoring is another field in which wireless sensor networks have been useful. Installed along the major highways, the wireless sensor network gathers data about vehicle speeds and lane occupancy. Such data makes it possible for the traffic monitoring system to calculate average speeds as well as travel time and the congestion levels on the highways.

Some of the health related applications for sensor networks include providing interfaces for the disabled, integrated patient monitoring, diagnostics and drug administration in hospitals, telemonitoring of human physiological data and tracking and monitoring doctors and patients inside a hospital [34, 35]

As technology advances further, sensor nodes can be embedded into household appliances such as microwaves, refrigerators and VCRs. These sensor nodes could interact with each other and with the external network through the internet and would allow end users to manage the devices locally as well as remotely.

2.3 Current Trends in Sensor Networks

The early versions of wireless sensor network platforms are now providing feedback for defining the new generation of hardware and software that better meets the demands of wireless sensor networks. At the mote level, low bandwidth needs have allowed radio engineers to trade bit-rate for power consumption in the newer radios.

Similarly, software development efforts have moved beyond the stage of building every application from scratch. There are now libraries for generic functionalities like routing, tracking, synchronization and for querying the network that help in accelerating the development of new sensor network applications.

For almost all classes of sensor network hardware, the performance is increasing for any given power budget. For example, the current MICA2 [28] mote has approximately eight times the memory and the communication bandwidth compared to the Rene mote while having the same power and cost. Part of the performance increase has been fueled by the development of newer Complementary Metal Oxide Semiconductor (CMOS) radios. These have been specifically designed for battery powered embedded devices such as the sensor motes. So where earlier radios on the MICA mote could transmit up to 52 kbaud per second (kbps), the MICA2 [28] and the MICA2DOT [30] motes can transmit up to 78.4 kbps and the newer MICAz [29] mote can transmit at up to 250 kbps.

As in the case of radio designs, micro controllers are being developed to cater to low power embedded devices. These micro controller designs remain focused on energy per instruction and peak power consumption but are also factoring in other small design concerns. In particular, newer micro controllers are being designed to reduce the wake up time for the CPU from the current order of milliseconds to an order of microseconds. Since the CPU wakes up several times per second, this translates into enormous power savings.

On the software front, the development of the TinyOS [4] software architecture has provided a unified platform for developing wireless sensor network applications. It

does so by providing abstractions tailored to sensor networks. Thus hardware developers are able to support hundreds of applications by writing a single software driver. Application developers also have the flexibility to choose from various networking algorithms based on what they need. By having a suite of protocols that provide the same TinyOS [4] component interface, applications can switch from one protocol to another by just changing one line of code.

2.4 Other Related Work

Efforts are also on to miniaturize the hardware itself to the point of being a speck so that such sensor networks can be ubiquitously deployed. The 'Smart Dust' project [32, 36] aims to do just this by trying to fit the whole sensing, computing and communication hardware within a space of 1 cubic millimeter.

Sensor networks are exposed to and collect a huge amount of detailed and continuous data about their environment. Quite a few approaches have been proposed for accessing this data. The most basic approach is to have all the sensor nodes to send all their data to a central repository which is external to the sensor network. Since this creates a continuous stream of data towards the base station, another approach proposed flooding the sensor network with a specific query, as and when the data is required, and the appropriate sensors would send the data back to the base station. Further savings in network traffic were realized by aggregating data [21, 42, 41] within the sensor network itself before sending it to the base station. Directed Diffusion [21] and TAG [41] describe particular forms of in-network aggregation. Systems like TinyDB [51] and Cougar [50]

go a step further and provide a high-level SQL like query interface to extract data from the sensor network.

Due to the highly ad-hoc nature of wireless sensor networks, routing data within the network has been an issue and has been the topic of considerable research [43, 44, 21, 45, 46, 47, 48]. Due to the energy constraints of sensor nodes, conventional routing protocols, as those used in ad-hoc networks, are not very well suited to wireless sensor networks. Adaptive protocols such as SPIN [48, 49] protocols aim at disseminating information among the sensor nodes by using information descriptors for negotiation before transmitting data .These information descriptors are called meta-data and are used to eliminate the transmission of redundant data in the network. Rumor Routing [43] uses gossiping techniques to route information in the network. Energy Aware Routing [47] proposes using sub-optimal paths occasionally, so as to substantially increase the lifetime of the network as a whole.

2.5 Assumptions

In this thesis, we make these basic assumptions about the wireless sensor network:

- Each sensor node has a globally unique identification. This ID need not necessarily be assigned at fabrication or compile time and may be assigned at run time, as long as each sensor node is uniquely identifiable. The ID can also be based on any other criteria such as being based on its approximate geographical location [38, 39, 40].
- There are completely reliable communication links between two neighbors, if both nodes are non-faulty.

- The network is connected in the sense that there exists a path from each node to each other node in the network

3 MEMBERSHIP REQUIREMENTS OF SENSOR NETWORK APPLICATIONS

As advances are made in sensor network design and in developing more powerful and more capable hardware, the uses that wireless sensor networks can be put to are also increasing. Concomitant with increasing fields of application, the complexity of the applications that are being deployed on wireless sensor networks is also increasing. Whereas earlier applications were kept very simple, current applications are becoming more and more complex in nature. In the earlier stages of sensor networks, developers had to program everything from scratch, including all the packet transmission and handling routines, but now shared libraries of commonly used routines have been developed and the use of TinyOS [4] has helped in moving the development effort 'up the stack'.

As applications grow more distributed in nature, coordination [25] between processes on different processors becomes more critical. Also, the design of sensor network applications would often require the availability of a consistent view of the application state among the participating processes. Such views would be important, as they would simplify both the programming and the verification tasks.

This chapter provides examples of a few sample applications and how they would require membership information and the various geographic distributions and membership models that are applicable to wireless sensor network applications.

3.1 Geographic Distributions

Many wireless sensor network applications could be classified as having one of the two major types of geographic distribution, namely, either Uniformly At Random (UAR) or Clustered (CLU).

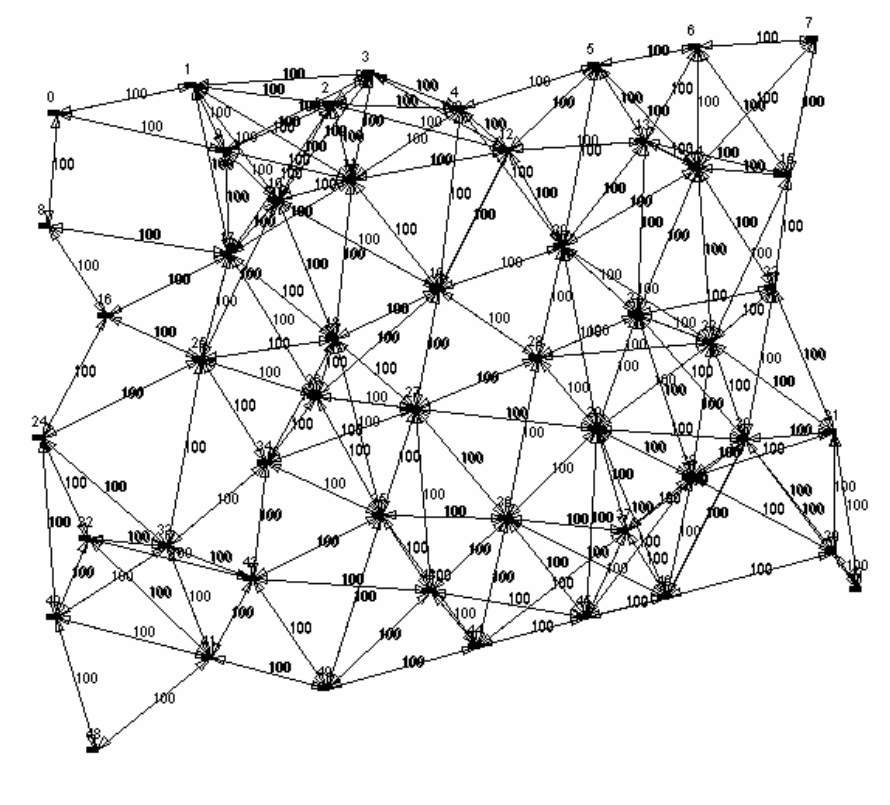


Figure 3.1: Uniformly at Random (UAR) geographic distribution

In a UAR geographic distribution, the sensor nodes are scattered in a uniform random pattern over a given area. Each node would be equally likely to be at a given position. Such a pattern could occur if sensors are uniformly scattered over a given geographic area, for example, from an airplane.

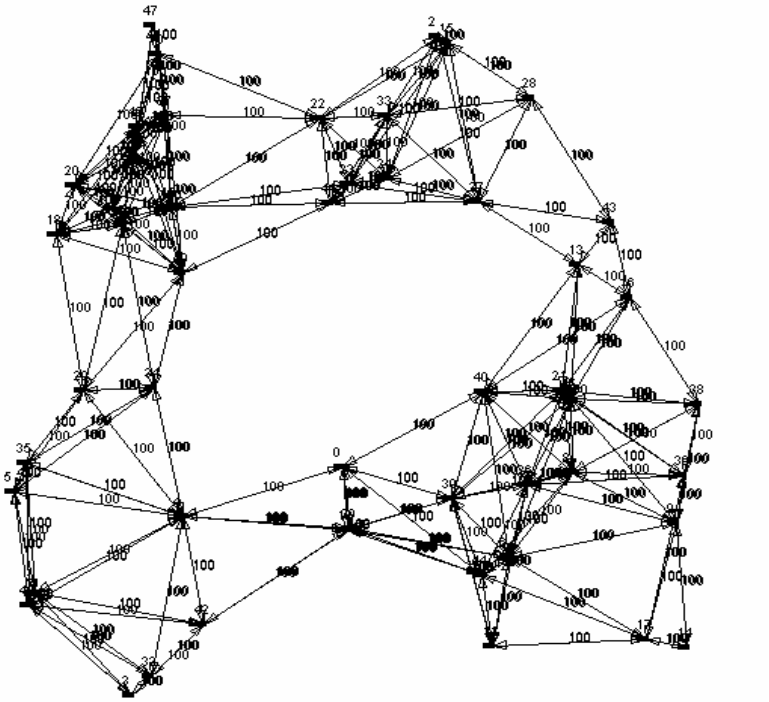


Figure 3.2: Clustered (CLU) geographic distribution

Sensors could also be organized in a Clustered geographic distribution. In such a distribution, sensors are generally placed in clusters with a given area having one or more clusters. In such a pattern, a sensor would have a higher probability of being part of a cluster rather than not. A clustered pattern could be deployed for applications wherein more sensors could be placed near event sources for more fine-grained sensing and for redundancy in sensing.

3.2 Membership Models

Different applications have differing requirements as to the membership graph they can construct and maintain. Some applications would want for each node to maintain a complete list of all the other active nodes in the system, whereas for other applications a partial membership list would suffice. Thus the most general classification of membership requirements of applications would be to maintain a complete list versus maintaining a partial list. This choice is very much application and hardware dependent, determined by processing capability, available power, available memory, length of deployment, etc.

The partial list could, furthermore, be constructed in a variety of ways. Five such methods of creating a partial list are enumerated below:

- *UAR (Uniformly at Random)*: The simplest way of creating a partial membership list is to select nodes uniformly at random from the set of member nodes. Thus, all member nodes in the system have an equal probability of being selected to be in a particular node's membership list.
- *LOCn (Local within n)*: Some applications might just be interested in maintaining membership information about their immediate vicinity. In such cases, the membership graph would only consist of nodes that are within *n* radio hops from the node. Thus a LOC1 graph would consist of nodes that are within 1 radio hop from the target node. The LOC*n* membership model is usually used when the application would want to save power.

- *KLOC (K-Local)*: In many cases it might happen that the LOC*n* graph may have a very limited number (or none at all) of other member nodes that satisfy the criteria of being within *n* hops of the target node. Hence, a modification of the LOC*n* scheme is to maintain a membership graph containing the nearest *K* neighbors.
- *SPA (Spatial)*: A fourth scheme is to create the membership graph in a spatial manner. Thus the probability that a node would be chosen to be in the membership list is equal to $d^{-\alpha}$ where 'd' is the distance (in hops) of the node from the target node and 'α' is a spatial probability constant. Thus, nodes that are closer to the target node are chosen with a higher probability compared to the nodes that are further away.
- *HIER (Hierarchical)*: Another method of constructing a partial membership list could be to create a hierarchical graph. Such a graph would be more relevant to an application in which the nodes are grouped into clusters and each cluster could have one or more leader nodes who would know of all the other leader nodes in the system. Each non-leader node in a cluster would know of all the other nodes in its cluster as well as the leaders of its cluster. This kind of a graph could also be extended to a UAR geographical distribution by creating clusters or regions at the application level and choosing leaders for each level. A hierarchical graph could also be created without any leader nodes. In such a case, each node would know of other nodes with the same cluster or region as itself and of a few randomly chosen nodes from the other clusters or regions.

3.3 Example Applications

We motivate the discussion and utility of a membership protocol for wireless sensor networks by describing a few applications and how they would use the membership protocol. Figure 3.3 shows a matrix of geographical distributions and membership graphs filled with sample applications that might use a particular combination of the geographic distribution and membership graph.

First, consider a tracking application that tracks the location of mobile objects such as intruders. The application would need to know of nodes that are currently tracking the object and of any changes in membership of such a group. The nodes themselves could either be clustered together or they could be spread in a uniform random manner. Each participating node would want to have either a complete or partial

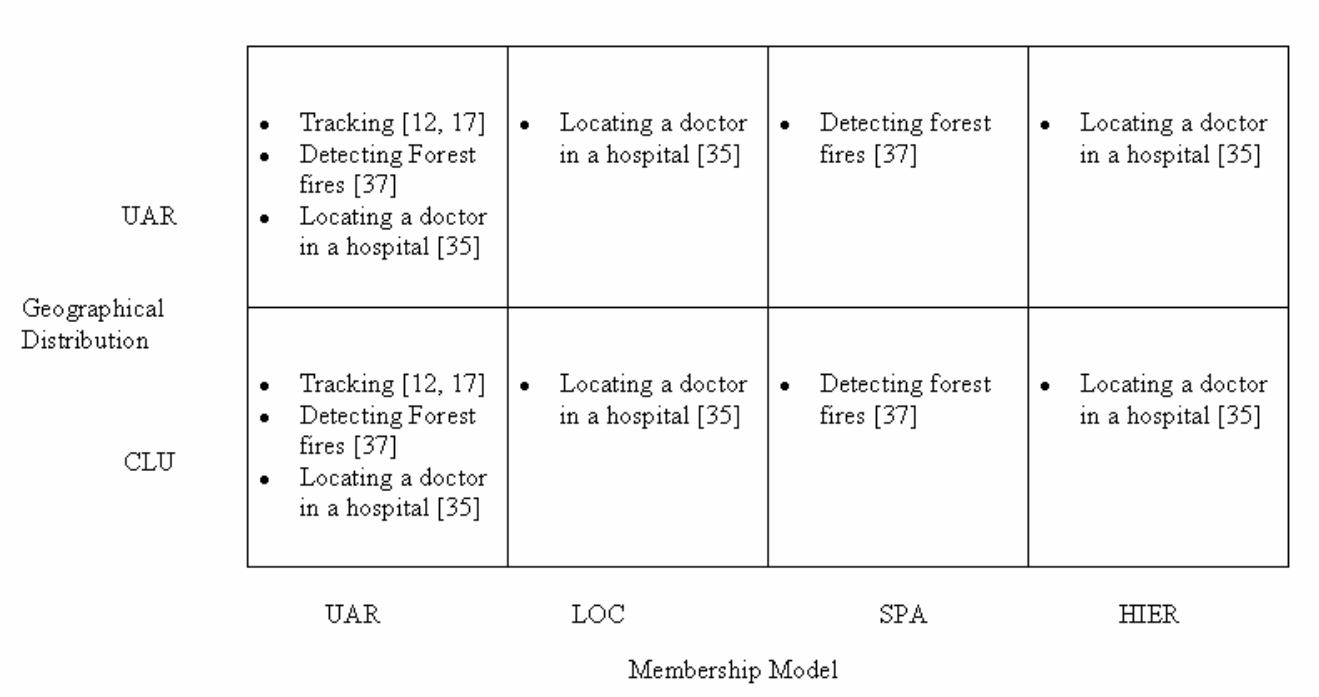


Figure 3.3: Example Applications for different geographic and membership models

membership graph of the other member nodes such that the nodes are picked uniformly at random from the group members. This would be useful for tracking and prediction of movement of objects moving through the area.

Another application that could use such a membership substrate would be an application that would detect forest fires. Nodes could be scattered uniformly at random in a forest area and could then sense for changes in temperature or for excessive smoke. A node participating in the application would require a membership graph drawn uniformly at random so as to be able to detect fires across the whole forest area. Another possibility is for a node to have a membership graph that is spatial in construction. In such a case nodes would lay more emphasis on monitoring their immediate vicinity compared to area further away. Such nodes could be placed either uniformly at random or in areas that are more prone to having fires and need more monitoring compared to other areas that may not be as fire-prone.

Recall the smart room example given in Chapter 1. Consider a whole building, such as a hospital, that has been configured to be 'smart'. In such a 'smart hospital' [35], the doctors, nurses and other staff as well as patients could be wearing electronic badges that work with the Active Badge Location System [1] and these badges could be used by the building's smart network to locate them. Consider such an application for locating doctors in the building. For this application, the sensor nodes would need to be spread across the whole building in a uniformly at random manner or clustered around high activity areas such as operation theaters, patient wards, etc. Also the participating nodes would need to maintain a membership graph drawn uniformly at random from among all

the group members. Another variation of this application could be an application that could locate a particular specialist doctor from among the doctors in the building. For such an application, the membership graph could be a Hierarchical (HIER) graph that could maintain 2 or more levels of hierarchy. A query for a particular specialist doctor could traverse up the hierarchical levels till a member is found. A third variation could be an application that would locate a doctor if he were within a specified distance. For this kind of an application, a node could maintain a localized (LOC) graph for a specified hop distance. This could be useful to, say, locate doctors who are on a particular floor of the building or who are in a particular section. Such an application would also reap the benefit of consuming less power (since it would use the LOC protocol). Finally, as mentioned earlier, the geographical distribution of the nodes could also be changed to make the nodes more clustered in those sections of the hospital where there is more activity. Thus there could be a higher concentration of nodes in the Out Patient Department as compared to the Recuperative ward.

4 MEMBERSHIP PROTOCOLS

A membership protocol provides each member (of a specified group) with a complete or partial, locally-maintained list of all non-faulty members that belong to that group. The protocol has to ensure that changes in membership, either through members joining or through members leaving or failing, are made known to all non-faulty members of the group. Most membership protocols have, at a minimum, two components: one for detecting failures or node withdrawals and the second for spreading the updates of membership information through the network.

4.1 Failure Detectors

Failure detection mechanisms have two basic metrics that have a bearing on their performance. The first one is *Completeness*, which is the guarantee that the failure of a node is eventually detected by all the non-faulty members of the group. The second metric is *Accuracy*, which is the guarantee that no non-faulty member would be declared as having failed by another non-faulty member. Basically, an accurate failure detector would not produce false positives.

Chandra and Toueg in [10] showed that it was impossible for a failure detector to deterministically achieve both completeness and accuracy over an asynchronous unreliable network. This led to the development of failure detection mechanisms that

guaranteed completeness but achieved accuracy only in a probabilistic manner [6, 7, 8, 9].

Failure detectors have traditionally relied on all-to-all heartbeating algorithms to detect failures. In such algorithms, each node would periodically send a heartbeat message to all the other nodes in the group. In case a heartbeat from a particular node was not received within a timeout, then that node would be declared as having failed. In most distributed systems [6, 7, 8], some variant of the heartbeat mechanism is used for failure detection.

However, heartbeat based membership protocols do not scale with group size [9] since either the detection time or the per-process overhead grows linearly with the group size. To overcome this, failure detectors such as SWIM [11] adopted an opposite policy of random-probing [22, 23, 24] to detect failures. In such a system, each node would choose a random node from its membership list and would ping that node. If the pinged node did not respond with an ack within a specified time-out, it would be declared as having failed. Membership protocols based on such failure detection scale much better because they exhibit detection times and per-process overheads that are invariant with group size.

We take the random pinging methodology to implement the failure detection component of our membership substrate. We evaluate three types of pinging strategies which are described below:

- *UAR (Uniformly at Random)*: The UAR pinging strategy is the simplest strategy of the three. In this, a node selects one node uniformly at random from among the nodes in its membership list and pings that selected node.

- *LOCn (Local within n)*: The LOC*n* scheme selects ping targets from among the nodes in the membership list, such that the target node is chosen uniformly at random from among the nodes within an *n*-hop radius.
- *SPA (Spatial)*: The SPA scheme selects its ping targets based on spatial probability. The probability of a node being chosen is proportional to $d^{-\alpha}$ where 'd' is the distance (in hops) of the target node from the pinging node. Thus, effectively, the SPA scheme tends to choose nodes that are closer with a higher probability as compared to nodes that are further away.

In this thesis, we are concerned only with completeness, speed of failure detection and traffic and not with accuracy or rate of false positives.

4.2 Update Dissemination

To disseminate the membership updates throughout the network, we evaluate three strategies:

- *Broadcast TTL* ∞: This scheme is equivalent to flooding the membership graph with the membership update. In this, when a node receives a membership update, forwards it once to all the nodes in its membership list. In case a node receives the same update for the second time, then it does not forward it further.
- *Broadcast TTL 1*: In this scheme, the node that detects a failure sends a membership update to all the nodes in its membership list but the node that receives a membership update does not forward the update any further.

- *No Broadcast*: In this scheme, no updates are propagated through the network and each node has to perform failure detection by directly pinging the failed nodes themselves.

4.3 Analysis of the Membership Management Schemes

In this section we analyze a few of the membership management schemes presented in the previous sections. We calculate the probability of a failed node getting directly pinged by at least one non-faulty node and from this we can calculate the average number of rounds it would take for the protocol to detect all failures. The analysis holds for both types of geographical distributions (CLU and UAR).

4.3.1 UAR Membership Graph and UAR Failure Detector

Let the group size be N. Thus, the maximum number of nodes in a membership list can be $n = N - 1$.

Let p be the fraction of nodes that have failed or are faulty.

Therefore, the fraction of non-faulty nodes $q = 1 - p$

Let F_j be a faulty node in the system.

Consider a complete membership list, then

P [at least one non-faulty member chooses F_j as a direct ping target]

$$= 1 - \left(1 - \frac{1}{n}\right)^{q.n}$$

$$\approx \quad 1 - \frac{1}{e^{q}} \qquad \text{(since } n >> 1 \text{ and } e^{-x} = \lim_{n \to \infty}\left(1 - \frac{x}{n}\right)^{n} \text{)}$$

Thus, average number of rounds R can be calculated as

$$R = \frac{1}{1 - e^{-q}}$$

Similarly, for maintaining a partial membership list, if l is the fraction of the complete list size, then

P [at least one non-faulty member chooses F_j as
a direct ping target]

$$= 1 - \left(1 - \frac{1}{l}\right)^{q.l}$$

$$\approx \quad 1 - \frac{1}{e^{q}} \quad \text{(since } n >> 1\text{)}$$

Note that in both cases (complete list and partial list), the probability of the failed node being pinged by a non-faulty node is independent of group size.

Pings vs. Updates

The behavior of the Pinging component (failure detector) at a particular node can be modeled using the following equation that gives the probability of detecting a failure in a particular round by direct pinging:

$$f(t) = \left(1 - \frac{1}{l}\right)^{t-1} . \frac{1}{l}$$

where t is the number of rounds and l is the list size as a fraction of the complete list. Similarly, the Membership Updates component (update dissemination) at a particular node can be modeled using:

$$g(t) = (1 - l)^{t-1} . l$$

which gives the probability of coming to know of a failure in a particular round, through a membership update. Here t is the number of rounds and l is the list size as a fraction of the complete list.

As we see from Figure 4.1, as the list size increases the graph moves towards the y-axis. This indicates that as the listsize increases, the probability that a failure will be detected by direct pinging in a particular round decreases. Also, we see that in Figure 4.2, as the list size increases, the upper part of the graph moves away from the y-axis whereas the lower part of the graph moves towards the y-axis.

Thus, we see two exponential distributions competing with each other. This shows that as the list size increases, the update component (update dissemination component) becomes more dominant than the ping component (failure detection component) in detecting failures.

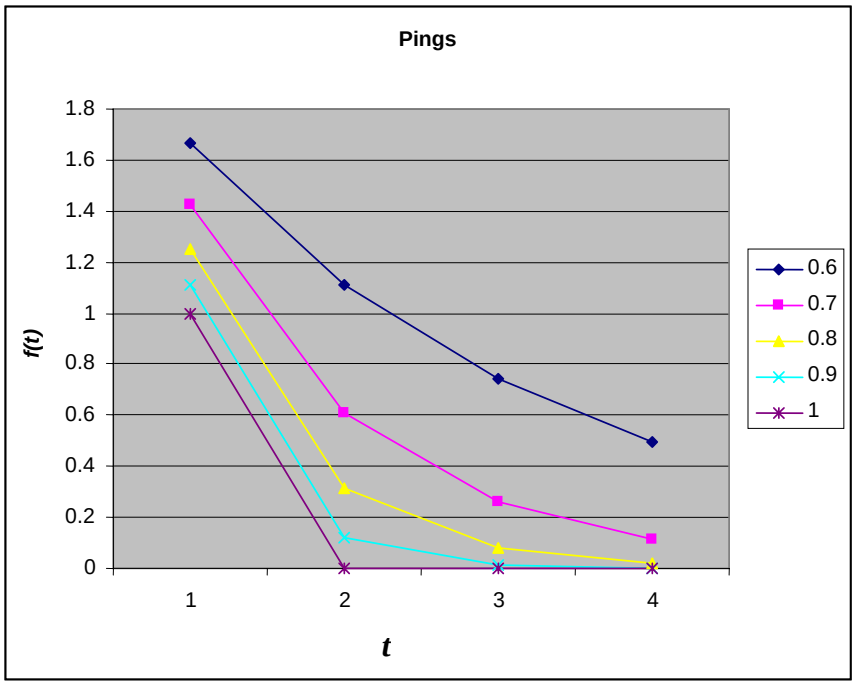


Figure 4.1: Ping behavior with increasing list size.

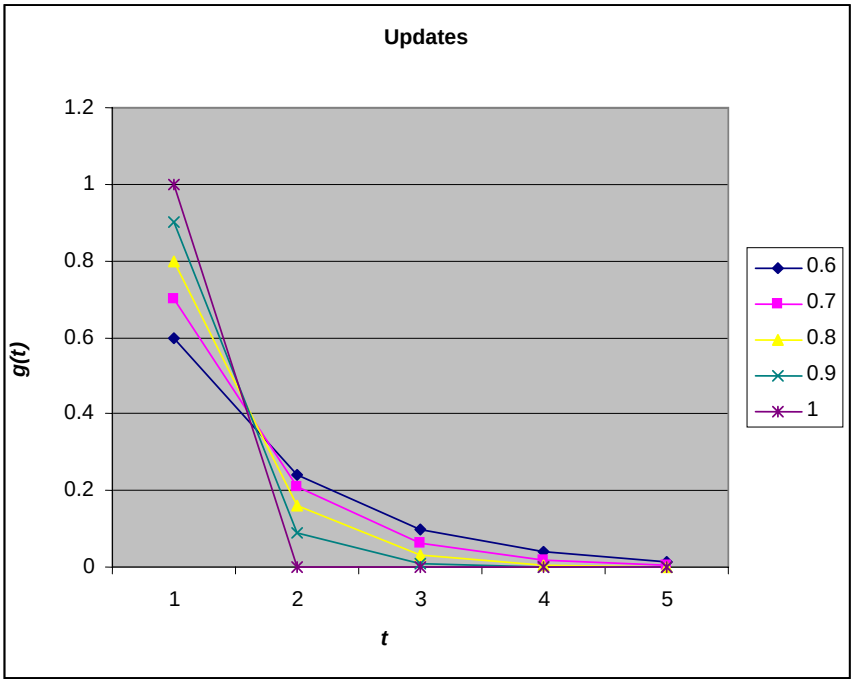


Figure 4.2: Update dissemination with increasing list size.

4.3.2 SPA Membership Graph and UAR Failure Detector

Let k be the average number of nodes that know about F_j.

Then for a complete list of size n

P [at least one non-faulty member chooses F_j as a direct ping target]

$$= 1 - \left(1 - \frac{1}{k}\right)^k$$

Thus, as $k \rightarrow \infty$, $1 - \left(1 - \frac{1}{k}\right)^k$ decreases asymptotically to $1 - e^{-1}$

4.3.3 UAR Membership Graph and LOC Failure Detector

Let k be the average number of neighbors of F_j

Then for a complete list of size n

P [at least one non-faulty member chooses F_j as a direct ping target]

$$= 1 - \left(1 - \frac{1}{k}\right)^k$$

Here too, as $k \rightarrow \infty$, $1 - \left(1 - \frac{1}{k}\right)^k$ decreases asymptotically to $1 - e^{-1}$

Similarly for a partial list with l as the fraction of the complete list

P [at least one non-faulty member chooses F_j as a direct ping target]

$$= 1-\left(1-\frac{1}{k.l}\right)^{k.l}$$

4.3.4 LOC Membership Graph and UAR Failure Detector

Let k be the average number of neighbors of F_j.

Then for a complete list of size n

P [at least one non-faulty member chooses F_j as a direct ping target]

$$= 1-\left(1-\frac{1}{k}\right)^{k}$$

Similarly, as $k \rightarrow \infty$, $1-\left(1-\frac{1}{k}\right)^{k}$ decreases asymptotically to $1-e^{-1}$

4.3.5 UAR Membership Graph and SPA Failure Detector

Let $d_i^{-\alpha}$ be the spatial probability of node i choosing to ping F_j.

Then for a complete list of size n

P [at least one non-faulty member chooses F_j as

a direct ping target]

$$= 1 - \prod_{i=1}^{q.n} \left(1 - \frac{d_i^{-\alpha}}{\sum_{i=1}^{q.n} d_i^{-\alpha}} \right)$$

and for a partial list with l as the fraction of the complete list

P [at least one non-faulty member chooses F_j as

a direct ping target]

$$= 1 - \prod_{i=1}^{q.l} \left(1 - \frac{d_i^{-\alpha}}{\sum_{i=1}^{q.l} d_i^{-\alpha}} \right)$$

4.3.6 Conclusion

An exhaustive mathematical analysis of all the possible combinations of membership graphs and failure detection protocols is intractable, and besides, complicated equations, such as those in Section 4.3.5. are not very useful as guidelines for an application developer.

In such situations, simulations have proven to be highly useful tools to understand the workings of models that might be very complicated to analyze mathematically. The next chapter presents results from our simulation experiments and evaluates the behavior

of different membership management protocols with respect to different application requirements.

5 MATCHING APPLICATION REQUIREMENTS WITH MEMBERSHIP PROTOCOLS

The previous chapters presented various geographical distributions, membership graphs and different membership management protocols. Chapter 4 also presented mathematical analysis of the membership protocols.

This chapter attempts to match application membership requirements to the various membership protocols using experimental results obtained from implementations of these protocols, which were tested using our simulator. Section 5.1 puts forward certain 'rules of thumb' regarding the membership protocols. Section 5.2 describes the experimental environment including the simulator and the simulation parameters used for the experiments while Section 5.3 discusses the experimental results so obtained.

5.1 Rules of Thumb

While choosing among the different membership protocols with regards to the requirements of a particular application, there are certain rules of thumb that need to be kept in mind:

I. *Of the various protocols described, the UAR and the SPA protocols ensure completeness regardless of the membership graphs or the failure detection schemes employed.*

II. *The LOC failure detection scheme can ensure completeness only if the ping radius is greater than or equal to the membership graph radius and if a complete membership list is maintained. In case partial membership lists are maintained, there always is a non-zero probability that a failed node might be isolated in a manner that it would not be in any other node's membership list and thus, would never be detected as having failed.*

5.2 Experimental Environment

We implemented a membership substrate as a nesC [3] library for use in wireless sensor network applications and evaluated it using TOSSIM [5]. Resource constraints limited the maximum network size to 200 nodes; hence we simulated the behavior of the membership protocols for larger group sizes using a custom simulator we developed in C.

The simulator we developed consists of two major components. The Network layer code provides the basic send/receive functionality to the application layer. Each packet that the application layer sends is queued in the network queues (called the Send queue and the Receive queue) and is retrieved by the application layer in the next round by using the send queue of the previous round as the receive queue of the current round. The Application layer code is used to model the various membership management policies and failure detection mechanisms at each node. There are wrapper functions to send and receive packets to and from the network. Each node also implements a process wrapper that enables the node to apply different membership protocols. There are handler functions for handling ping timeouts and maintaining membership lists.

We simulated the behavior of the membership protocols for different network sizes ranging from 500 nodes to 1000 nodes in the network. The network, in each case, is connected in the sense that there exists a path from each node to each other node in the network. We performed simulations for two different network densities with one set of topologies having a diameter of 7 hops and the second set with a diameter of 3 hops for all the network sizes. We also simulated the behavior of the membership protocols for partial list sizes. For the spatial distribution ($d^{-\alpha}$) schemes, we set the spatial probability constant $\alpha = 4.0$.

Network Sizes (N)	500,600,700,800,900,1000
No of Member Nodes (M)	N/3
No of Failed Nodes (F)	M/3
List sizes (%age of M)	10,25,50,75,100
Network Diameters (D)	3, 7
Spatial probability constant (α)	4.0
No of Clusters (C)	4
No of Leaders per cluster	3
Failure Model	Fail-Stop

Figure 5.1: Simulation Parameters

For the CLU geographic distribution, we create 4 clusters each having (on average) the same number of members. Furthermore, each cluster has 3 leader nodes that are used to maintain a hierarchy for the HIER protocol.

For the simulations, each network had one-third of the nodes as members of a specified group. At t=0, one-third of the members are chosen uniformly at random and are rendered as failed nodes and remain failed throughout the simulation. The simulation is then run to detect all the failures and to update each node's membership list.

Also, as stated earlier, we assume that each sensor node has a globally unique identification. This ID need not necessarily be assigned at fabrication or compile time and may be assigned at run time, as long as each sensor node is uniquely identifiable. The ID can also be based on any other criteria such as approximate location [38, 39, 40], etc.

The various simulation parameters are summarized in Figure 5.1.

5.2.1 Protocol Combinations

Recall that in the previous chapters we have presented various geographical distributions, membership graphs and membership management protocols for wireless sensor networks. Figure 5.2 gives an overview of all the combinations possible using these properties.

We have 2 geographic distributions, viz. UAR and CLU and for each of these distributions we have 6 and 7 (respectively) relevant membership graphs. This equals a total of 13 combinations for the geographic distribution and the membership graph. These two parameters, that is, the geographic distribution and the membership graph are

essentially properties of the application being built and thus are the base cases for which the membership protocols are evaluated.

For each of these 13 application properties combinations, there are 5 failure detection strategies and for each failure detection strategy, there are 3 information dissemination strategies. Thus, there are a total of 195 combinations for matching application requirements to membership protocols. The goal of our thesis is to eliminate the bad combinations among these, and provide enough analysis and experimental data for the other combinations that makes an application designer's job easier.

Application Properties | Membership Protocol

Geographic Distributions	Membership Models	Failure Detection strategies	Dissemination strategies
UAR	• UAR • SPA • LOC1 • LOC2 • LOC4 • KLOC	• UAR • LOC1 • LOC2 • LOC4 • SPA	• Broadcast TTL1 • Broadcast TTL ∞ • No Broadcast
CLU	• UAR • SPA • LOC1 • LOC2 • LOC4 • KLOC • HIER	• UAR • LOC1 • LOC2 • LOC4 • SPA	• Broadcast TTL1 • Broadcast TTL ∞ • No Broadcast

Figure 5.2: Matching application properties and membership protocols

Furthermore there are some combinations which are intuitively unworkable. For example, a UAR geographic distribution, a UAR membership graph, a LOC1 failure detection strategy and a No Broadcast dissemination strategy would not be able to detect all failures. This is so because the No Broadcast strategy implies that each node would have to discover all the failed nodes in the system by pinging them directly, and the LOC1 failure detection strategy limits the ping radius to 1 hop. Since the radius of the network is greater than 1 and the membership graph is UAR, there is a high possibility that there would be a failed node that has active nodes outside a 1-hop radius which would never ping it and thus would never detect it as failed.

Another factor is that some combinations degenerate into some other combination. For example, with a UAR geographical distribution and a LOC1 membership graph, a UAR failure detection scheme with a Broadcast TTL1 would be essentially the same as a LOC1 failure detection scheme with a Broadcast TTL1 which, in turn, would be the same as a SPA failure detection scheme with a Broadcast TTL1 (because the LOC1 membership graph restricts the graph to a 1-hop radius, so all the candidate ping targets are within a 1 hop distance and thus for the SPA failure detection scheme the probability of being chosen as a ping target is equal among all candidate nodes since they are all an equal distance from the pinging node).

Figure 5.3 provides a list of the 53 combinations that are unworkable along with a short comment on why a particular combination does not work.

Geographic Distribution	Membership Graph	Failure Detection	Dissemination	Comments
UAR, CLU	UAR	LOCn	No Broadcast	All failures not detected if n < D/2
UAR,CLU	LOC1	LOCn	Any	Degenerates to LOC1-UAR
UAR,CLU	LOC1	SPA	Any	Degenerates to LOC1-UAR
UAR,CLU	LOC*k*	LOC*n* with $n < k$	No Broadcast	Not all failures detected
UAR,CLU	LOC*k*	LOC*n* with $n \geq k$	Any	Degenerates to LOC*k*-UAR
UAR,CLU	SPA	LOCn	No Broadcast	Not all failures detected if n < D/2
CLU	HIER	LOCn	No Broadcast	All failures not detected if n < cluster radius

Figure 5.3: Application requirements and membership protocols combinations that do not work.

5.3 Experimental Results

Over the next few pages in this section, we study, through simulations, the different membership protocols and how they perform for different types of application requirements.

For the simulation results, each point on the plots represents the average of 5 runs and by default we use the Broadcast TTL1 scheme to disseminate membership

updates in the network. The default network diameter is 7 hops. Standard deviations are plotted on all the plots, although they may be too small to be visible.

For the purposes of this study, we define the following terms as:

- *Nodes*: The number of nodes in the network. This includes those nodes that are not part of the specified group.
- *Listsize*: The fraction of the group size that is maintained as the membership list at each member node.
- *Average Rounds*: The number of rounds it takes, on average, for the protocol to terminate either by detecting all failures successfully or by reaching a convergent state with some failures remaining undetected.
- *Maximum Rounds*: The number of rounds it takes for the protocol to detect the last failure before terminating successfully or unsuccessfully.
- *Traffic*: The total traffic in the system which is the total number of *hops* that the protocol incurs before termination. The total traffic comprises of two components: *Ping Traffic* and *Update Traffic*.
- *Ping Traffic*: The number of *hops* incurred by the protocol on account of the periodic pinging mechanism.
- *Update Traffic*: The number of *hops* incurred as a result of spreading membership updates in the network.
- *Duplicate Detections*: For each failed node, if the number of other nodes discovering this failure by direct pinging (as opposed to receiving through a membership update broadcast) is *m*, then number of duplicate detections for this failed node is *m-1*. The *Duplicates* metric reflects the sum of duplicate detections for all the failed nodes.

This metric is a measure of the ratio of the failures being detected by direct pinging as opposed to being updated with information from another node.

- *Residue*: The fraction of failures (defined as membership list entries that should get deleted as opposed to failed nodes) that remain undetected when the protocol terminates. This is a measure of the completeness of the protocol.

As mentioned earlier, in this thesis, we are concerned only with completeness, speed of failure detection and traffic and not with accuracy or rate of false positives.

For the purposes of our study, we use Broadcast-TTL1 as the default update dissemination scheme. Where not mentioned on the plots, Broadcast-TTL1 is the default dissemination scheme. Broadcast-TTL1 generates a fraction of the traffic that Broadcast-TTL∞ generates and thus is suited for resource constrained systems like wireless sensor networks. However, Broadcast-TTL1 achieves lesser traffic compared to Broadcast-TTL∞ by taking more time (or rounds) to detect all failures. Another side effect of having a Broadcast-TTL1 dissemination scheme is that it creates a distinction between maintaining a partial membership list and maintaining a complete membership list. This is also reflected in the plots presented later in the section. We have highlighted these two distinct behaviors by plotting the behavior of an increasing partial list with the list size going up to 75% and then plotting the behavior for a complete list alongside in the same plot. Though the behavior remains similar even till 99% list size but resource constraints limited us from gathering data till such high list sizes. A detailed explanation about the reason behind this dichotomy is presented in Section 5.3.3.

5.3.1 TOSSIM Evaluation

We have implemented the membership protocol as a nesC [3] library so that it could be used in applications for wireless sensor networks. We simulated the behavior of the substrate using TOSSIM [5] to evaluate its working and performance. We ran the simulations for a different numbers of nodes up to a maximum of 200 nodes.

This section presents a comparison between the TOSSIM [5] behavior and the behavior observed in our simulator. As Figures 5.4 and 5.5 show the behavior of the two simulators is comparable both in terms of average number of rounds to detect all failures as well as in the total traffic generated. The plots show the performance for network sizes for up to 60 nodes but similar behavior was observed for networks sizes up to 200 nodes.

Due to resource constraints we could not simulate for larger network sizes on TOSSIM [5], but since the behavior of the two simulators is similar, we expect the evaluations done using our simulator to hold in TOSSIM [5] as well.

5.3.2 UAR Failure Detector on a UAR Membership Graph

The UUuar scheme has a UAR geographic distribution of the sensor nodes, a UAR membership graph, employs a UAR pinging strategy for failure detection and uses Broadcast TTL1 for membership update dissemination.

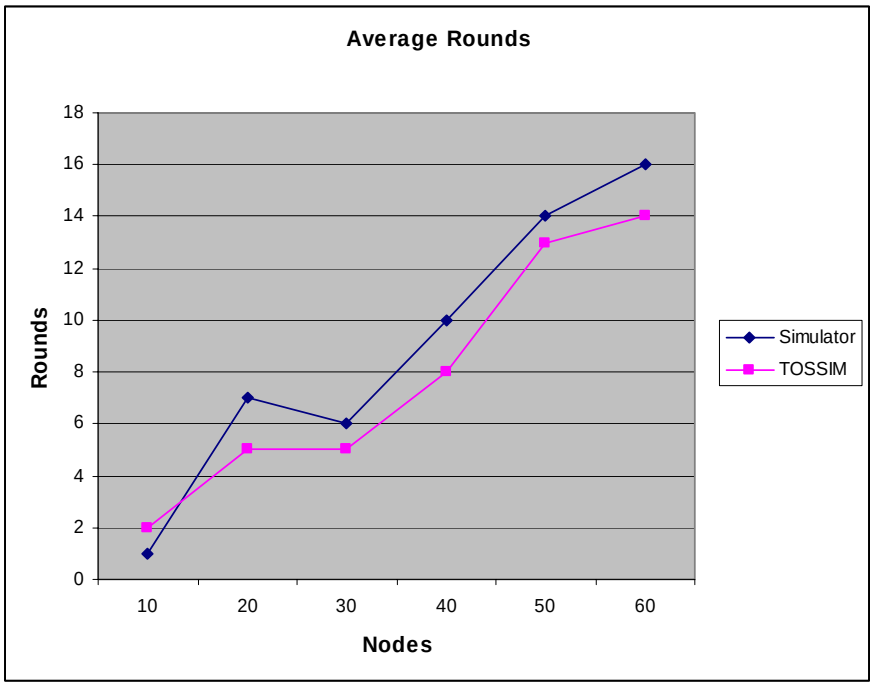


Figure 5.4: **Average number of rounds for UUuar at list size 100%**: comparisons between TOSSIM and our simulator.

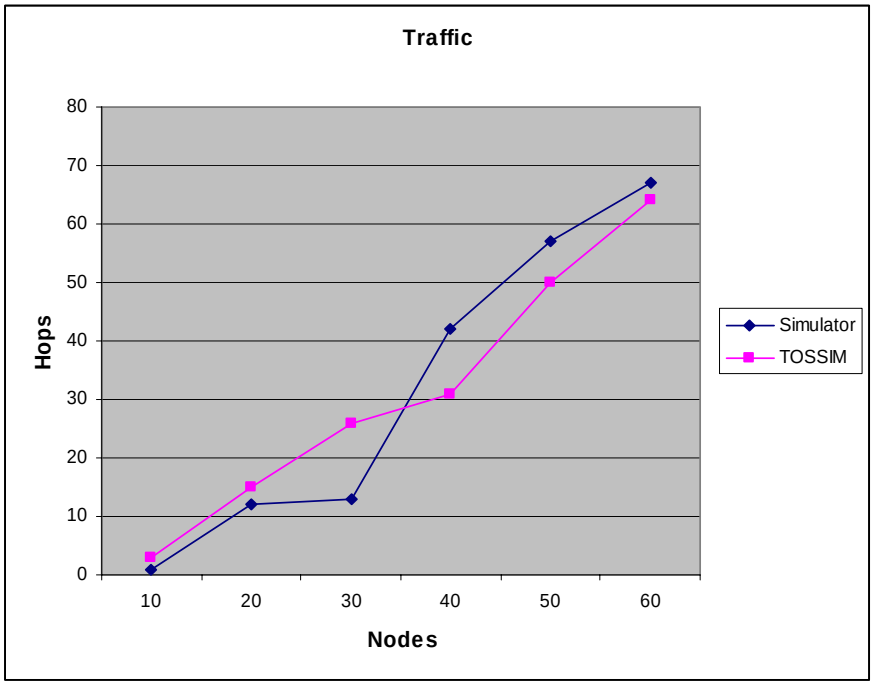


Figure 5.5: **Total traffic for UUuar at list size 100%**: comparisons between TOSSIM and our simulator.

Figure 5.4 shows the average number of rounds that it takes for the scheme to terminate, either in a successful manner by detecting all failures or unsuccessfully by reaching a convergent state wherein some failures remain undetected. As expected and as shown in the previous analysis, as the list size increases, the average number of rounds it takes for the protocol to stabilize decreases. This is so because as the list size increases there is a corresponding increase in the overlap between the membership lists of different nodes. Thus with the increase in the list size, a failure that is detected by one node is quickly propagated to the other nodes in the network. This behavior can be more clearly inferred from Figure 5.9 which shows the number of duplicate detections performed in the network. We see that as the list size increases, the number of duplicate detections decreases, to the point that with a complete list (100% list size) there are no duplicate detections. The reason for this is that since, every node maintains a complete list of all the other nodes in the group, any failure detected by one node is immediately propagated to all the other nodes in the network and they also mark it as failed and do not ping it in future.

Figure 5.5 shows a plot of the maximum number of rounds it took for the protocol to stabilize. We see that as the list size increases, the maximum number of rounds, which is the number of rounds it takes to detect the last failed node, also increases. This is intuitive because as the list size increases, there are that many more failures to be detected in the system and going by the UAR pinging scheme, it may take a longer amount of time to ping the last undetected node. Also, when the list size is less than complete, duplicate detections are needed to detect all the failures in the system and so the last node to be detected as failed may actually have to be pinged more than once for

all nodes to mark that node as having failed. We also see that the maximum number of rounds drops drastically as the list size becomes 100%. This is so because when the list is complete the membership updates are immediately reflected at all the nodes, so the protocol just needs to detect a failed node once (and when the last node is left, everybody is trying to detect that node so the probability of pinging the failed node increases). A detailed explanation behind the reason for the comparatively lower values when the list size is 100% is presented in the following section (Section 5.3.3).

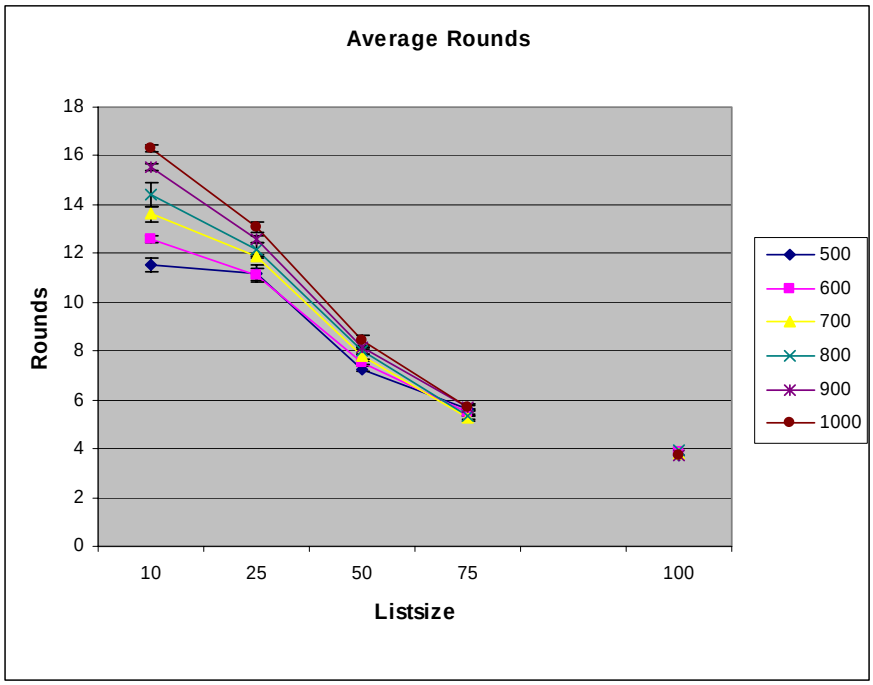


Figure 5.4: **Average number of Rounds for a UUuar scheme**: Each trendline represents different network size and is plotted at different membership list sizes.

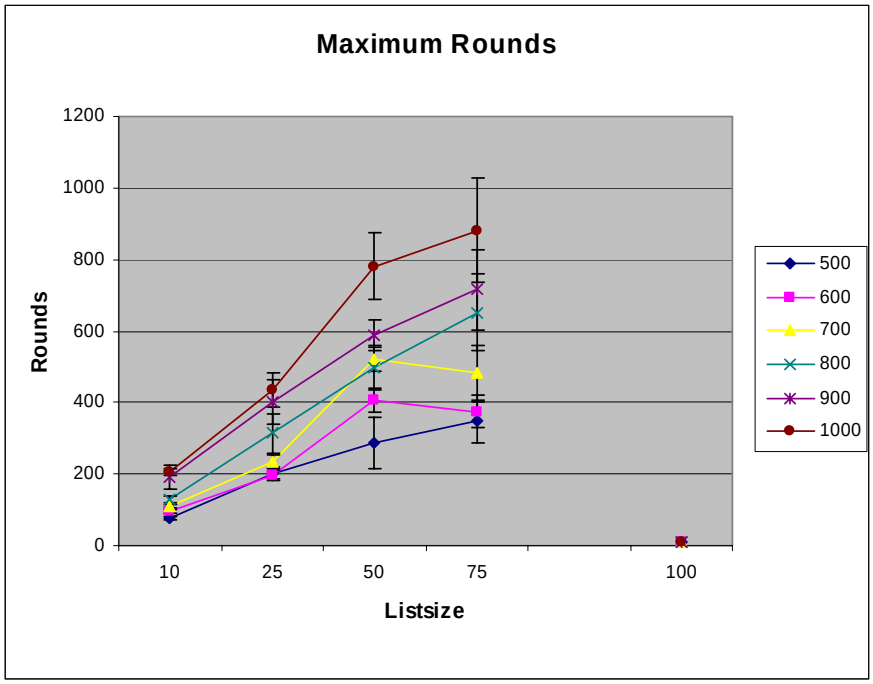


Figure 5.5: **Maximum number of Rounds for a UUuar scheme**: Each trendline represents different network size and is plotted at different membership list sizes

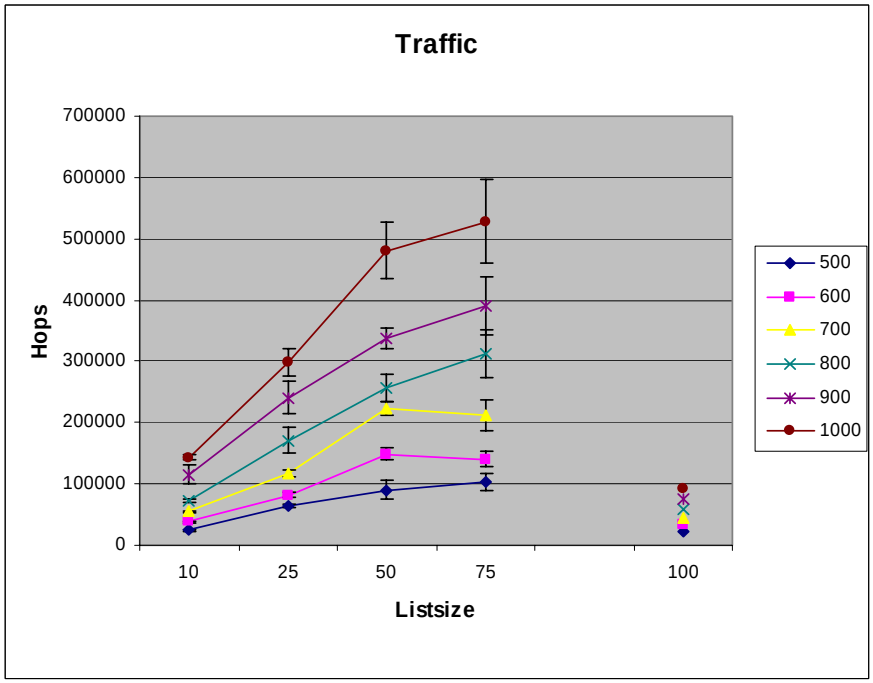


Figure 5.6: **Total Traffic for UUuar scheme**: Plotted for various network sizes at different membership list sizes.

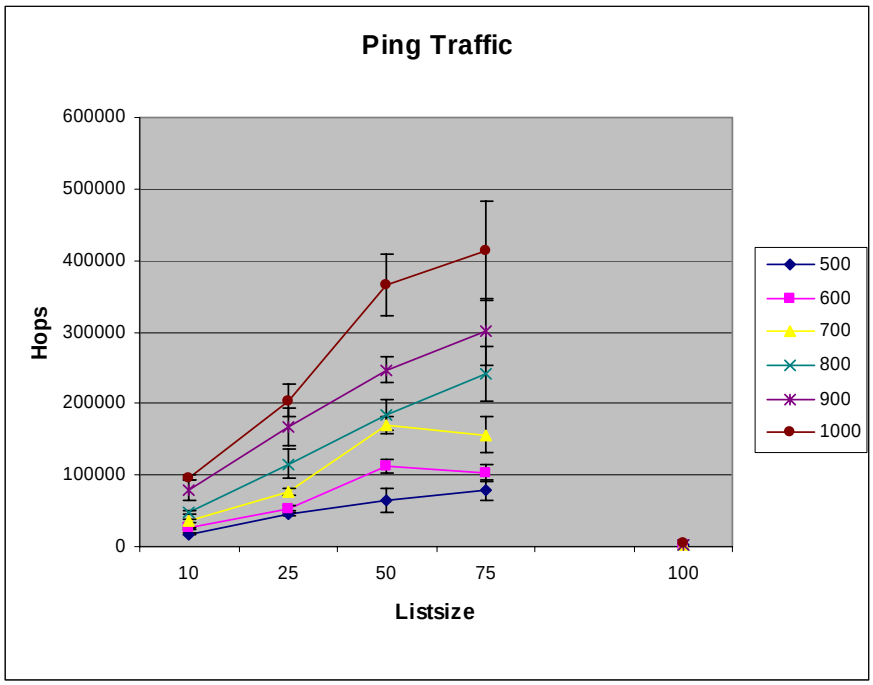


Figure 5.7: **Ping Traffic for UUuar scheme**: Plotted for various network sizes at different membership list sizes.

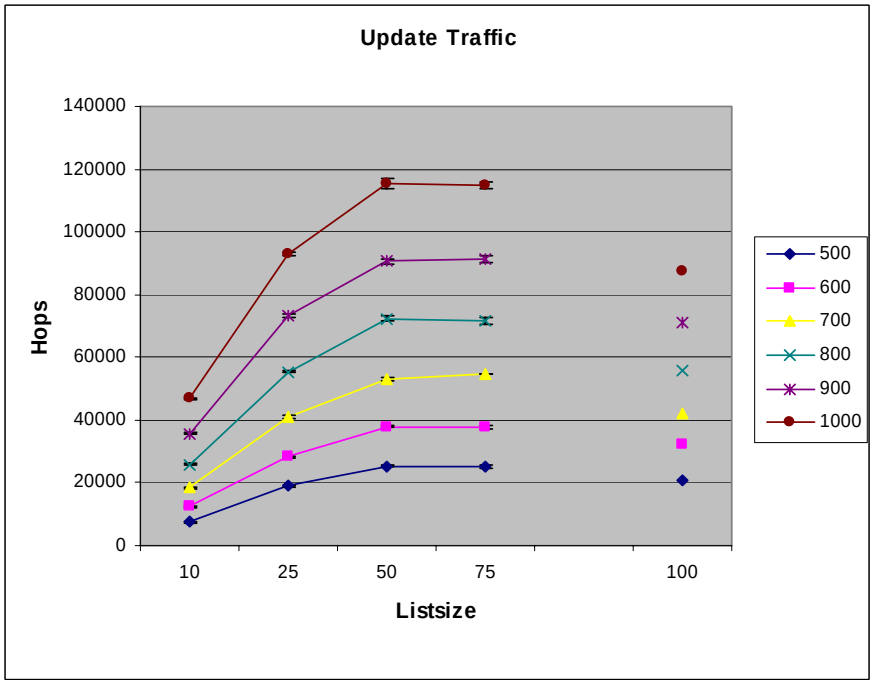


Figure 5.8: **Update Traffic for UUuar scheme**: Plotted for various network sizes at different membership list sizes.

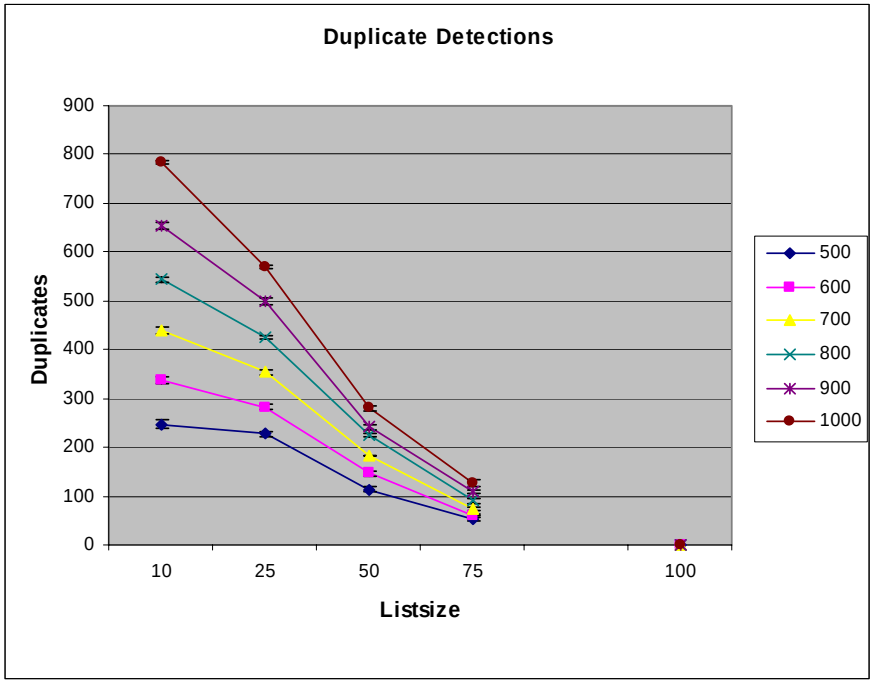


Figure 5.9: **Number of Duplicate Detections performed for UUuar scheme**: Plotted for various network sizes at different membership list sizes. **Residue for this scheme is zero.**

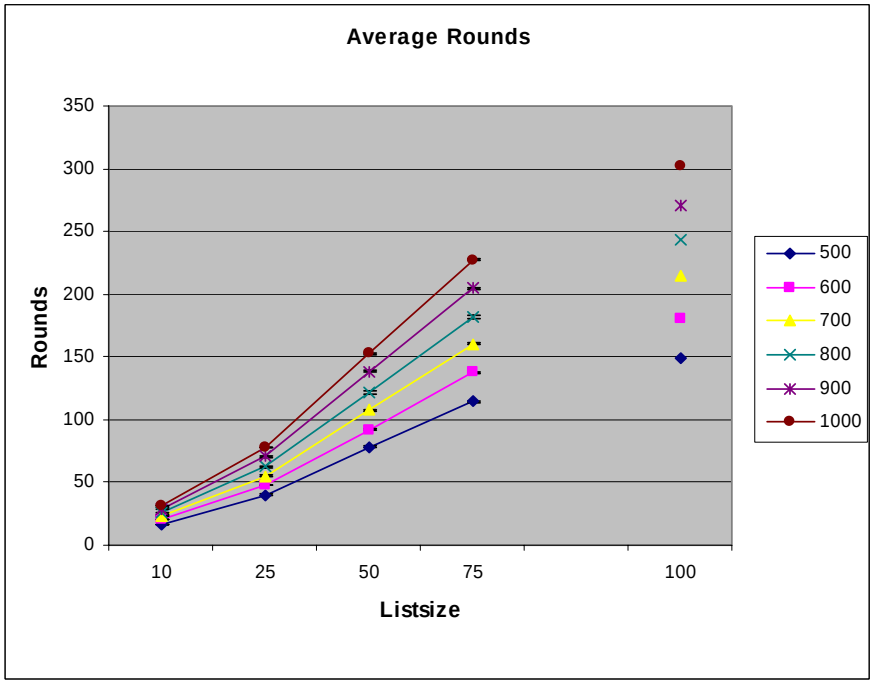


Figure 5.10: **Average number of Rounds for UUuar with No Broadcast**: Plotted for various network sizes at different membership list sizes.

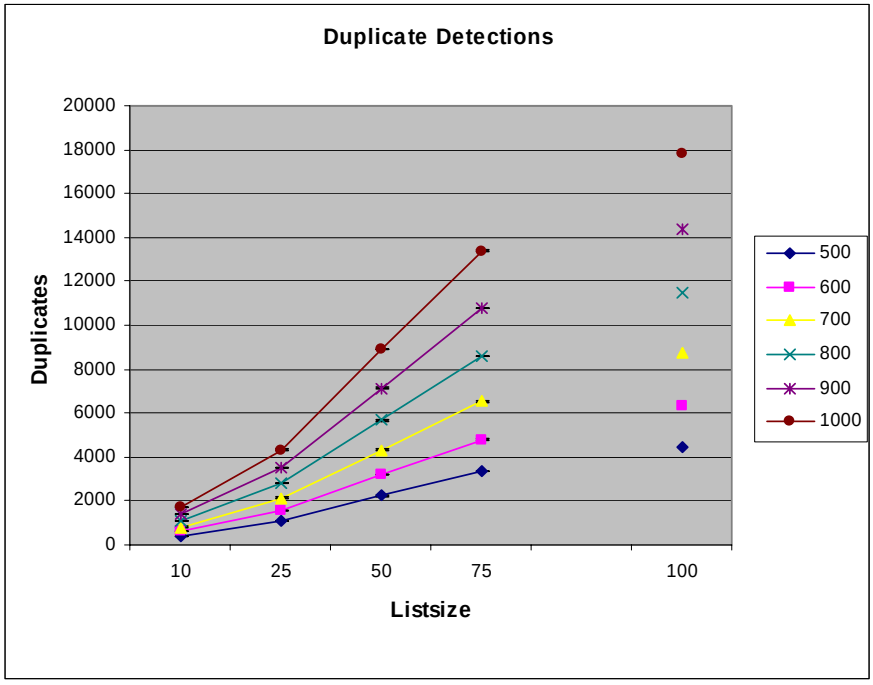


Figure 5.11: **Duplicate Detections for UUuar with No Broadcast**: Plotted for various network sizes at different membership list sizes.

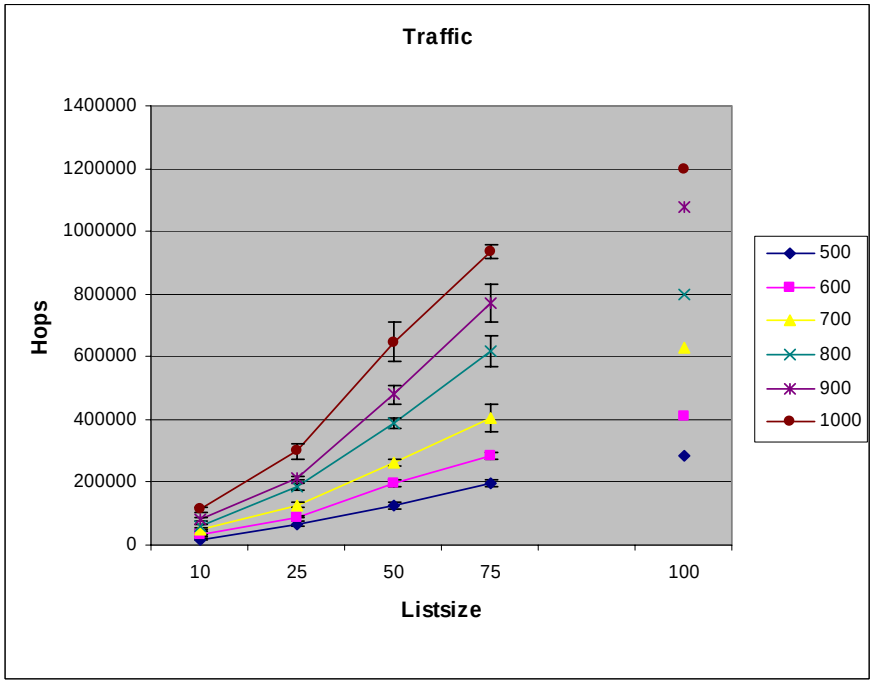


Figure 5.12: **Total Traffic for UUuar with No Broadcast**: Plotted for various network sizes at different membership list sizes.

Figures 5.6, 5.7 and 5.8 show the total traffic, the ping traffic and the update traffic respectively for the protocol. The total traffic is more of a reflection of the maximum number of rounds it takes for the protocol to stabilize. For each round, the total traffic is composed of the ping traffic and the update traffic. In each round, each active node pings one other node from its membership list so the ping traffic is almost proportional to the maximum number of rounds. The update traffic increases as the list size increases because with the increase in list size, the update has to be sent to as many more nodes. The update traffic is related to the number of failures that need to be detected at individual nodes and since there is a fixed number of failures in the system that need to be detected, the update traffic plateaus after the list size grows beyond a particular size.

Since the failure detection scheme is UAR, it ensures Eventual Completeness in the sense that all the failures in the system will eventually be detected. Thus there is zero residue for the scheme.

To see the effect of the dissemination of membership updates on the protocol behavior we ran the simulations using UUuar but with No Broadcast. Figure 5.10 shows the average number of rounds it takes for the protocol to stabilize. The average number of rounds increases as the list size increases. This is as expected because as the list size increases, there are as many more detections to be made before all the active nodes discover all the failed nodes and because there are no membership updates being spread in the network, each node has to detect each failed node by pinging it directly. Similarly, as Figures 5.11 and 5.12 show, the number of duplicate detections and the total traffic also increase in a manner similar to the average number of rounds.

Comparison : UUuar Bcast TTL1 vs. UUuar No Bcast

Figures 5.13 and 5.14 show the comparison in the average number of rounds and total traffic for the UUuar scheme with and without membership updates. While the average number of rounds decrease with increasing list size when updates are propagated, in case the updates are not propagated the average number of rounds goes up drastically. Correspondingly, the total traffic also increases drastically if updates are not propagated.

Similarly for a particular list size, the average number of rounds and the total traffic is significantly greater when updates are not spread, as shown in Figures 5.15 and 5.16. A similar behavior is observed for different list sizes and network sizes.

5.3.3 The Complete List as a Special Case in Broadcast-TTL1

As shown in the plots presented in the previous section, the behavior regarding maintaining a complete membership list (a membership list with 100% list size) is very different from the behavior displayed for maintaining a partial membership list. Consider the UUuar protocol and the maximum number of rounds it takes for the protocol to terminate (Figure 5.5). We see that the maximum number of rounds it takes for the protocol to terminate keeps increasing with increase in the list size till the list size reaches 75% (actually, this behavior continues until (100-ε)% for any positive ε, but is not shown on the plots due to resource constraints) and then for a list size of 100% the maximum number of rounds decreases considerably.

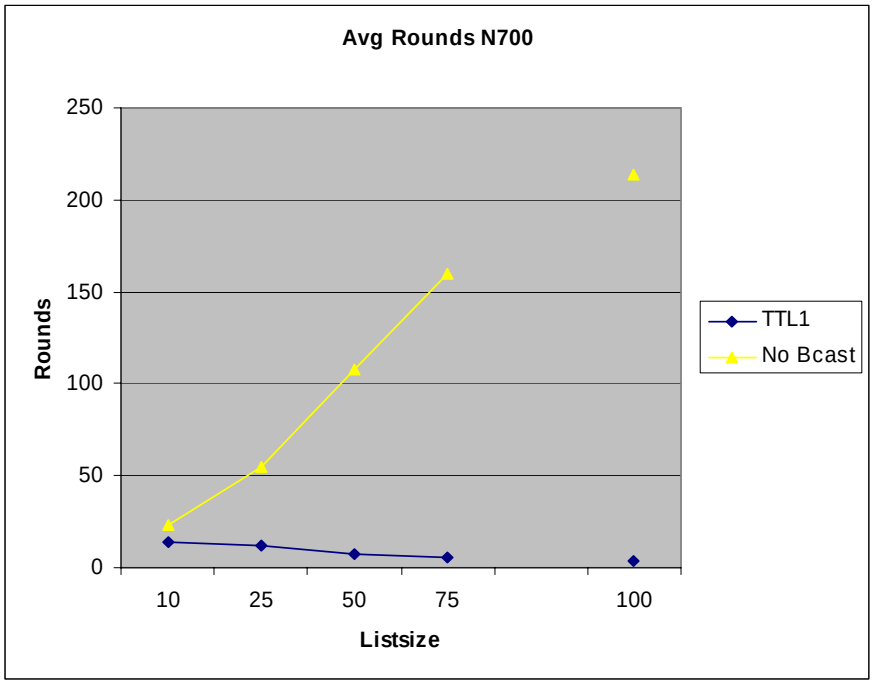


Figure 5.13: **Average number of rounds for a network size of 700 nodes**: comparison between Bcast TTL1 and No Bcast for UUuar.

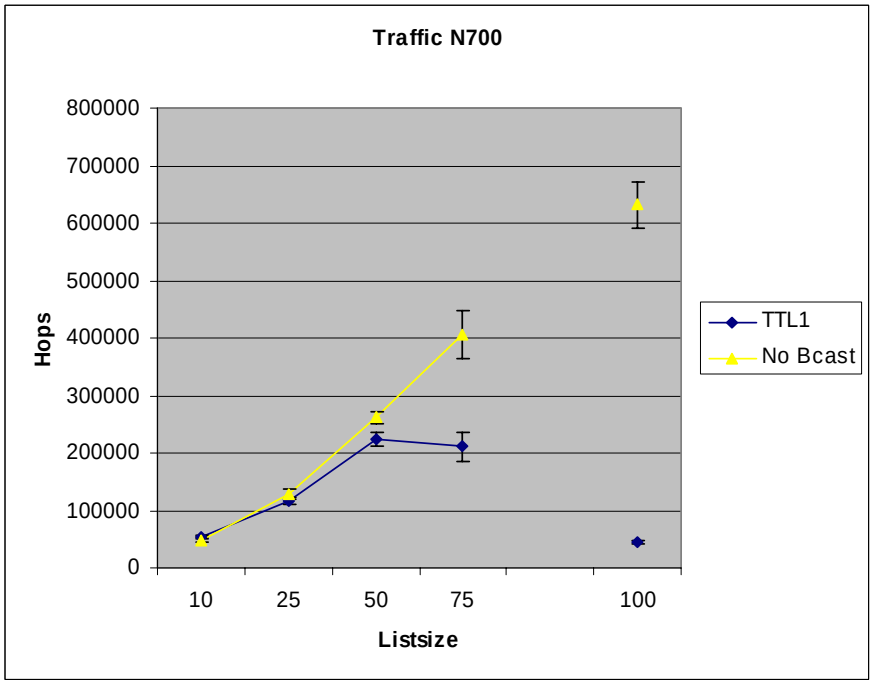


Figure 5.14: **Total traffic for a network size of 700 nodes**: comparison between Bcast TTL1 and No Bcast for UUuar.

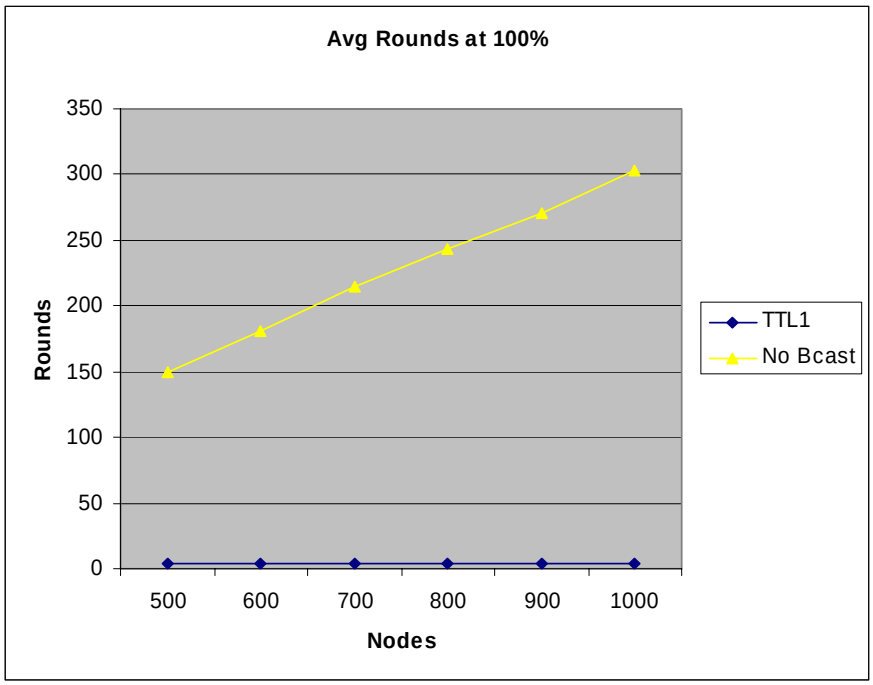


Figure 5.15: **Average number of rounds for maintaining a complete list**: comparison between Bcast TTL1 and No Bcast for UUuar.

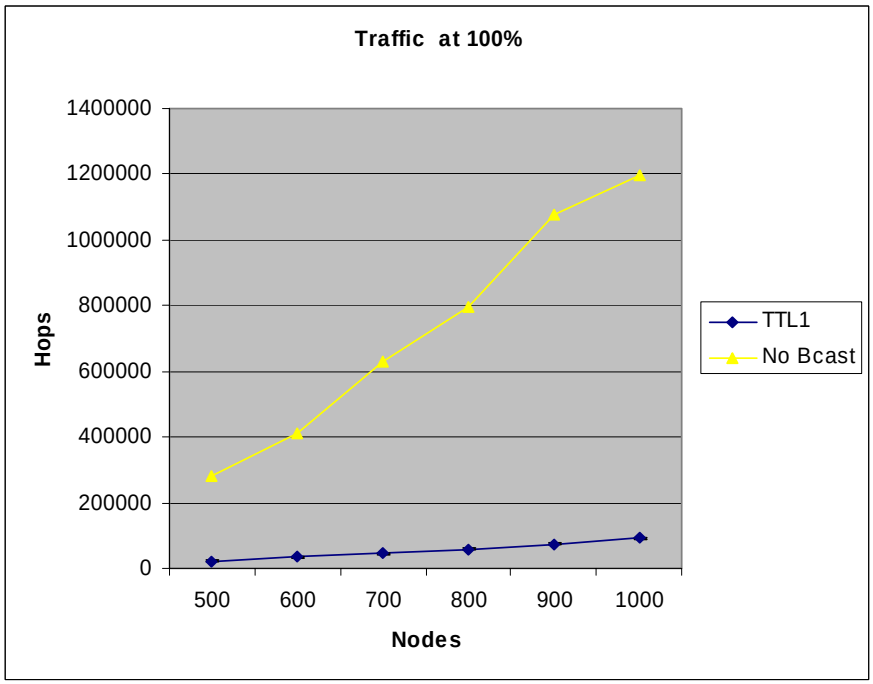


Figure 5.16: **Total traffic for maintaining a complete list**: comparison between Bcast TTL1 and No Bcast for UUuar.

To understand the reason behind this behavior, consider a network which has 101 members in it. Now, if each member node maintains a complete list, then each node's membership list will contain the other 100 members (all other members except itself, therefore, list size = 101 - 1). During the protocol, if one node pings a faulty node and discovers that the node is faulty, it will send the membership update to all the nodes in its membership list. Thus, every non-faulty member will come to know that a particular node is faulty and will proceed to mark it as failed in its own membership list. Thus, when maintaining a complete list, it is required for only one non-faulty node to detect a failed node, and this information will be instantaneously updated in the membership lists of all other non-faulty nodes. And so, the maximum number of rounds required to detect all failures remain very low, as is demonstrated in the plots.

Now, to explain why the maximum number of rounds keeps increasing for maintaining a partial membership list, consider the same network of 101 member nodes and let each member maintain a partial membership list of 99 nodes (1 node less than a complete membership list). Now, when a non-faulty member detects a failed node, it sends out the membership update to the 99 nodes in its membership list and these 99 nodes update their membership lists accordingly. But, there would be 1 node that was not in the detecting member's membership list and that did not get the membership update and thus, would not know of the particular node having failed. Since the dissemination protocol is Broadcast-TTL1, therefore, there is no forwarding of any membership updates and thus, the lone node is left to detect the failure by pinging the failed node directly. Now, as the list size increases, the probability of a particular node being chosen as a target goes down inversely proportional to listsize, (since there are more number of nodes

to choose from) and therefore, it takes longer for the failed node to be chosen as a ping target. Thus we see an increasing trend in the maximum number of rounds which goes on increasing as long as the membership list is not complete.

This behavior, as we shall see occurring in quite a few schemes, is basically due to the use of the Broadcast-TTL1 scheme and can be remedied by increasing the TTL for the broadcast to infinity. But, as will be shown in the next section (Section 5.3.4), using a Broadcast-TTL∞ scheme increases the traffic by orders of magnitude. Thus, it may be worthwhile for application developers to consider using a Broadcast-TTL1 scheme to reduce the traffic drastically in return for a marginal increase the time taken to detect all failures.

5.3.4 LOC Failure Detector on a UAR Membership Graph

The UUloc scheme has a UAR geographic distribution and a UAR membership graph. For the failure detection scheme, ping targets are chosen from the membership list so that they are within a specified hop distance. As mentioned earlier, such a scheme is obviously unable to detect all failures in case no broadcast of membership updates is done.

A Bcast TTL1 information dissemination scheme and a LOC1 failure detection strategy in which the ping targets are chosen so that they are only 1 hop away is never able to detect all the failed nodes unless a complete list is maintained. In case a complete list is maintained, a failure detected at any node is propagated to all other nodes too. Thus

except at a list size of 100%, at all other list sizes LOC leaves a residue of undetected failures as shown in Figure 5.17.

Similarly for a LOC2 failure detection scheme where ping targets are chosen from a 2 hop radius, a residue is left except when maintaining a complete membership list. This is shown in Figure 5.18.

The behavior of the LOC4 scheme shows that it does not leave a residue for any list size. Recall that the network diameter is 7 hops, so choosing ping targets from a radius of 4 hops encompasses the whole network and thus, all failed nodes are within pinging range.

The reason that LOC1 and LOC2 schemes leave a residue is that, since the update dissemination is Bcast TTL1, so only those nodes that are in a node's membership list are sent the update but the update is not forwarded to other nodes. This can be remedied by using a scheme with an infinite TTL in which every node that receives an update forwards it once to all the other nodes in its membership list. As Figures 5.19, 5.20 and 5.21 show, using an infinite TTL does ensure completeness by bringing the residue down to zero. The average number of rounds it takes to detect all failures is also reduced but the traffic in terms of hops goes up enormously.

5.3.5 SPA Failure Detector on a UAR Membership Graph

The UUspa scheme has a UAR geographic distribution of the sensor nodes, a UAR membership graph and spatial ping target selection strategy for failure detection with a Broadcast TTL1 update dissemination scheme. The UUspa scheme ensures

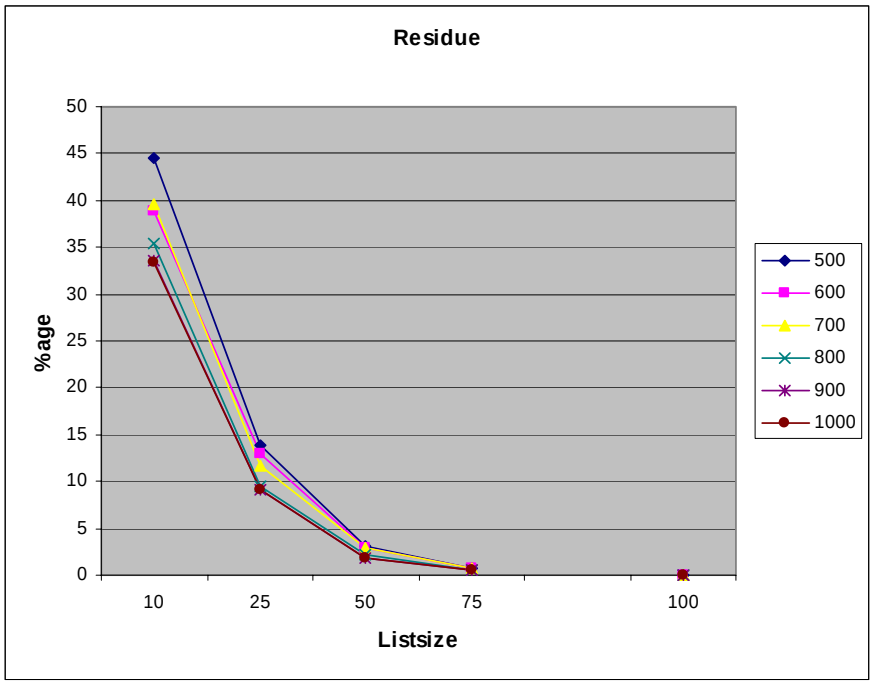


Figure 5.17: **Residue for a UUloc1 scheme**: residue is shown as a fraction of total failures.

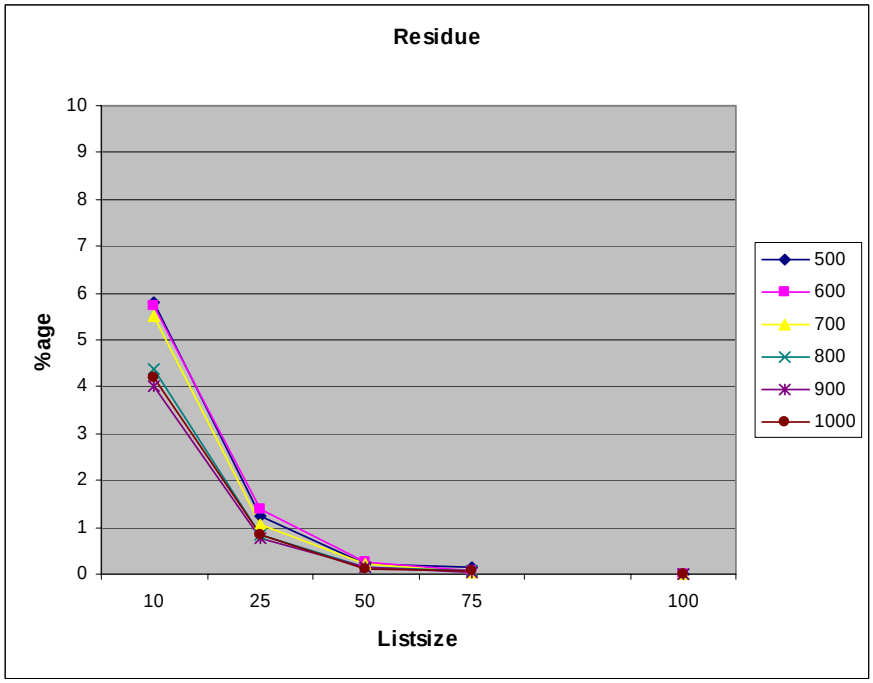


Figure 5.18: **Residue for a UUloc2 scheme**: residue is shown as a fraction of total failures.

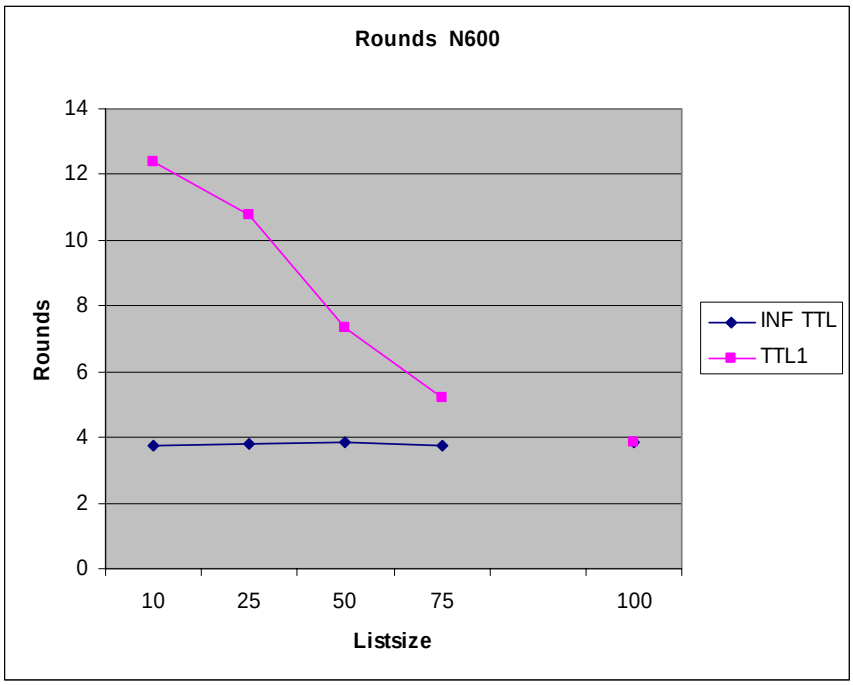


Figure 5.19: **Average number of rounds for a UUloc2 scheme for a network size of 600 nodes**: comparison between using a TTL1 or an infinite TTL.

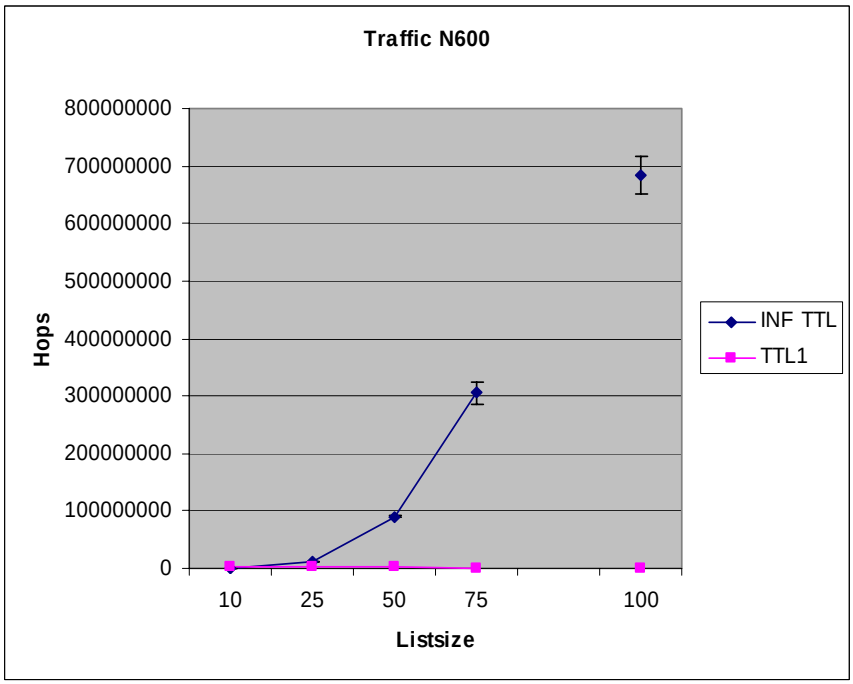


Figure 5.20: **Total traffic for a UUloc2 scheme for a network size of 600 nodes**: comparison between using a TTL1 or an infinite TTL.

List size	TTL1 600Nodes	TTL ∞ 600 Nodes	TTL1 500 Nodes	TTL ∞ 500 Nodes
10	5.75 %	0 %	5.82 %	0 %
25	1.40 %	0 %	1.25 %	0 %
50	0.25 %	0 %	0.23 %	0 %
75	0.06 %	0 %	0.13 %	0 %
100	0 %	0 %	0 %	0 %

Figure 5.21: **Residue for a UUloc2 scheme**: comparison between TTL1 and an infinite TTL

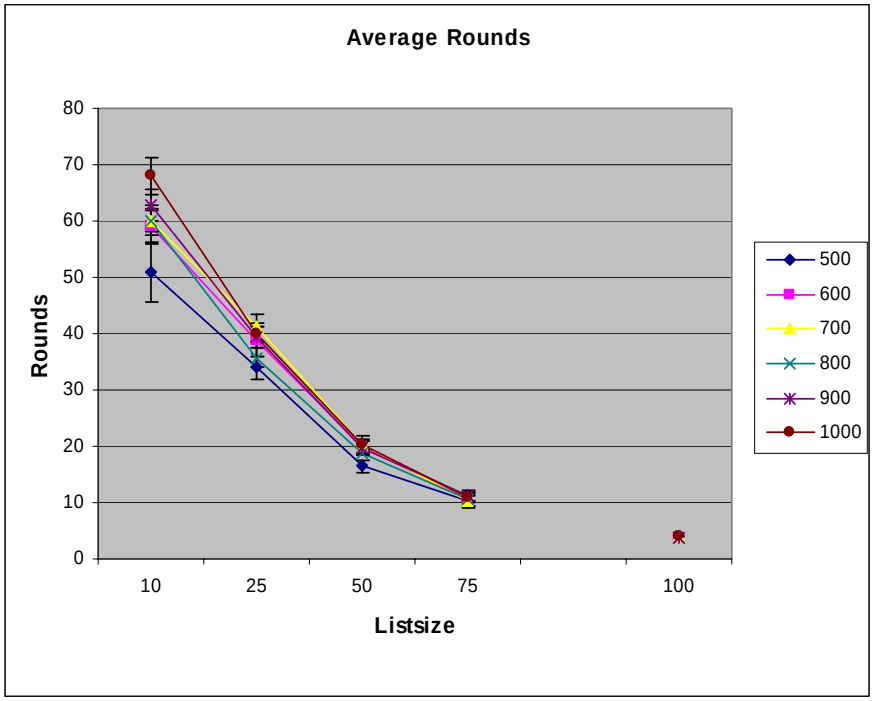


Figure 5.22: **Average number of rounds for UUspa**

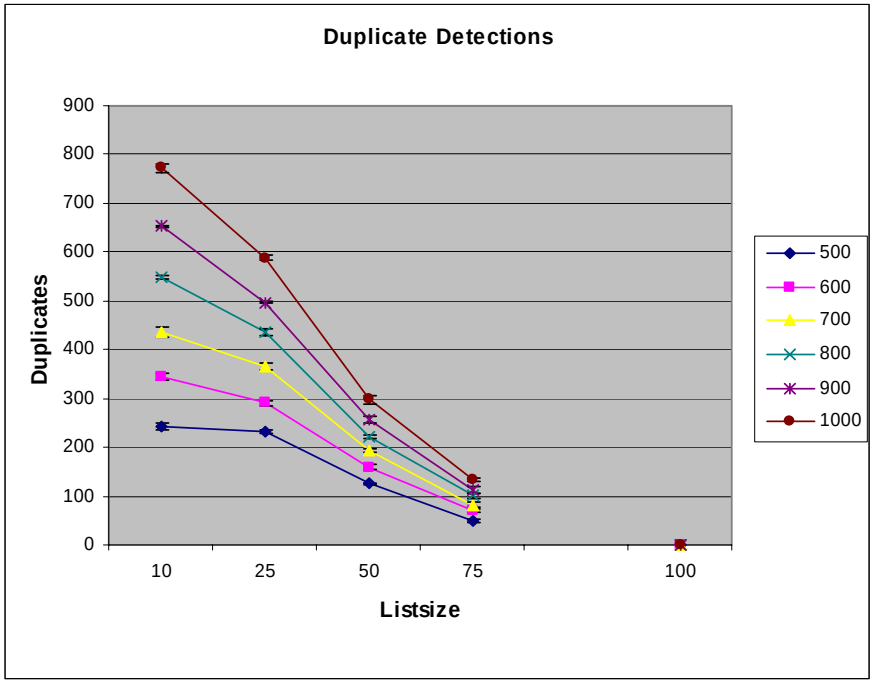


Figure 5.23: **Number of duplicate detections for UUspa**

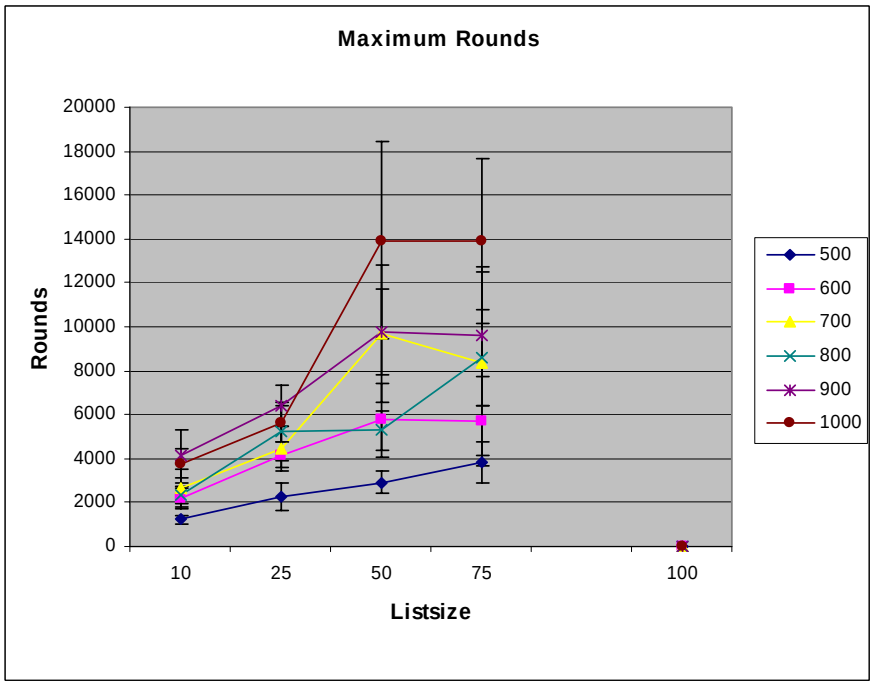


Figure 5.24: **Maximum number of rounds for UUspa**

completeness because every node in the membership list has a non-zero probability of being chosen as a ping target.

Figure 5.22 shows the decrease in average number of rounds as the list size increases. The corresponding number of duplicate detections is shown in Figure 5.23. Consider the drop in number of duplicate detections between list sizes 25 and 50. There is a corresponding drop in the average number of rounds between list sizes 25 and 50. The maximum number of rounds for UUspa, shown in Figure 5.24, reflects the corresponding increase between list size 25 and 50. The maximum number of rounds is considerably larger than the UUuar scheme because UUspa tends to choose closer nodes with a higher probability, so it takes a longer time for a node farther away to be selected as a target.

5.3.6 Comparisons: Schemes with UAR Geographic Distribution and UAR Membership Graph

This section presents a comparison of the different schemes evaluated in the previous sections. We consider only those failure detection schemes that were able to detect all the failures in the system.

Figures 5.25 and 5.26 show the average number of rounds and the total traffic for a network of 700 nodes. The UAR and the SPA failure detection schemes had a zero residue. LOC4 was also able to detect all failures even though in some cases the diameter of the membership graph for a particular node may be less than the network diameter. Thus, UAR, SPA and LOC4 were able to detect all the failures and so are plotted here. The UAR and the LOC4 schemes perform very similar to each other but the SPA strategy

performs worse both in terms of average number of rounds as well as in terms of traffic. The reason for this kind of behavior of the SPA scheme is that since the membership list of a node is drawn uniformly at random, it consists, both of nodes that are near and of nodes that are farther away. As the ping target selection is spatial, so it tends to target the closer nodes more than it targets the nodes that are farther away. And so, failed nodes that are farther away take longer to be picked as ping targets. This is also substantiated by Figure 5.27. Secondly, when the list size is small, there is less overlap between the membership lists of different nodes, so even if updates are propagated they do not spread very far. As the list size grows, so does the overlap between different node's membership lists and an update spreads more in the network. This is also reflected in Figure 5.23 which shows that when the list size is small, a larger number of duplicate detections have to be made. The reason for this is that with a smaller list size, a membership update does not spread very much and so quite a few nodes have to detect all the failures by pinging the failed nodes directly rather than coming to know of the failures through an update.

Figures 5.27 and 5.28 show the comparative performance of the schemes at 100% list size. All the strategies perform almost equally in terms of average number of rounds as well as total traffic. This is borne out by the analysis done in Section 4.3.1 which showed that the average number of rounds it would take for the system to converge would be

$$R = \frac{1}{1-\left(1-\frac{1}{n}\right)^{q.n}}$$

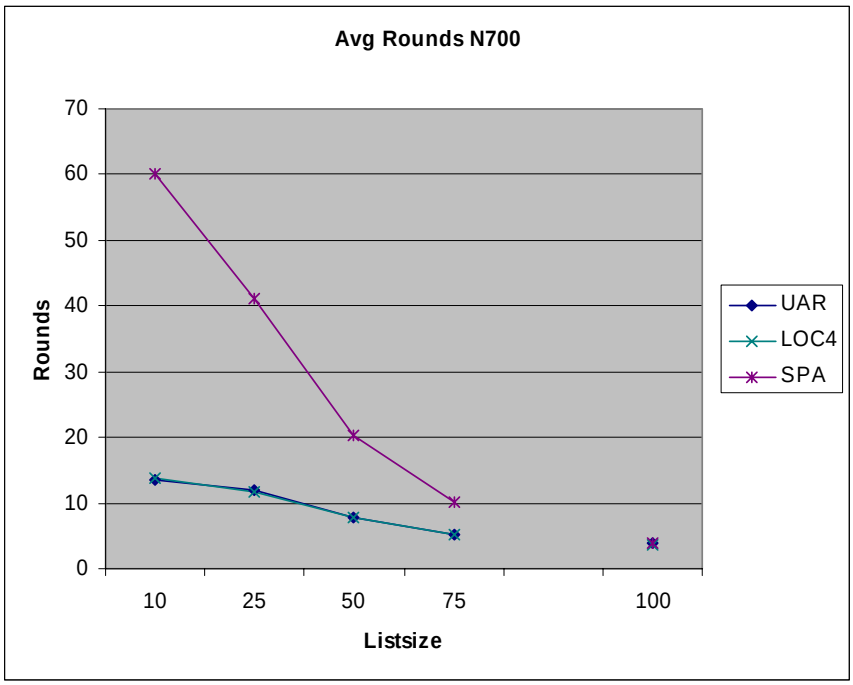


Figure 5.25: **Average number of rounds for network size of 700 nodes**: comparisons between different failure detection strategies for UU*.

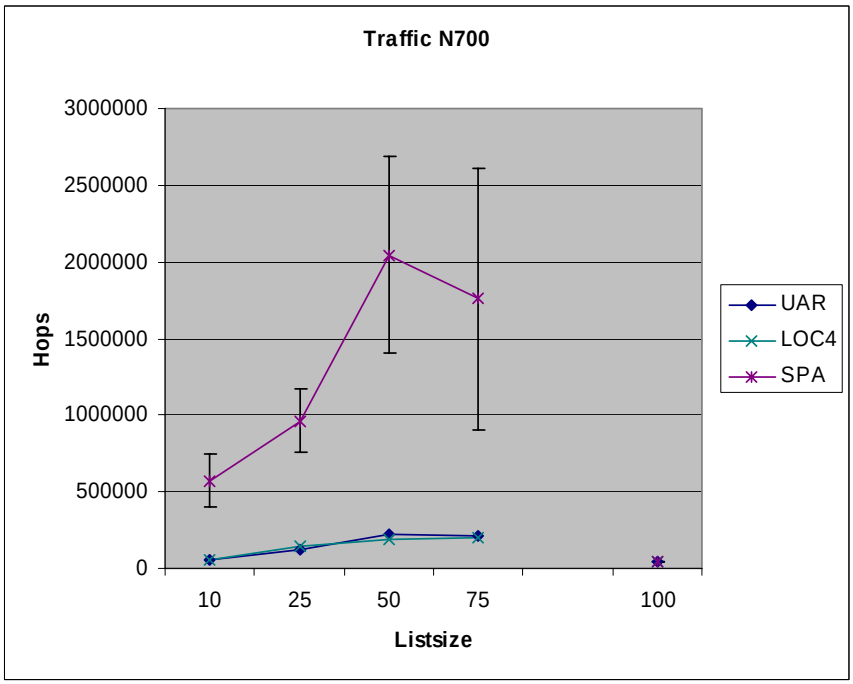


Figure 5.26: **Total traffic for network size of 700 nodes**: comparisons between different failure detection strategies for UU*.

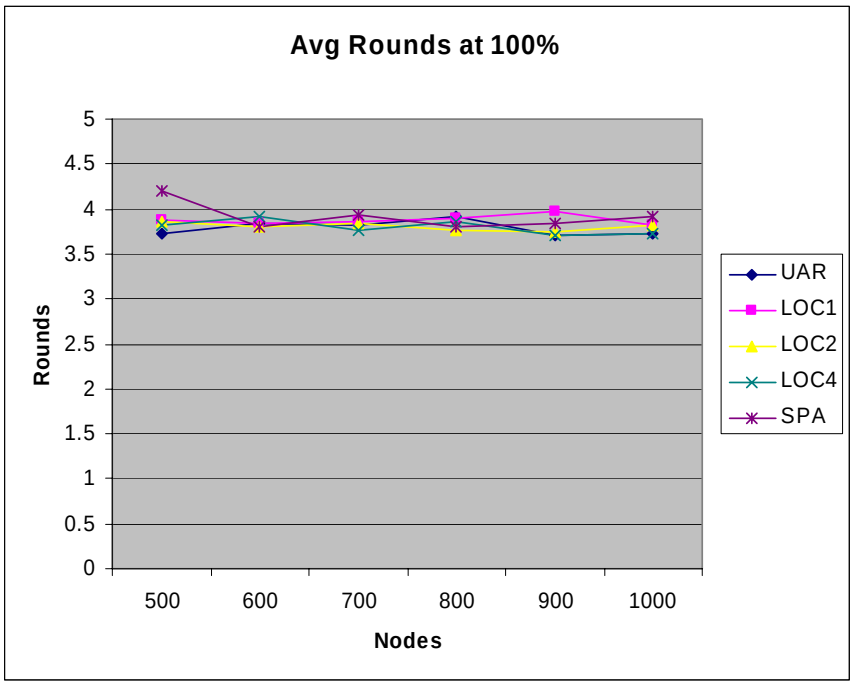


Figure 5.27: **Average number of rounds for maintaining a complete list**: comparisons between different failure detection strategies for UU*.

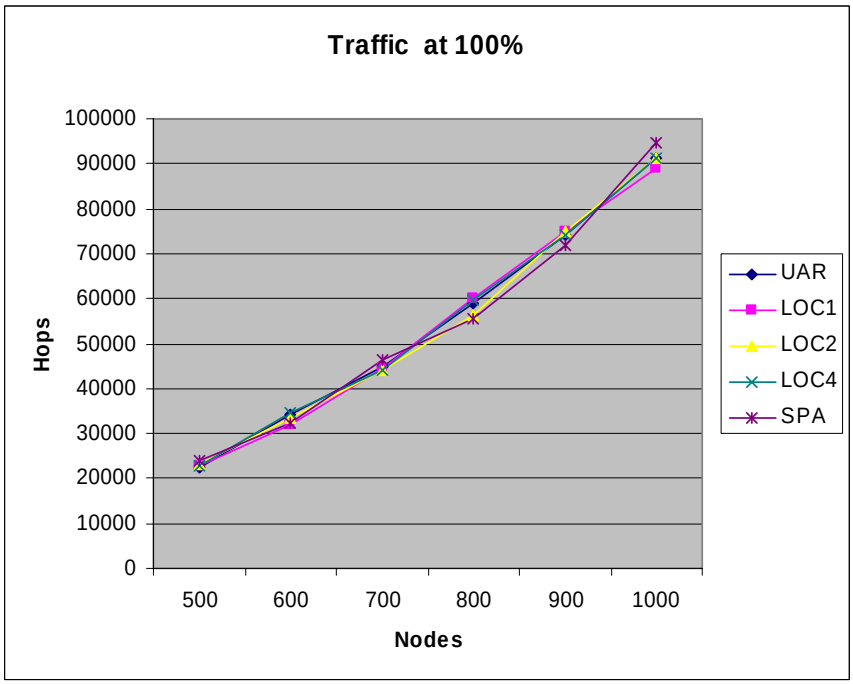


Figure 5.28: **Total traffic for maintaining a complete list**: comparisons between different failure detection strategies for UU*.

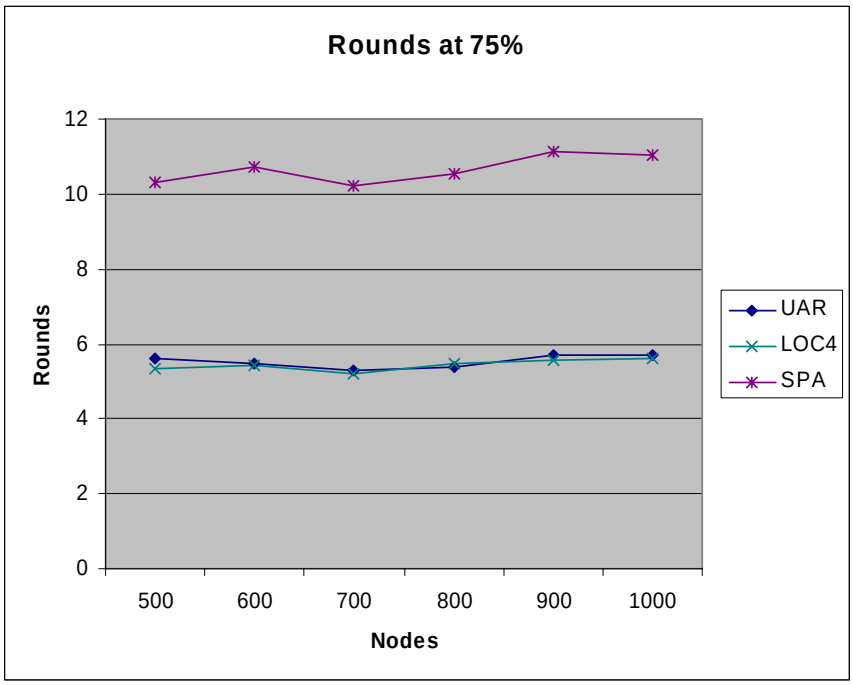


Figure 5.29: **Average number of rounds for maintaining a 75% membership list**: comparisons between different failure detection strategies for UU*.

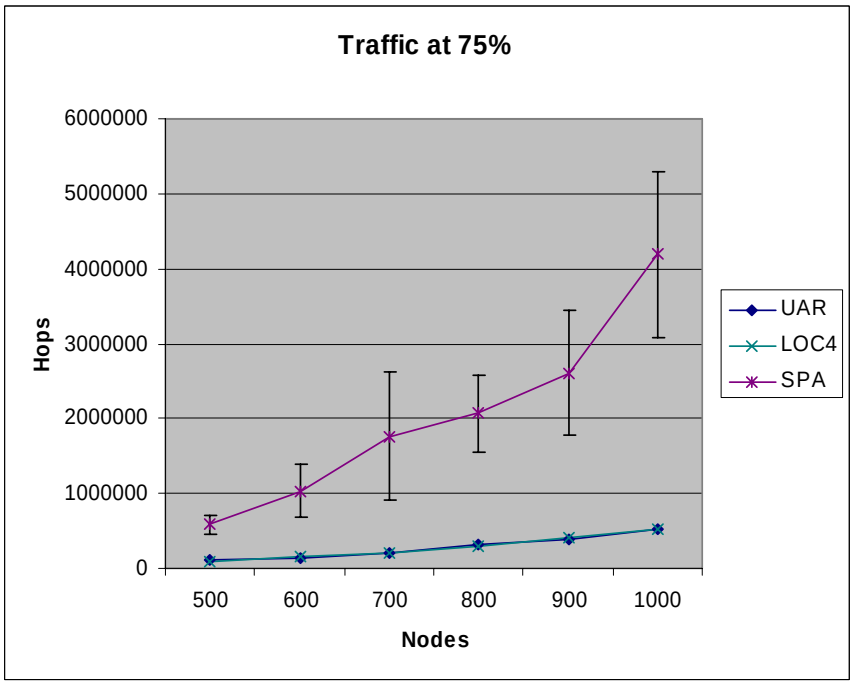


Figure 5.30: **Total traffic for maintaining a 75% membership list**: comparisons between different failure detection strategies for UU*.

Which converges quickly (and asymptotically as $n \to \infty$) to

$$R = \frac{1}{1 - e^{-q}}$$

Thus, increase in the number of nodes does not have much of an effect on the average number of rounds, and consequently, on the traffic in case a complete list is being maintained.

Figures 5.29 and 5.30 show the performance of the UAR, LOC4 and the SPA protocols while maintaining a list size of 75%. As shown by the previous analysis, the SPA scheme performs worse than UAR and LOC4. It requires more number of rounds on average to detect all failures and generates a lot more traffic than the other two schemes.

5.3.7 Comparisons: Schemes with CLU Geographic Distribution vs. UAR Geographic Distribution

We repeated all the simulations after changing the geographical distribution of the sensors to a CLU type of geographic distribution. The behavior of the schemes was observed to be almost similar to the ones using a UAR geographic distribution with the exception of the SPA schemes. The plots for the behavior of the various CLU based schemes are attached as appendix.

Figures 5.31 – 5.36 compare the behavior of the UAR membership protocol and UAR failure detection mechanism when applied to a UAR geographic distribution and to a CLU geographic distribution. As can be seen from the plots and from the previously presented analysis the schemes perform very similar to each other. This is so, because the when the membership graph is drawn uniformly at random and the ping targets are also chosen uniformly at random, the underlying geographical distribution does not affect the membership protocol.

Other membership protocols also behave in a similar manner when compared between a CLU geographic distribution and a UAR geographic distribution with the exception of USspa and CSspa which are shown in Figures 5.37 and 5.38. These plots show the traffic when using a SPA membership graph and a SPA failure detection mechanism. As expected the traffic with the CLU geographic distribution is less as compared to the UAR geographic distribution. The reason is that since the membership graph is drawn in a spatial manner, there are more number of nodes in the membership list that are closer to the pinging node that those that are further away. Since in a CLU geographic distribution, the nodes are clustered together, therefore, the membership list is mostly made up of nodes that are within the cluster and thus, within a few hops. In contrast, even the closer nodes in a UAR geographic distribution are some hops away (which would tend to be a larger number of hops than in a CLU distribution). Therefore, we see the difference in the traffic between the two geographic distributions (though the average numbers of rounds are comparable).

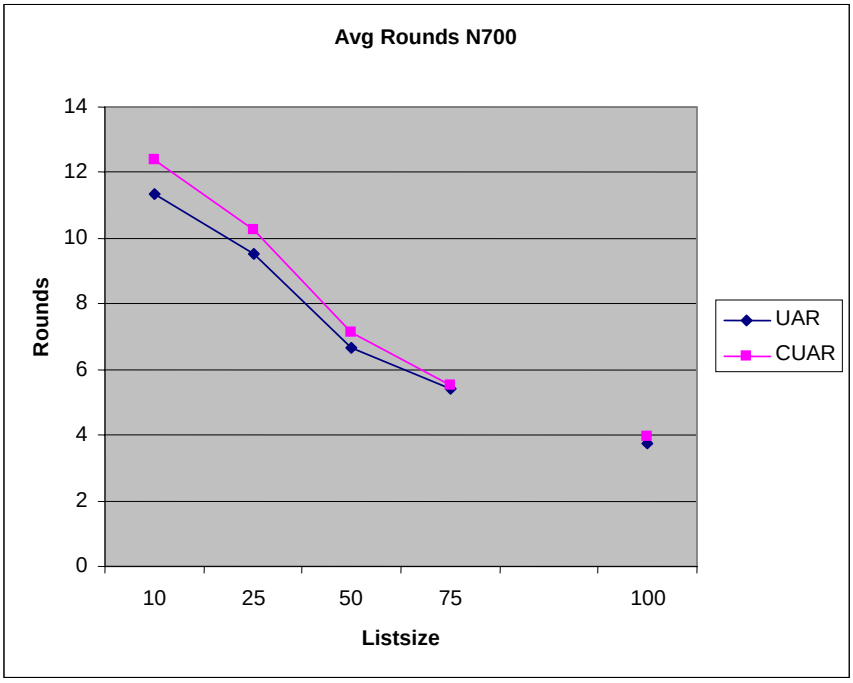


Figure 5.31: **Average number of rounds for a network size of 700 nodes**: comparisons between different geographic distributions strategies for UAR membership graph and UAR failure detection.

Figure 5.32: **Total traffic for a network size of 700 nodes**: comparisons between different geographic distributions strategies for UAR membership graph and UAR failure detection.

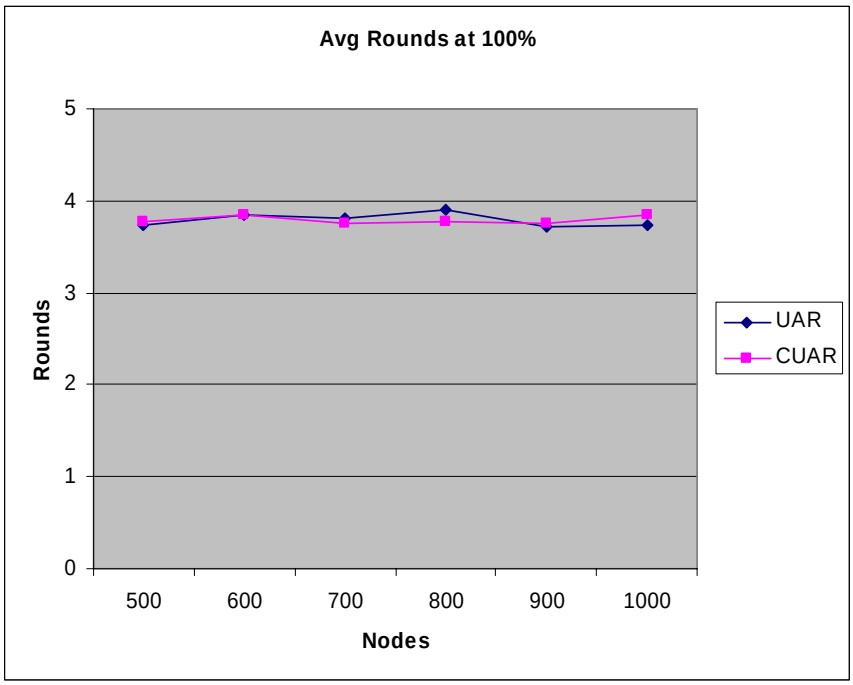


Figure 5.33: **Average number of rounds for a maintaining a complete list**: comparisons between different geographic distributions strategies for UAR membership graph and UAR failure detection.

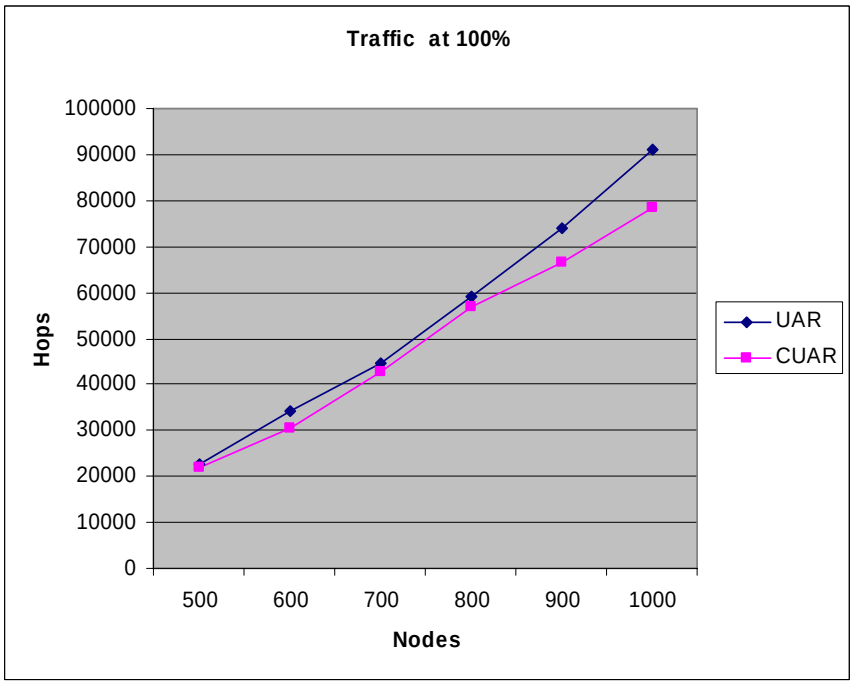


Figure 5.34: **Total traffic for a maintaining a complete list**: comparisons between different geographic distributions strategies for UAR membership graph and UAR failure detection.

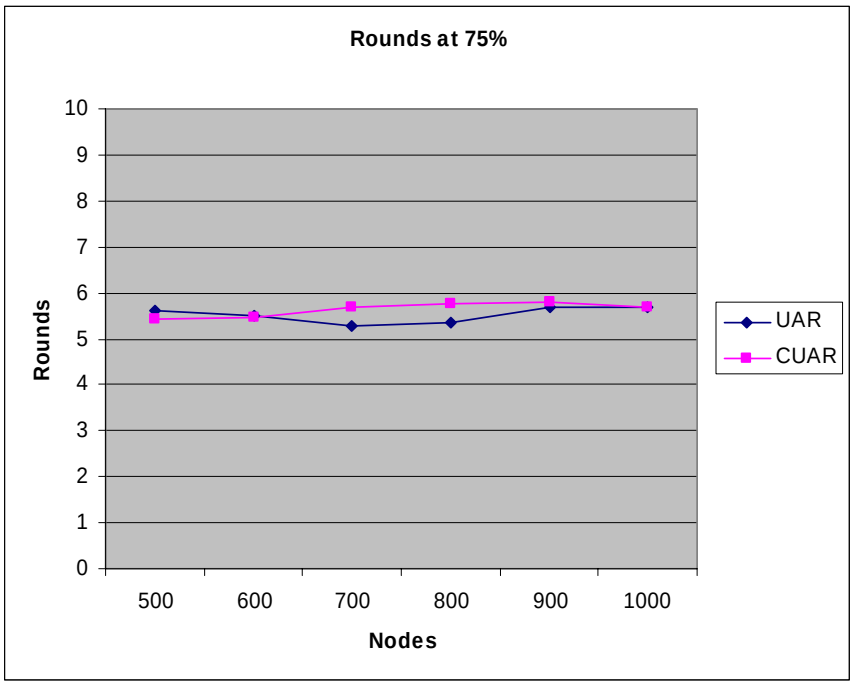


Figure 5.35: **Average number of rounds for a maintaining a 75% list**: comparisons between different geographic distributions strategies for UAR membership graph and UAR failure detection.

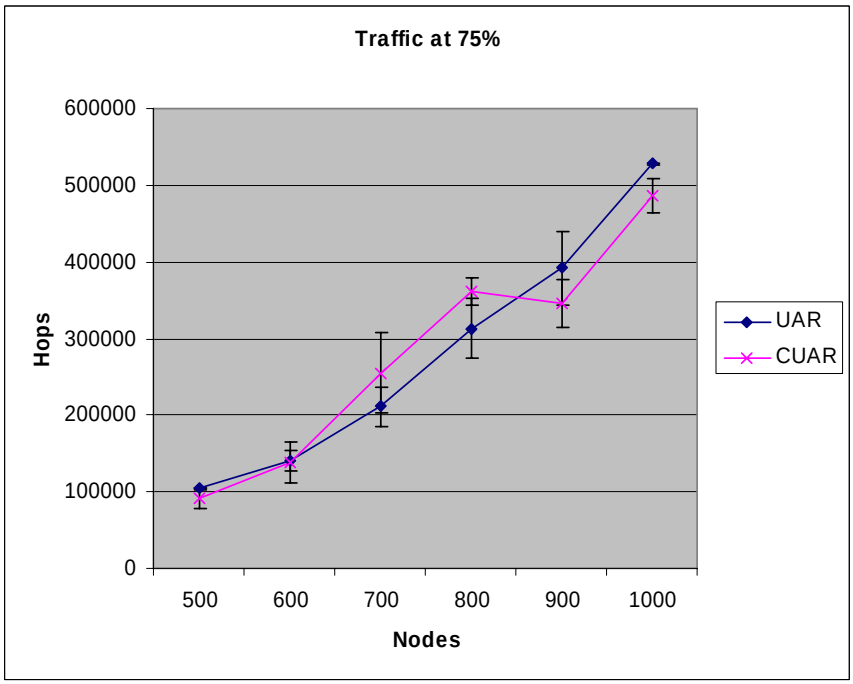


Figure 5.36: **Total traffic for a maintaining a 75% list**: comparisons between different geographic distributions strategies for UAR membership graph and UAR failure detection.

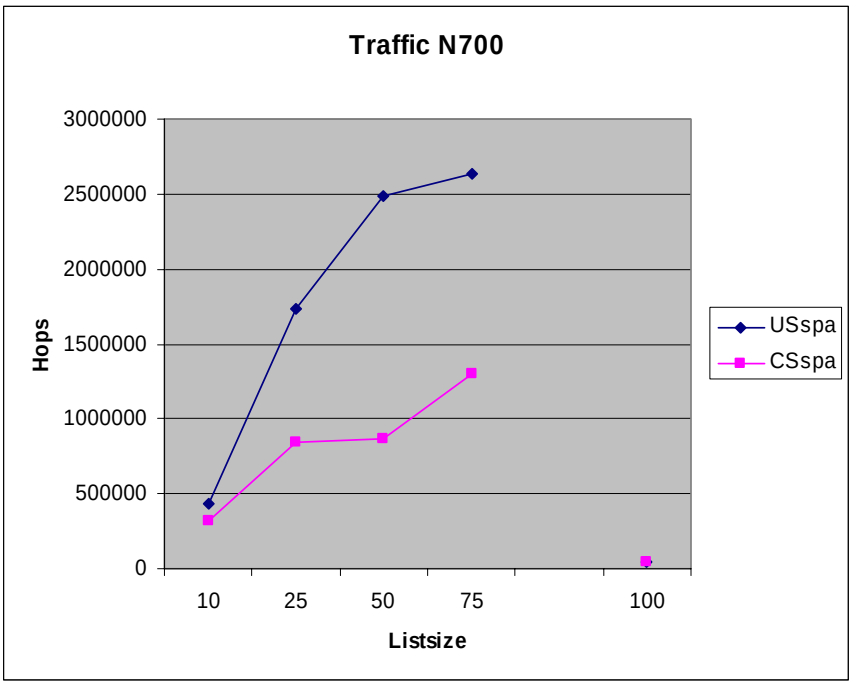


Figure 5.37: **Total traffic for a network size of 700 nodes**: comparisons between different geographic distributions strategies for SPA membership graph and SPA failure detection.

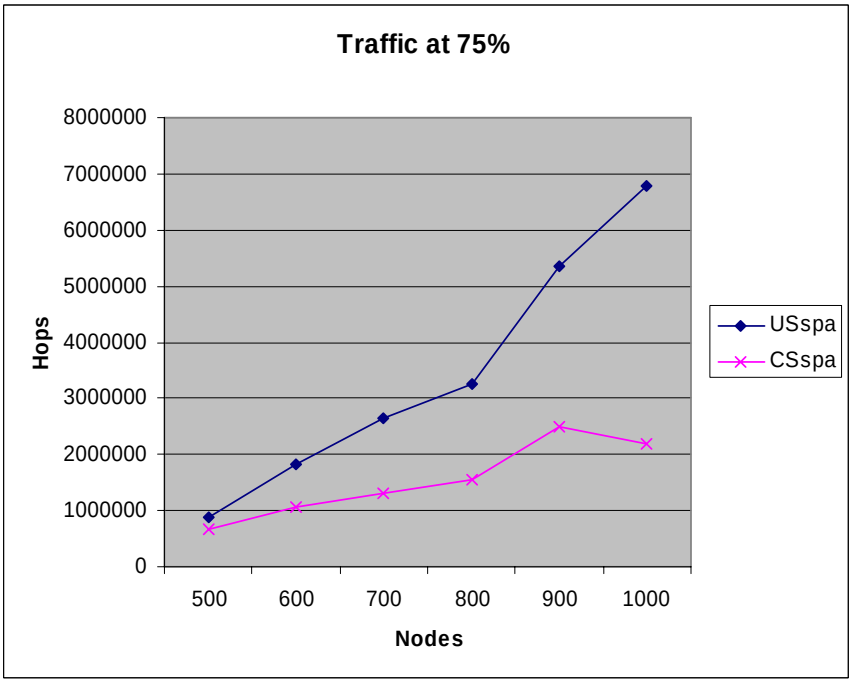


Figure 5.38: **Total traffic for a maintaining a 75% list**: comparisons between different geographic distributions strategies for SPA membership graph and SPA failure detection.

5.3.8 Comparisons: Schemes with CLU Geographic Distribution and HIER Membership Graph

An application with the CLU-HIER requirement has the sensor nodes in a CLU geographic distribution and the membership graph at each node is created in a hierarchical (HIER) manner. Each node in a cluster knows about its leaders and about other nodes in the cluster. A leader knows of all the leaders in the network and about other nodes that are part of its cluster.

Figure 5.39 shows the average number of rounds it takes for various membership protocols to stabilize in a 700 node network. All the failure detection strategies, including SPA, perform equally for this combination of parameters. The reason for SPA performing equal to the other schemes in this setting is that since the membership graph is constructed in a HIER manner, so the farthest node in the membership list of a node would be 'cluster-diameter' hops away. Thus most of the nodes would have an equal probability of being chosen as a ping target.

However, the total traffic for the SPA scheme is still higher than the other schemes as is shown in Figure 5.40. Since the total traffic is a reflection of the maximum number of rounds taken to detect all failures, this means that the SPA scheme's maximum numbered round is higher than the other schemes. This can be attributed to the effect of the leader nodes which have other long distance nodes in their membership lists

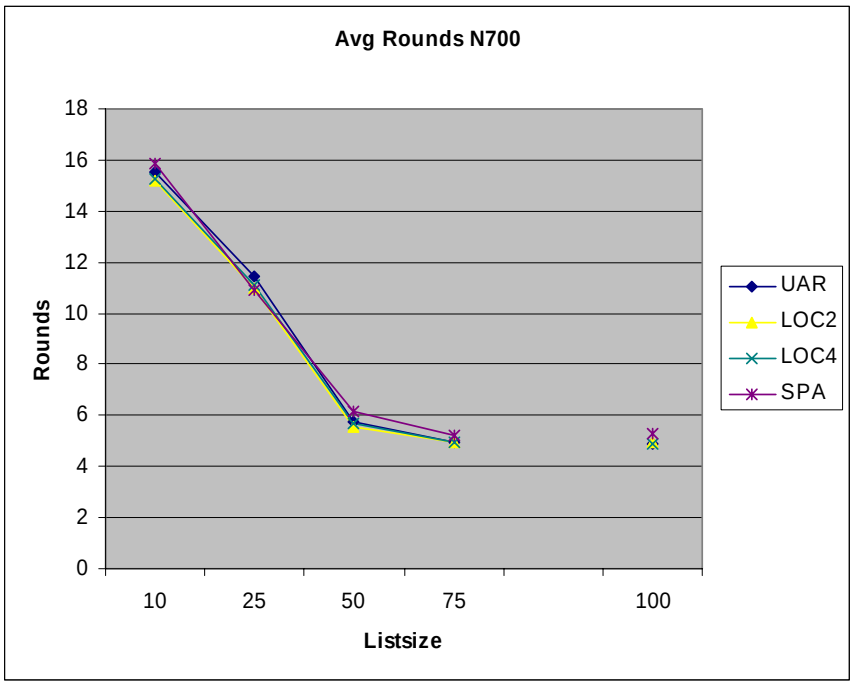


Figure 5.39: **Average number of rounds for a network size of 700 nodes**: comparisons between different failure detection strategies for CLU-HIER.

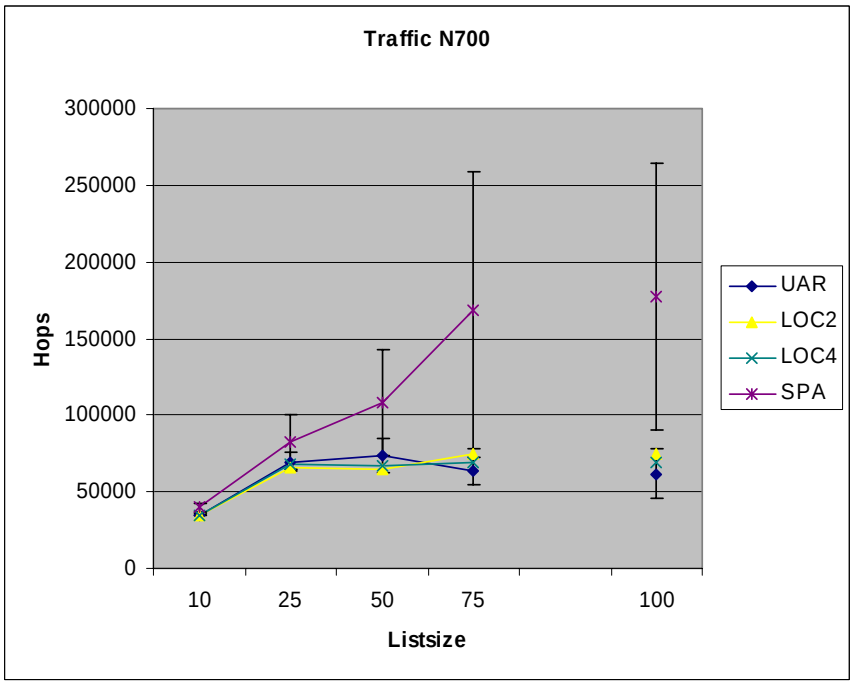


Figure 5.40: **Total traffic for a network size of 700 nodes**: comparisons between different failure detection strategies for CLU-HIER.

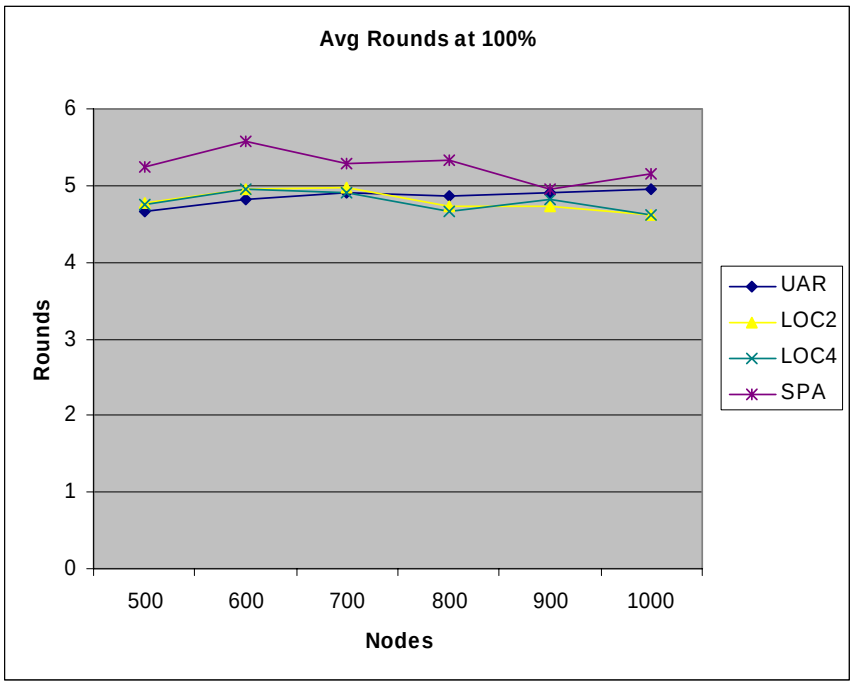


Figure 5.41: **Average number of rounds for maintaining a complete list**: comparisons between different failure detection strategies for CLU-HIER.

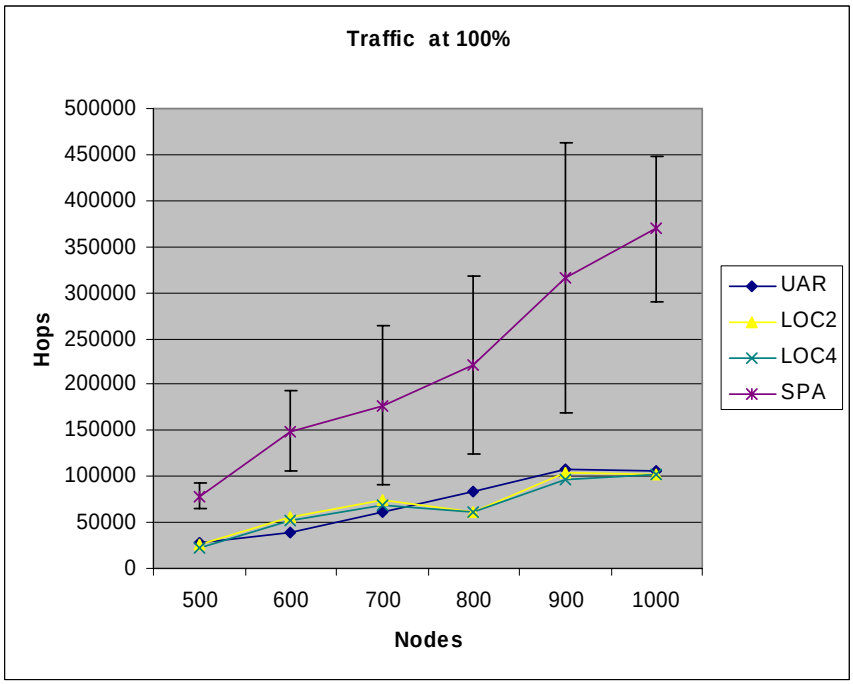


Figure 5.42: **Total traffic for maintaining a complete list**: comparisons between different failure detection strategies for CLU-HIER.

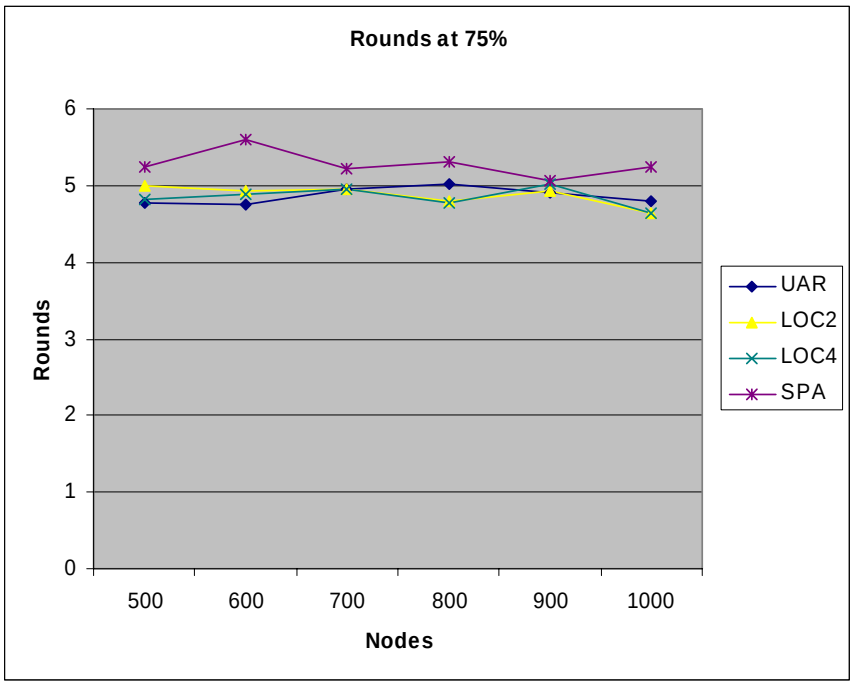


Figure 5.43: **Average number of rounds for maintaining a 75% membership list**: comparisons between different failure detection strategies for CLU-HIER.

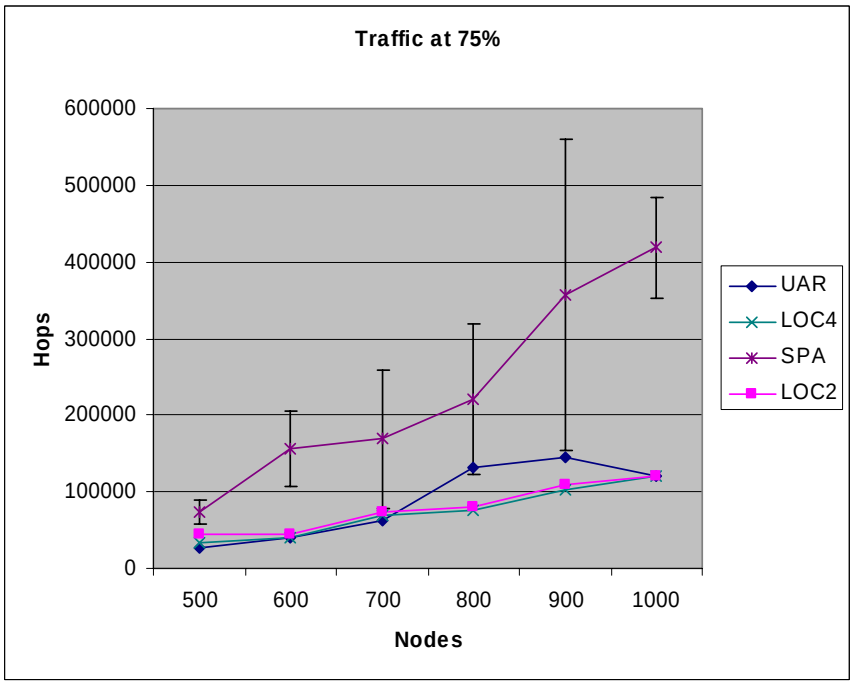


Figure 5.44: **Total traffic for maintaining a 75% membership list**: comparisons between different failure detection strategies for CLU-HIER.

and which take a longer time to be chosen as ping targets. As the list size increases, more and more nodes within the cluster are added to each membership list. This reduces the probability of long distance nodes being chosen even further (due to the normalizing constant) and results in the widening gap between the SPA and other schemes as the list size increases.

Similar behavior can be seen for the average number of rounds and total traffic for list sizes of 100% and 75% as shown in Figures 5.41 – 5.44.

5.3.9 Comparisons: Network Diameter 7 Hops vs. 3 Hops

To measure the effect of membership protocols on small diameter networks, the simulations were also run using a network diameter of 3 hops. This section presents the comparison between the behaviors of the protocols using different network diameters.

As is seen from Figure 5.45, the average number of rounds it takes for a protocol to stabilize are the same for both networks. This is because, as shown in the analysis in Section 4.1, the average number of rounds is independent of the group size. Figures 5.46 and 5.47 show the difference in total traffic at list sizes of 100% and 50% respectively. The traffic, as expected, is higher in the network with the larger diameter and is almost proportional to the ratio of the diameters.

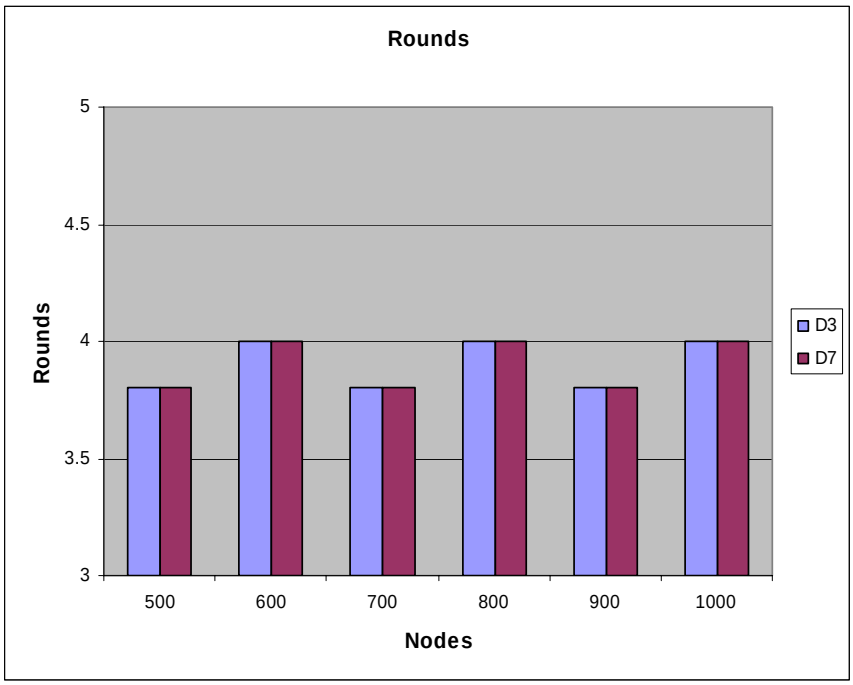


Figure 5.45: **Average number of rounds for UUuar at list size of 100%**: comparisons between different network diameters.

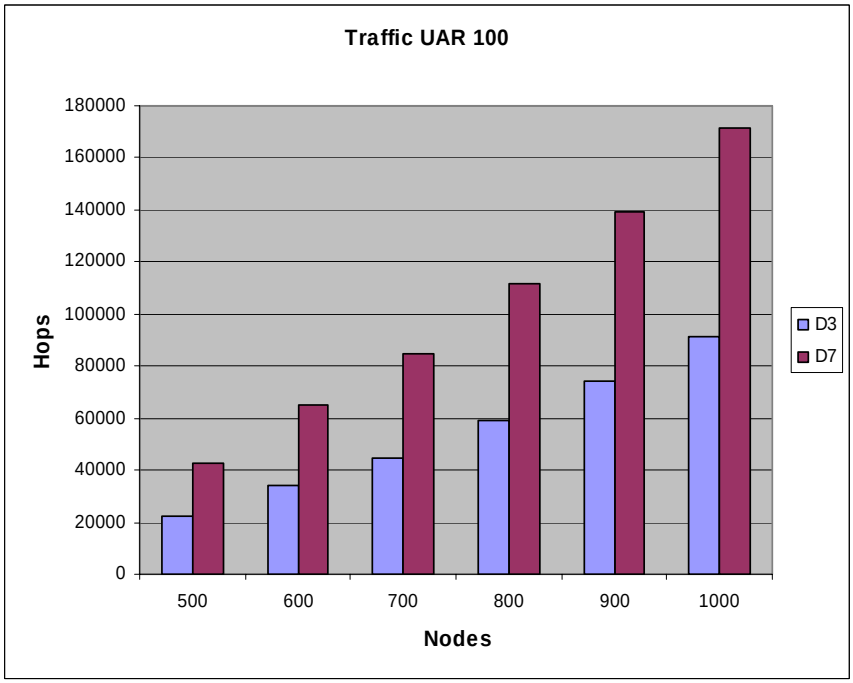


Figure 5.46: **Total traffic for UUuar at list size of 100%**: comparisons between different network diameters.

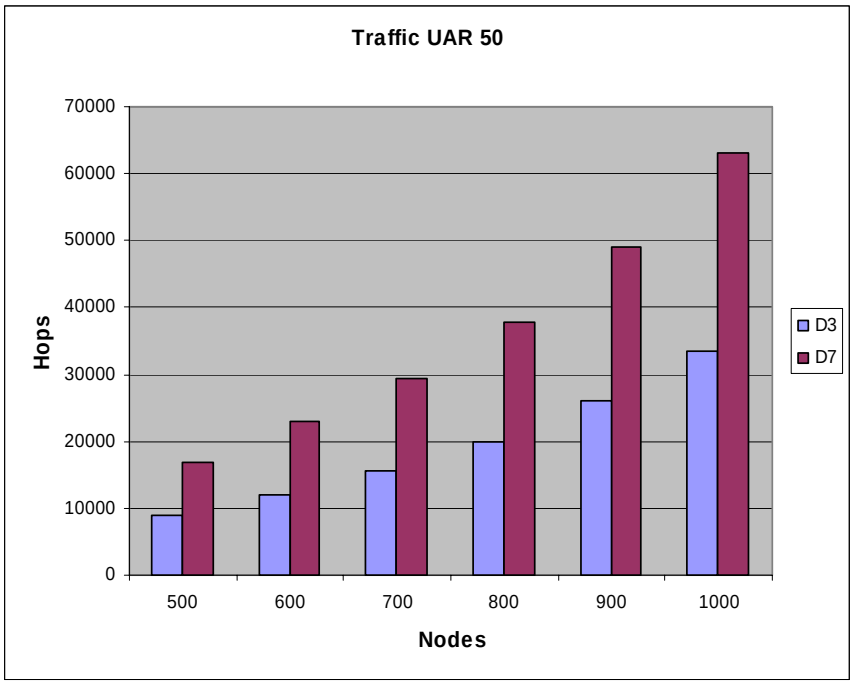


Figure 5.47: **Total traffic for UUuar at list size of 50%**: comparisons between different network diameters.

5.3.10 Comparisons: Other Schemes

Most of the other application membership requirements display a behavior similar to the application requirements evaluated above. The behavioral and comparative plots for all the other combinations of application requirements and membership protocol strategies (Figure 5.2) are attached as Appendix A.

The only combination which differs from the above schemes is the CLU-HIER scheme and has been evaluated in the section 5.3.6.

6 CONCLUSION

Wireless sensor networks are becoming more and more useful these days and considerable research is being done to further extend the fields of application to which wireless sensors can be put to. As sensor network applications grow in complexity, so would a need to move from the current build-from-scratch approach to application development to an abstractions based approach which makes development easier and more manageable.

As sensor network applications move to leverage the collaborative advantages of having a multitude of processors, a membership management service would be most useful to take the burden off the application developers. To this end, this thesis presented and evaluated a membership substrate for wireless sensor networks. We implemented the membership substrate as a nesC [3] library that can be used for application development on sensor networks. We also evaluated the membership requirements that applications have and analyzed different membership management schemes. We then saw how those requirements matched up with membership protocols, using our own simulator developed in C to simulate the behavior of different failure detection strategies for various geographical distributions and membership models, and explored their tradeoffs with respect to network traffic and completeness in failure detection. Finally, based on our experiments, we make some recommendations for application developers who would want to use or develop a membership substrate for wireless sensor networks.

6.1 Recommendations to Application Developers

In the previous chapter, we laid out certain 'rules of thumb', which were common sense, intuitive guidelines regarding choices that could be made while mixing and matching different application properties with membership protocols. Based on our experimental observations, we expand on those 'rules of thumb' to make some recommendations to application developers;

I. The UAR and the SPA failure detection protocols ensure completeness (i.e. detect all failures at all non-faulty nodes) regardless of the combination of membership model and the geographic distribution of the application.

II. The SPA based membership protocols generally perform worse than UAR based protocols when a partial membership list is maintained. Both have similar performance when a complete membership list is maintained.

III. Using a Broadcast-TTL1 update dissemination strategy provides a good tradeoff of traffic for an increased number of rounds. Compared to the Broadcast-TTL∞ dissemination scheme, Broadcast-TTL1 generates significantly less traffic to spread updates though the average number of rounds it takes to converge goes up marginally.

IV. When using a Broadcast-TTL1 dissemination scheme, in terms of traffic, it makes sense to maintain either a complete list or a small partial membership list. As the list size of the partial membership list increases, the traffic also correspondingly goes up, making moderate list sizes perform badly.

V. The LOC failure detection scheme can ensure completeness only if the ping radius is greater than or equal to the membership graph radius and if a complete membership list is maintained. For the LOC failure detection scheme, in case partial membership lists are maintained, there always is a non-zero probability that a failed node might be isolated in a manner that it would not be in any other node's membership list and thus, would never be detected as having failed.

VI. The problems associated with the LOC scheme's lack of completeness can be remedied, to an extent, by using a Broadcast-TTL∞ scheme for disseminating updates, but this would not work in case a node is isolated.

6.2 Future Work

Wireless sensor networks are an emerging field and there is immense scope for further research in terms of identifying, evaluating and implementing various services that sensor network applications could require. As part of future work on membership protocols for wireless sensor networks, more membership models can be identified and evaluated. Optimizations on such membership models can also be studied. More particularly, application-specific membership models can be identified and evaluated, as well as studied with respect to optimization techniques such as piggybacking of pings on membership updates. Another area of further research could be the development of efficient membership protocols and especially adaptive membership protocols in which the application could specify its requirements to the protocol through a utility function or a service contract. Lastly, hybrid membership management protocols also bear

investigation. Such protocols could combine two or more of the proposed membership protocols based on application specified parameters or could adaptively use a combination of such protocols in response to changes in the network or the environment.

REFERENCES

[1] R. Want, A. Hopper, V. Falcao and J. Gibbons, "The active badge location system," In *ACM Transaction on Office Information Systems*, Vol. 10. No. 1, pp. 91-102, January 1992.

[2] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," In *Proceedings of ASPLOS*, October 2002.

[3] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler, "The nesC language: A holistic approach to networked embedded systems," In *Proceedings Programming Language Design and Implementation (PLDI)*, June 2003.

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler and K. S. Pister, "System architecture directions for networked sensors," In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 93-104, Boston, November 2000.

[5] P. Levis, N. Lee, M. Welsh and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[6] M. K. Aguilera, W. Chen and S. Toueg, "Heartbeat: a timeout-free failure detector for quiescent reliable communication," In *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG'97)*, pp. 126-140, September 1997.

[7] C. Almeida and P. Verissimo, "Timing failure detection and real-time group communication in real-time systems," In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, June 1996.

[8] R. van Renesse, Y. Minsky and M. Hayden, "A gossip-style failure detection service," *In Proceedings of the International Conference and Distributed Systems Platforms and Open Distributed Processing (IFIP)*, September 1998.

[9] I. Gupta, T. D. Chandra and G. S. Goldszmidt, "On scalable and efficient distributed failure detectors," In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 170–179, August 2001.

[10] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM*, 43(2): pp. 225-267, March 1996.

[11] A. Das, I. Gupta and A. Motivala, "SWIM: Scalable Weakly-Consistent Infection-Style Process Group Membership Protocol," In *Proceedings of The International Conference on Dependable Systems and Networks*, pp. 303-312, June 2002

[12] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko and D. Rus, "Tracking a Moving Object with a Binary Sensor Network," In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[13] S. Kumar, C. Alaettinglu and D. Estrin, "Scalable object-tracking through unattended techniques (SCOUT)," In *Proceedings of the 2000 International Conference on Network Protocols*, pp. 253, November 2000

[14] V. Mehta and M. El Zarki, "Fixed Sensor Networks for Civil Infrastructure Monitoring – An Initial Study," *First Annual Mediterranean Ad Hoc Networking Workshop, Med-hoc-Net 2002*, Sardegna, Italy, September 2002.

[15] A. Elgamal, J. P. Conte, L. Yan and M. Fraser, "A Framework for Monitoring Bridges and Civil Infrastructure," In *Proceedings of the 3rd China-Japan-US Symposium on Structural Health Monitoring and Control*, Dalian, China, October 2004.

[16] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson, "Wireless sensor networks for habitat monitoring," In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 88–97, Atlanta, September 2002.

[17] K.S. Pister, "Tracking vehicles with a uav-delivered sensor network," March 2001. [Online]. Available: http://robotics.eecs.berkeley.edu/~pister/29Palms0103/

[18] V. A. Kottapalli, A. S. Kiremidjian, J. P. Lynch, E. Carryer, T. W. Kenny, K. H. Law and Y. Lei, "Two-tiered wireless sensor network architecture for structural health monitoring," In *Proceedings of the SPIE 10th Annual International Symposium on Smart Structures and Materials*, San Diego, CA, March 2000.

[19] Center for Information Technology Research in the Interest of Society, "Smart buildings admit their faults," 2002. [Online]. Available: http://www.citris.berkeley.edu/applications/disaster_response/smartbuildings.html

[20] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao, "Habitat monitoring: Application driver for wireless communications technology," In *Proceedings of the Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[21] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," In *Proceedings of International Conference on Mobile Computing and Networking*, August 2000.

[22] S. Hedetniemi, S. Hedetniemi and A. Liestman, "A survey of gossiping and broadcasting in communication networks," Networks 18 (4) (1988) pp. 319–349.

[23] N. T. J. Bailey, *Mathematical theory of infectious diseases and its applications*, 2nd Edition, Charles Griffin and Company Ltd.

[24] D. Kempe, J. Kleinberg and A. Demers, "Spatial Gossip and Resource Location Protocols," In *Proceedings of the 33rd ACM Symposium on the Theory of Computing (STOC)*, July 2001

[25] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," In *Proceedings of MOBICOM*, pp. 263-270, Seattle, 1999.

[26] M. Welsh and G. Mainland, "Programming Sensor Networks Using Abstract Regions," In *Proceedings of NSDI*, pp. 29-42, San Francisco, March 2004

[27] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui and B. Krogh, "Energy-Efficient Surveillance System Using Wireless Sensor Networks," In *Proceedings of MOBISYS*, Boston, June 2004.

[28] "CrossBow Mica2 data sheet," Crossbow Technology Inc., San Jose, CA. [Online]. Available: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-05_A_MICA2.pdf

[29] "CrossBow Micaz data sheet," Crossbow Technology Inc., San Jose, CA. [Online]. Available: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0060-01_A_MICAz.pdf

[30] "CrossBow Mica2Dot data sheet," Crossbow Technology Inc., San Jose, CA. [Online]. Available: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0043-04_A_MICA2DOT.pdf.

[31] B. Blum, P. Nagaraddi, A. Wood, T. Abdelzaher, S. Son and J. Stankovic, "An Entity Maintenance and Connection Service for Sensor Networks," In *Proceedings of MobiSys 2003: The First International Conference Mobile Systems, Applications, and Services*, San Francisco, May 2003.

[32] J. M. Kahn, R. H. Katz and K. S. J. Pister, "Next Century Challenges: Mobile Networking for Smart Dust," In *Proceedings of 5th ACM/IEEE International Conference on Mobile Computing*, September 1999.

[33] P. Levis, N. Patel, D. Culler and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," In *Proceedings of NSDI*, San Francisco, March 2004

[34] P. Bauer, M. Sichitiu, R. Istepanian and K. Premaratne, "The mobile patient: wireless distributed sensor networks for patient monitoring and care," In *Proceedings 2000 IEEE EMBS International Conference on Information Technology Applications in Biomedicine*, pp. 17–21, 2000.

[35] D. Malan, T. Fulford-Jones, M. Welsh and S. Moulton, "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care," *International Workshop on Wearable and Implantable Body Sensor Networks*, April 2004.

[36] B. Warneke, B. Liebowitz, K. S. J. Pister, "Smart dust:communicating with a cubic-millimeter computer," *IEEE Computer*, pp. 2-9, January 2001.

[37] P. Bonnet, J. Gehrke and P. Seshadri, "Querying the physical world," *IEEE Personal Communications*, pp. 10-15, October 2000.

[38] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," In *Proceedings of IEEE Infocom*, San Francisco, April 2003.

[39] S. Ray, R. Ungrangsi, F. De Pellegrini, A. Trachtenberg and D. Starobinski, "Robust location detection in emergency sensor networks," In *Proceedings of IEEE Infocom*, San Francisco, April 2003.

[40] S. Meguerdichian, S. Slijepcevic, V. Karayan and M. Potkonjak, "Localized algorithms in wireless ad-hoc networks: location discovery and sensor exposure," *ACM Mobihoc*, Long Beach, October 2001.

[41] S. R. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks," In *OSDI*, Boston, December 2002.

[42] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," In *SOSP*, October 2001.

[43] D. Braginsky and D. Estrin, "Rumor routing algorithm for sensor networks," In *WSNA*, September 2002.

[44] R. Govindan, W. Hong, S. Madden, M. Franklin and S. Shenker, “The sensor network as a database,” USC Technical Report No. 02-771, September 2002.

[45] B. Karp and H.T. Kung, “GPSR: Greedy Perimeter Stateless Routing for wireless networks,” In *MOBICOM*, Boston, August 2000.

[46] R. Govindan, B. Karp, S. Shenker, L. Yin, F. Yu, S. Ratnasamy and Deborah Estrin, “Data-centric storage in sensornets,” In *WSNA*, September 2002.

[47] R. C. Shah and J. Rabaey, "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks," *IEEE Wireless Communications and Networking Conference (WCNC)*, Orlando, March 2002.

[48] J. Kulik, W. Heinzelman and H. Balakrishnan, “Negotiation-based protocols for disseminating information in wireless sensor networks,” *Wireless Networks*, March 2002.

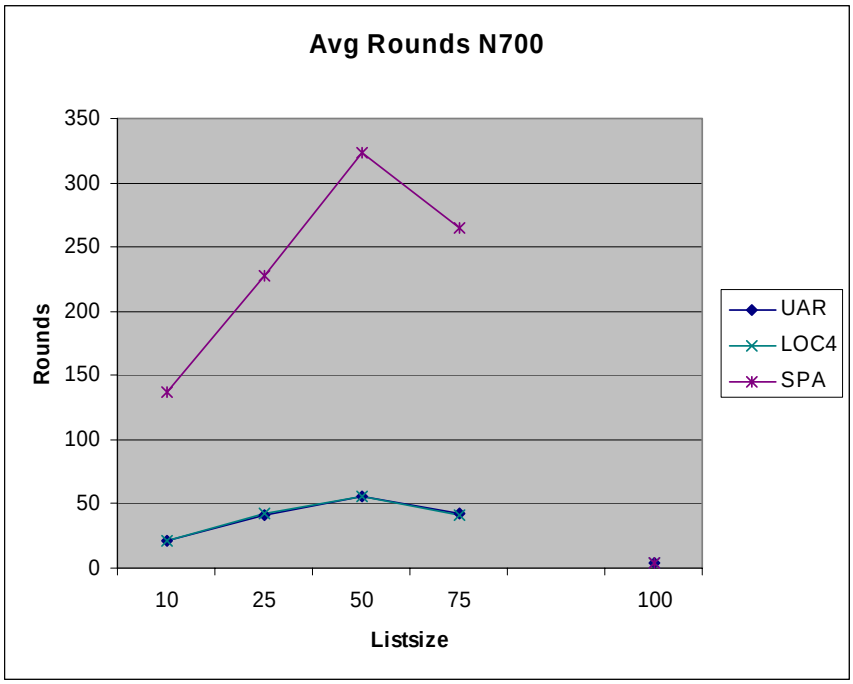
[49] W.R. Heinzelman, J. Kulik and H. Balakrishnan, “Adaptive protocols for information dissemination in wireless sensor networks,” In *Proceedings of the ACM MobiCom’99*, pp. 174–185, Seattle, 1999.

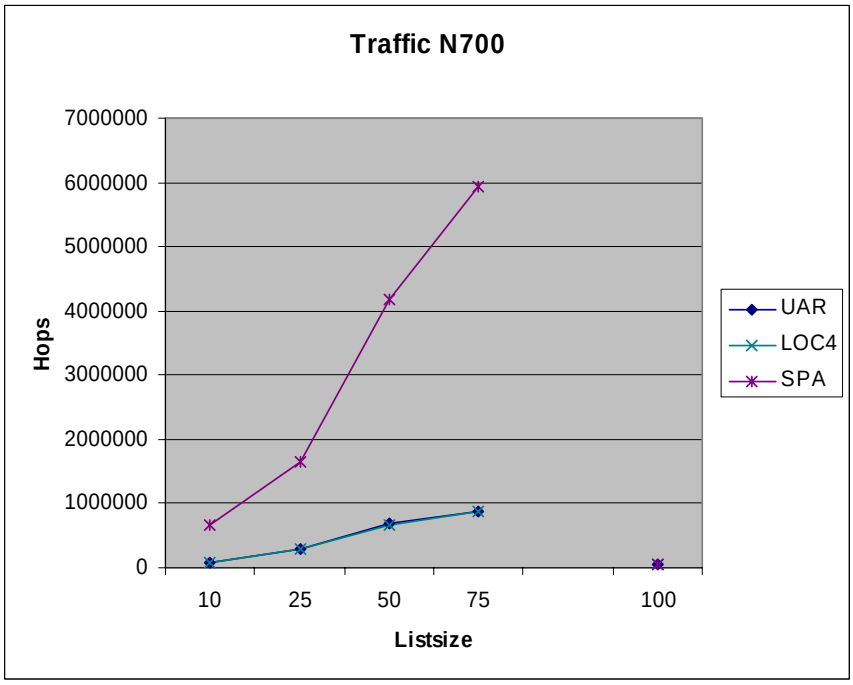
[50] Y. Yao and J. E. Gehrke, “The Cougar approach to in-network query processing in sensor networks,” *ACM Sigmod* Record, 31(3), September 2002.

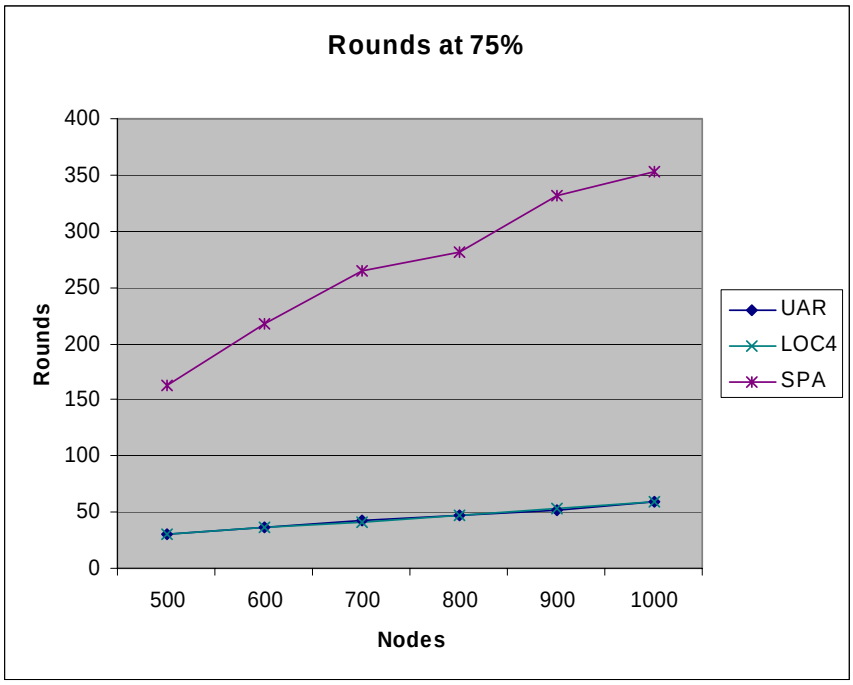
[51] S. Madden, M. Franklin, J. Hellerstein and W. Hong, “The design of an acquisitional query processor for sensor networks,” In *Proceedings of ACM SIGMOD*, San Diego, June 2003.

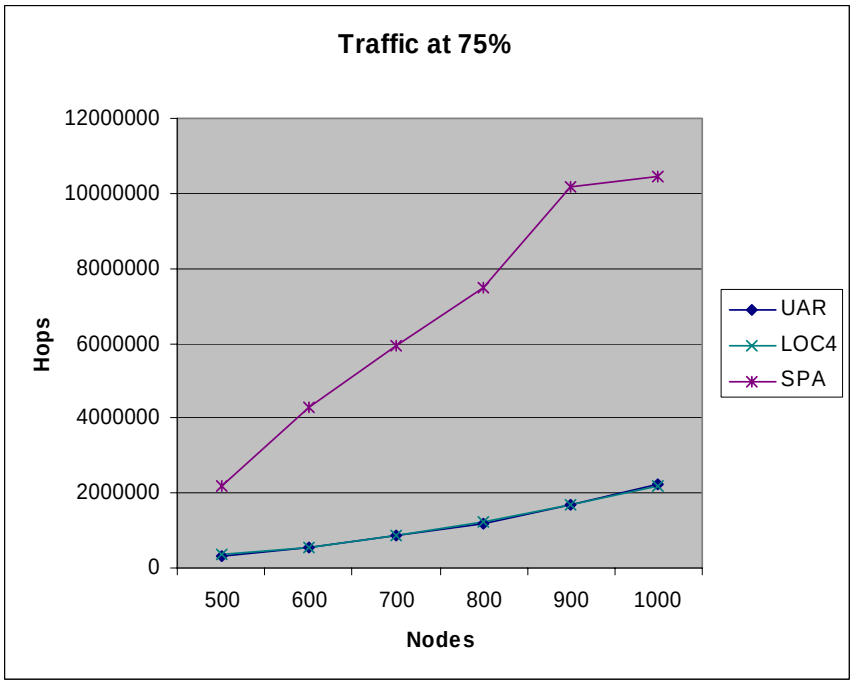
APPENDIX A : COMPARATIVE PLOTS

Plots for UAR Geographical Distribution and LOC4 Membership Graph

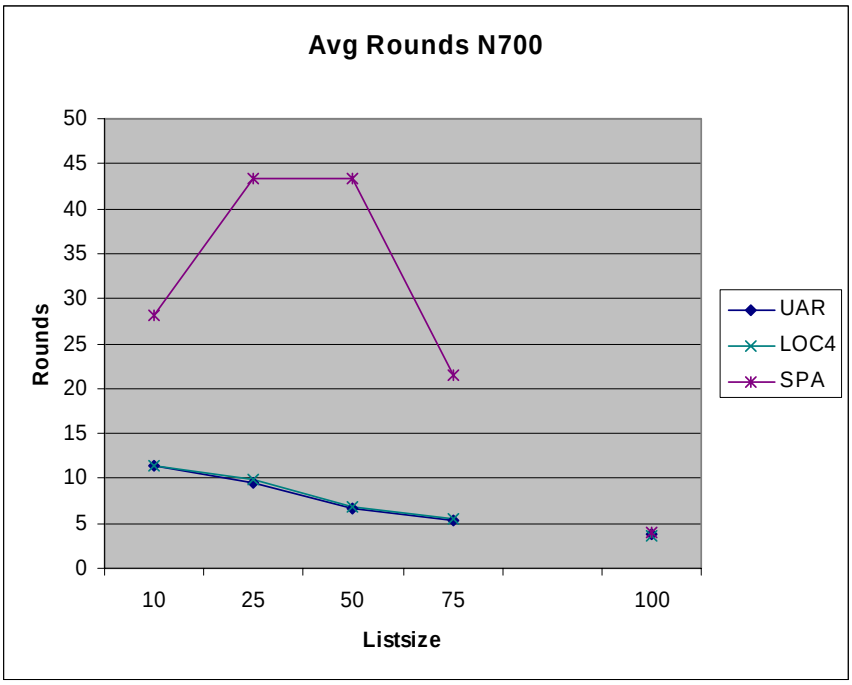


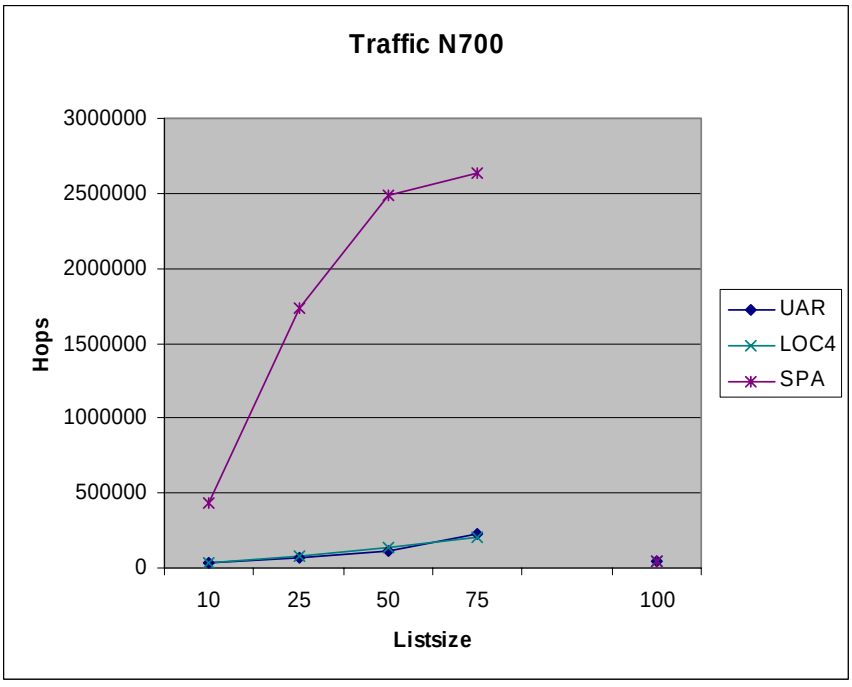


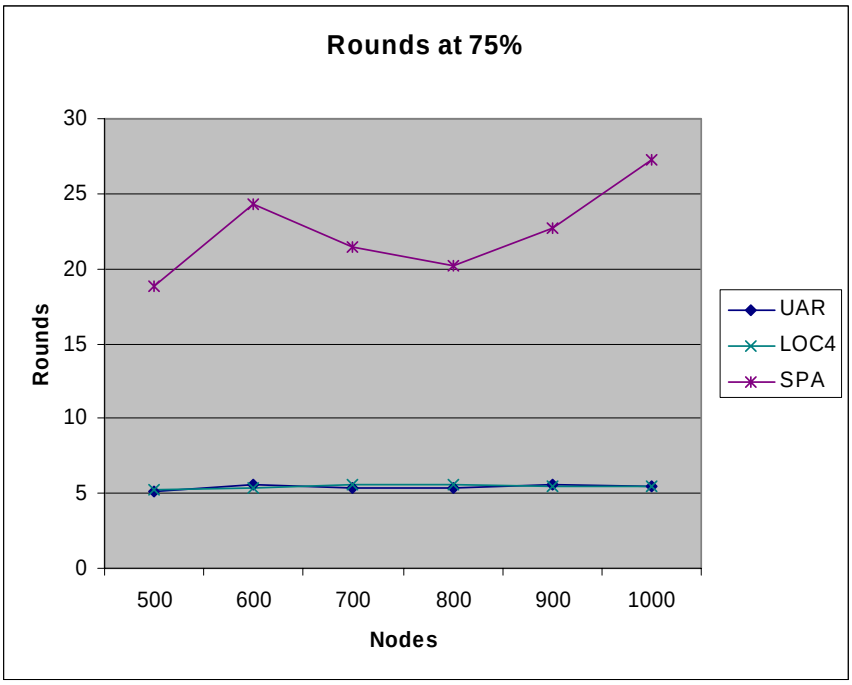

Rounds at 75%
400
350
300
250
200
150
100
50
0
Rounds
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

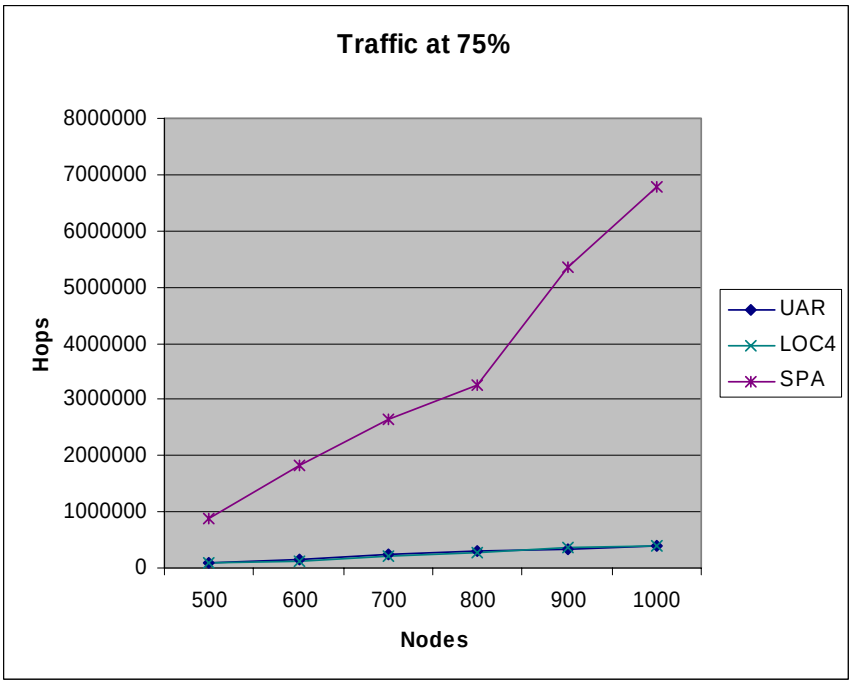

Traffic at 75%
12000000
10000000
8000000
6000000
4000000
2000000
0
Hops
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

Plots for UAR Geographical Distribution and SPA Membership Graph

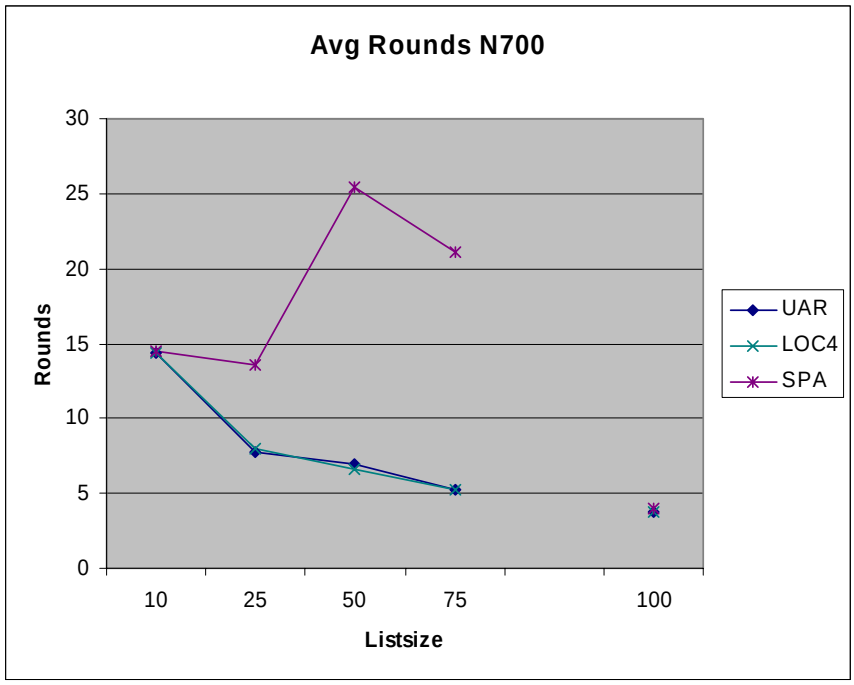


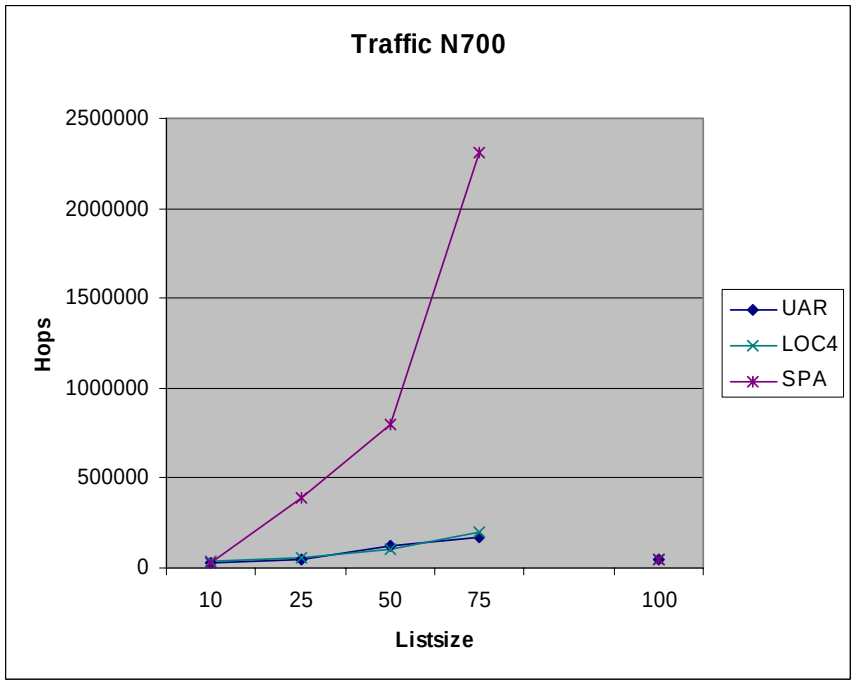


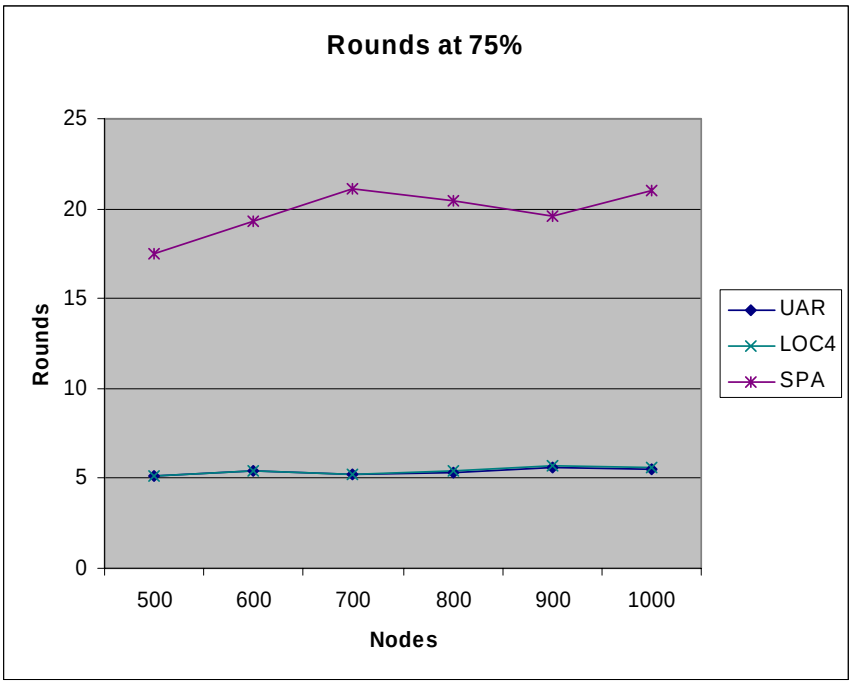

Rounds at 75%
30
25
20
15
10
5
0
Rounds
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

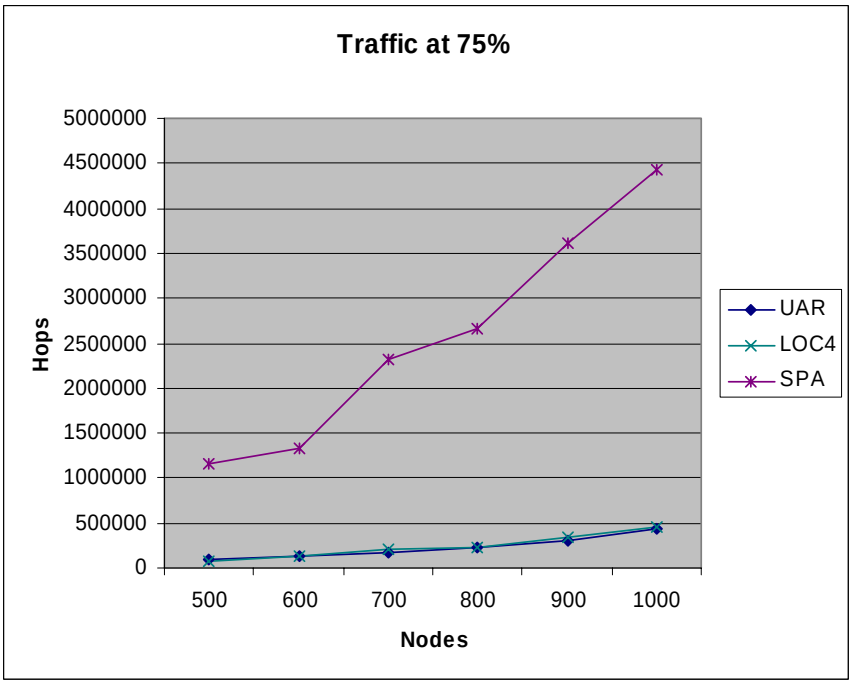

Traffic at 75%
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
Hops
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

Plots for UAR Geographical Distribution and KLOC Membership Graph

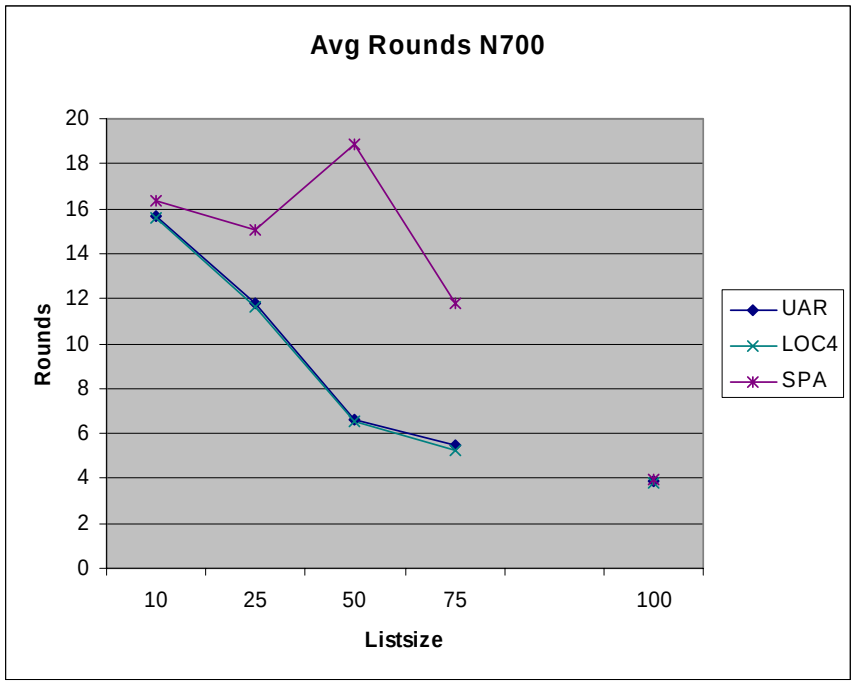


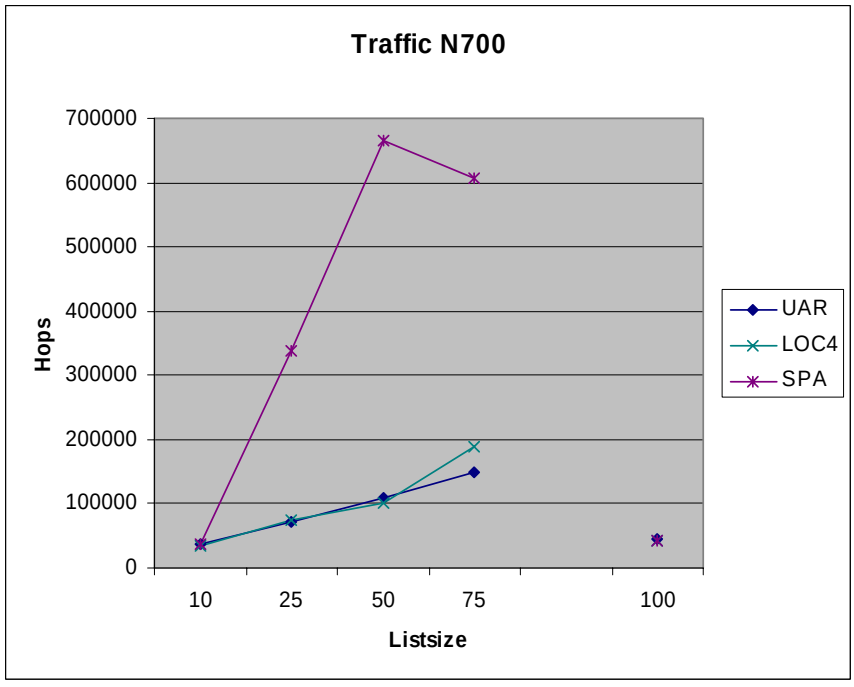



Rounds at 75%
25
20
15
10
5
0
Rounds
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

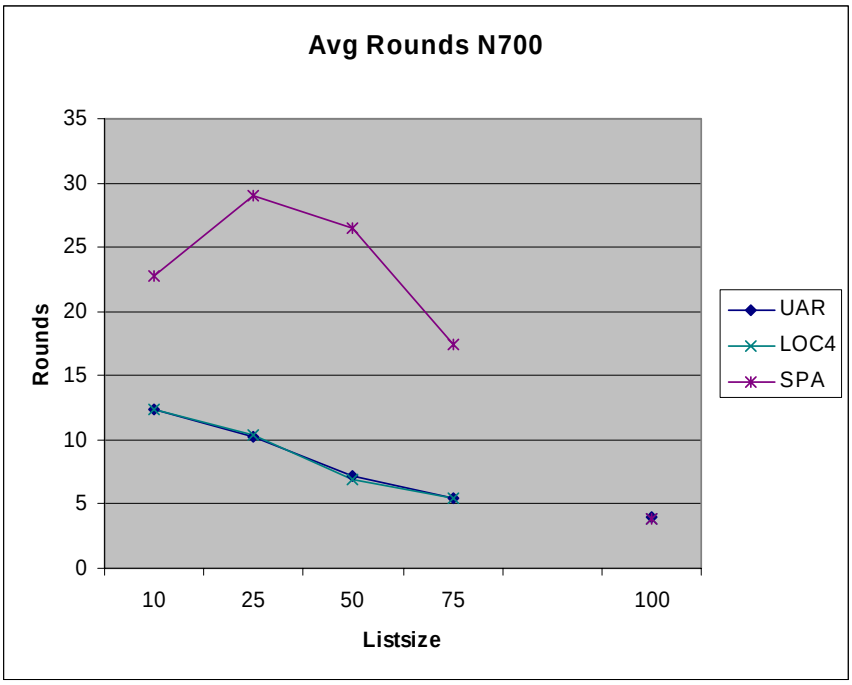

Traffic at 75%
5000000
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
Hops
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

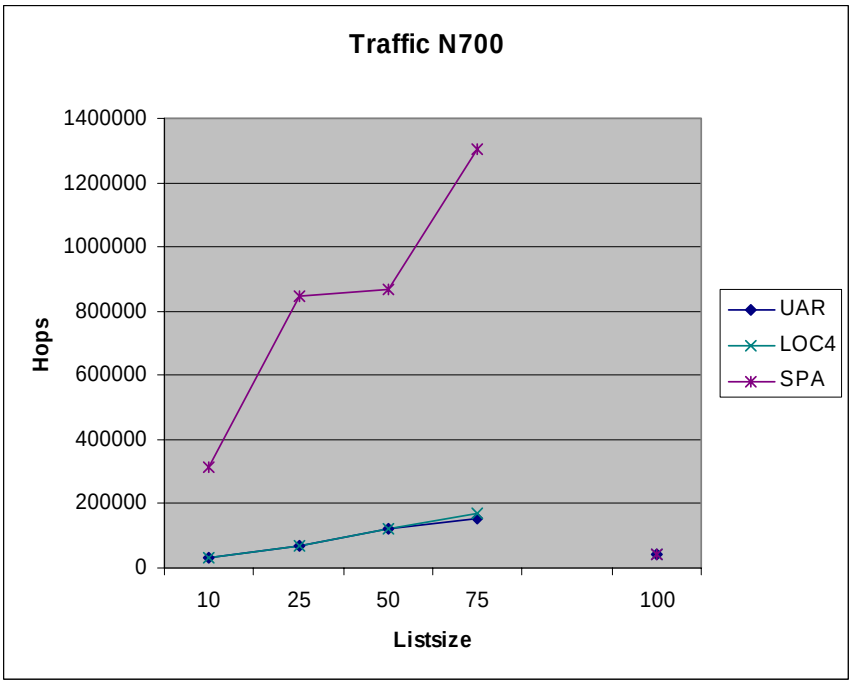
Plots for CLU Geographical Distribution and KLOC Membership Graph

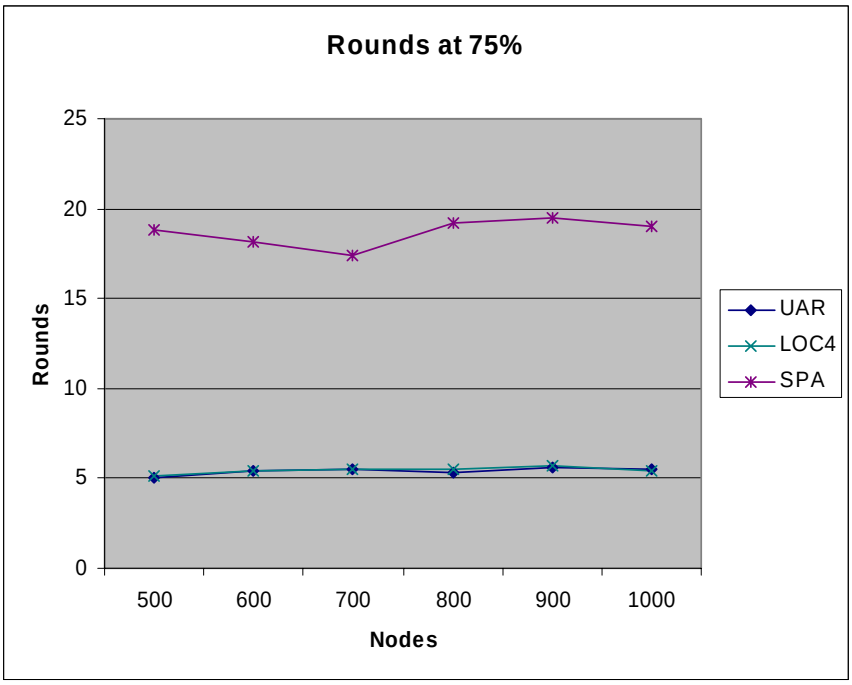


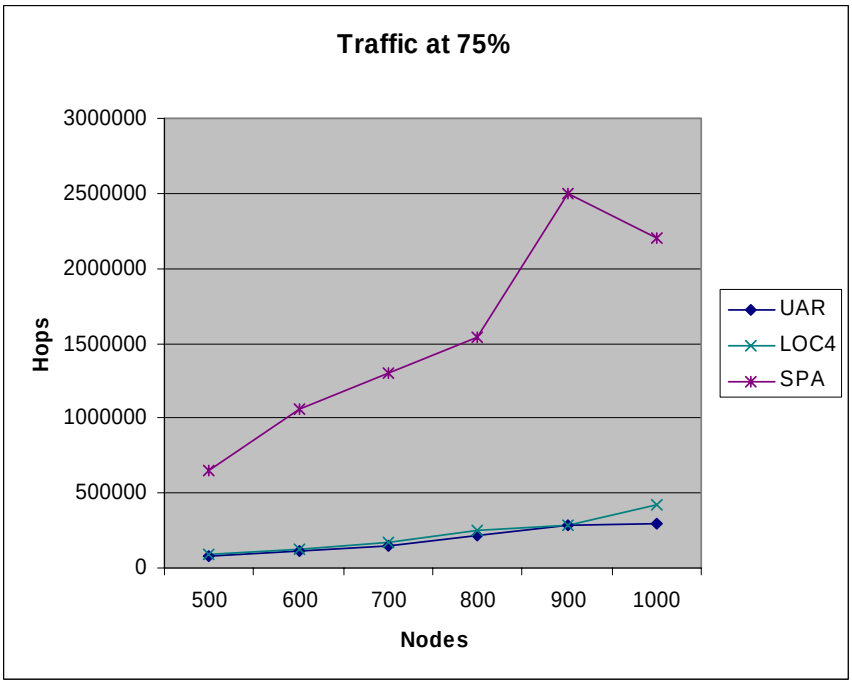


Plots for CLU Geographical Distribution and SPA Membership Graph

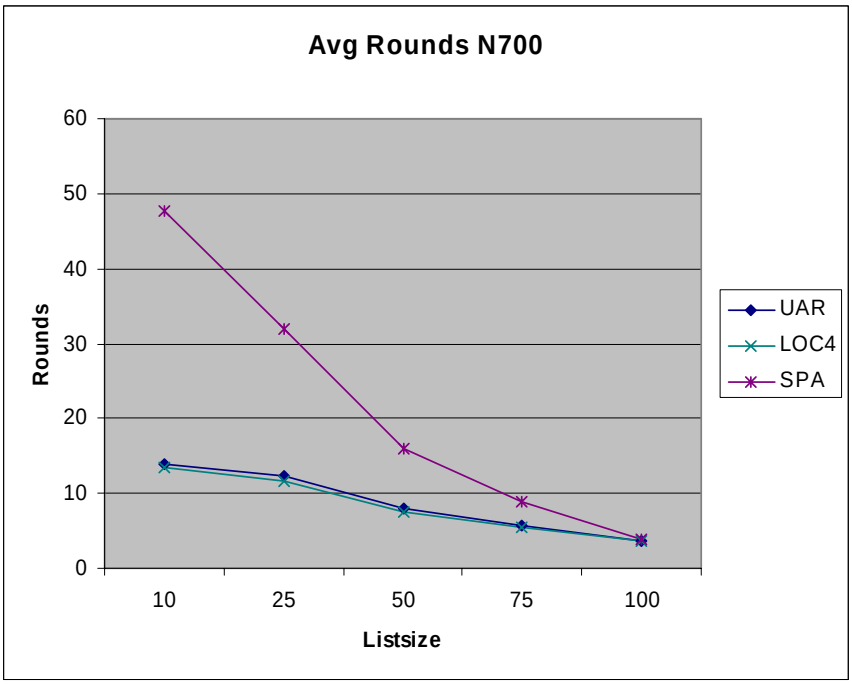


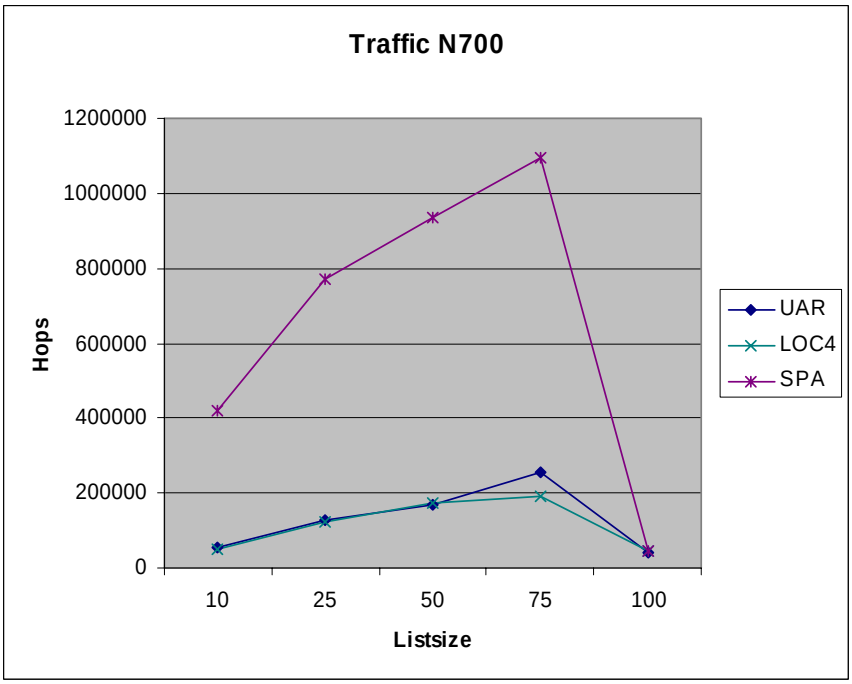


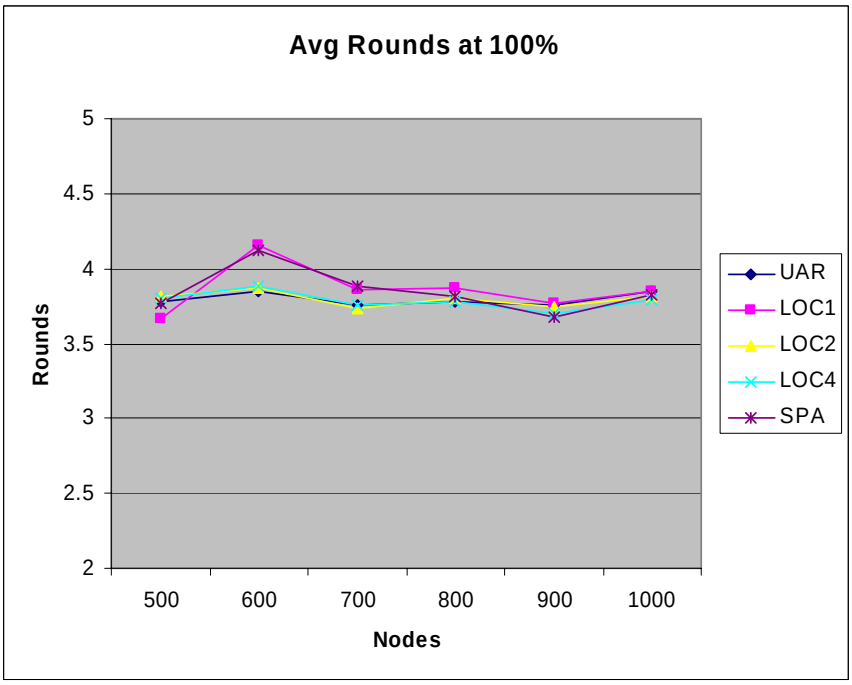

Rounds at 75%
25
20
15
10
5
0
Rounds
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

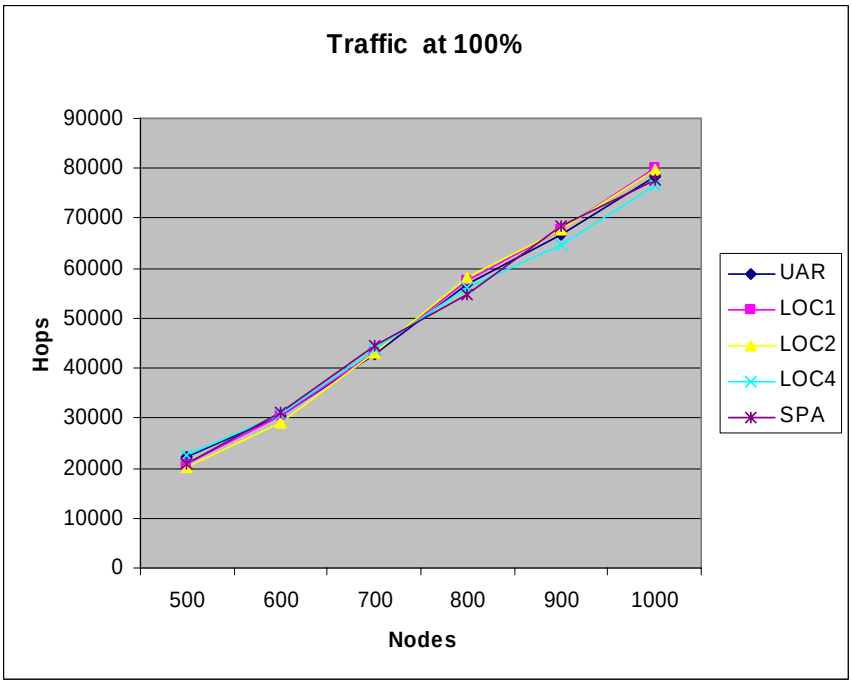

Traffic at 75%
3000000
2500000
2000000
1500000
1000000
500000
0
Hops
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

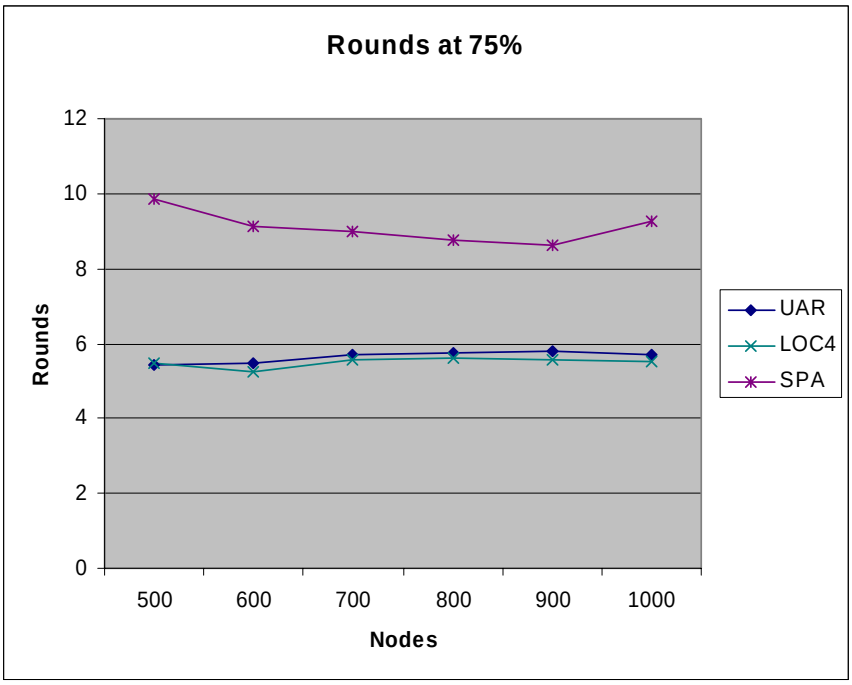
Plots for CLU Geographical Distribution and UAR Membership Graph

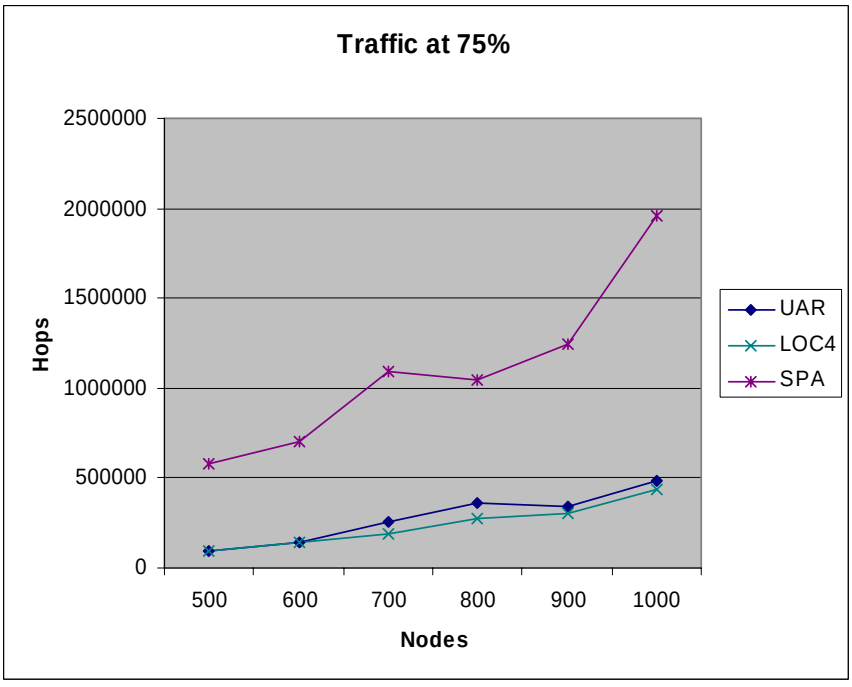





Avg Rounds at 100%
Rounds
5
4.5
4
3.5
3
2.5
2
500
600
700
800
900
1000
Nodes
UAR
LOC1
LOC2
LOC4
SPA


Traffic at 100%
Hops
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
500
600
700
800
900
1000
Nodes
UAR
LOC1
LOC2
LOC4
SPA


Rounds at 75%
12
10
8
6
4
2
0
Rounds
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA


Traffic at 75%
2500000
2000000
1500000
1000000
500000
0
Hops
500
600
700
800
900
1000
Nodes
UAR
LOC4
SPA

APPENDIX B : TOSSIM CODE

Filename : FadetTest.nc

```
configuration FadetTest {
}

implementation {
   components WSN_Main,
              FadetTestM,
              FadetAODV as Fadet,
              TimerC;

   FadetTestM.TimerControl -> TimerC;

   FadetTestM.Control <- WSN_Main;

   FadetTestM.TRControl -> Fadet;
}
```

Filename : FadetTestM.nc

```
includes WSN;

module FadetTestM {
   provides {
      interface StdControl as Control;
   }
   uses {
      interface StdControl as TRControl;
      interface StdControl as TimerControl;
   }
}

implementation {
   command result_t Control.init() {
      call TimerControl.init();
      call TRControl.init();

      return SUCCESS;
   }

   command result_t Control.start() {
      call TimerControl.start();
```

```
      call TRControl.start();

      return SUCCESS;
   }

   command result_t Control.stop() {
      call TimerControl.stop();
      call TRControl.stop();
      return SUCCESS;
   }
}
```

Filename : FadetAODV.nc

```
includes WSN;
includes WSN_Messages;
includes WSN_Settings;

configuration FadetAODV {
   provides {
      interface StdControl as Control;
      interface Intercept;
   }
   uses {
      interface Piggyback;
   }
}

implementation {
   components Fadet_AODVM,
              TimerC, RandomLFSR,
#if SINK_NODE
              UART_Gateway as UART,
#else
              NoUART as UART,
#endif
       AODV as RoutingLayer,
       LedsC, Photo;

   Control = Fadet_AODVM.Control;
   Piggyback = Fadet_AODVM.Piggyback;
   Intercept = Fadet_AODVM.Intercept;

   Fadet_AODVM.Random -> RandomLFSR;
   Fadet_AODVM.Timer->TimerC.Timer[unique("Timer")];
   Fadet_AODVM.MHopControl -> RoutingLayer;
   Fadet_AODVM.Send -> RoutingLayer.Send[APP_ID_FADETPING];
   Fadet_AODVM.Send -> RoutingLayer.Send[APP_ID_FADETACK];


   Fadet_AODVM.SendMHopMsgPing ->
RoutingLayer.SendMHopMsg[APP_ID_FADETPING];
   Fadet_AODVM.ReceivePing -> RoutingLayer.Receive[APP_ID_FADETPING];
```

```
   Fadet_AODVM.SendMHopMsgAck ->
RoutingLayer.SendMHopMsg[APP_ID_FADETACK];
   Fadet_AODVM.ReceiveAck -> RoutingLayer.Receive[APP_ID_FADETACK];

   Fadet_AODVM.MultiHopIntercept ->
RoutingLayer.Intercept[APP_ID_FADETPING];
   Fadet_AODVM.MultiHopIntercept ->
RoutingLayer.Intercept[APP_ID_FADETACK];
   Fadet_AODVM.UARTControl -> UART;
   Fadet_AODVM.UARTSend -> UART;
   Fadet_AODVM.MultiHopMsg -> RoutingLayer;
   Fadet_AODVM.SingleHopMsg -> RoutingLayer;
   Fadet_AODVM.Leds -> LedsC;
   Fadet_AODVM.ADC -> Photo;
   Fadet_AODVM.ADCControl -> Photo;
}
```

Filename : FadetAODVM.nc

```
/**
 * WARNING: This module packs addresses into 8 bytes.  Be careful when
 * using 16-byte addressing.
 */

#ifndef TR_PIGGYBACK_LEN
#define TR_PIGGYBACK_LEN 0
#endif

#ifndef TR_PLUGIN_LEN
#define TR_PLUGIN_LEN 0
#endif

#ifndef TR_SEND_RATE
#if PLATFORM_PC
#define TR_SEND_RATE 70
#else
#define TR_SEND_RATE 30
#endif
#endif

#define ACK_NUM_TRIES 5
#define PING_NUM_TRIES 3

#define FADET_MAX_MEM 5
#define FADET_FAIL 2
#define FADET_ROOT 0

#define ADC_THRESHOLD 0x20

module Fadet_AODVM {
```

```
   provides {
      interface StdControl as Control;
      //      interface Settings;
      interface Intercept;
   }
   uses {
      interface Timer;
      interface StdControl as MHopControl;
      interface Send;
      interface Intercept as MultiHopIntercept;
      interface StdControl as UARTControl;
      interface BareSendMsg as UARTSend;
      interface Leds;
      interface Piggyback;
      interface MultiHopMsg;
      interface SingleHopMsg;
      interface SendMHopMsg as SendMHopMsgPing;
      interface Receive as ReceivePing;
      interface SendMHopMsg as SendMHopMsgAck;
      interface Receive as ReceiveAck;
      interface SendMHopMsg as SendMHopMsgList;
      //interface Receive as ReceiveList;
      interface Random;
      interface ADC;
      interface StdControl as ADCControl;
   }
}

implementation {
   TOS_Msg msg_buf;
   TOS_MsgPtr msg;
   bool send_pending;
   TOS_Msg delivery_buf;
   TOS_MsgPtr delivery_msg;
   bool delivery_pending;
   uint16_t xmitRate;   // in sec
   uint16_t timeCounter;
   uint16_t pingcounter;
   TOS_Msg ack_buf;
   TOS_MsgPtr ack;
   uint16_t ackNumTries;
   uint16_t pingNumTries;

   bool isMember;
   bool isFailed;
   uint8_t ackTaskPending;
   uint8_t sendTaskPending;

   uint8_t ackTarget;
   uint8_t targetID;

   uint8_t temptgt;
   uint16_t rounds;

   uint8_t memberList[FADET_MAX_MEM];
   uint8_t pingList[FADET_MAX_MEM];
   uint8_t failList[FADET_MAX_MEM];
```

```
   typedef struct {
      uint8_t sourceMoteID;
      uint8_t action;
      uint8_t hops;
   } Fadet_Msg;

   typedef Fadet_Msg *Fadet_MsgPtr;


   enum {
      FADET_PING = 2,
      FADET_ACK = 3
   };

   command result_t Control.init() {
      int i;

      msg = &msg_buf;
      ack = &ack_buf;
      send_pending = FALSE;
      delivery_msg = &delivery_buf;
      delivery_pending = FALSE;
      xmitRate = TR_SEND_RATE;
      timeCounter=0;
      pingcounter=0;
      ackTarget = 0;
      ackNumTries = 0;
      targetID = 0;
      pingNumTries = 0;

      rounds = 0;
      temptgt =0;

      ackTaskPending = TASK_DONE;
      sendTaskPending = TASK_DONE;
      isMember = FALSE;
      isFailed = FALSE;
      if (TOS_LOCAL_ADDRESS < FADET_MAX_MEM)
            isMember = TRUE;

#if PLATFORM_PC
      if ((isMember) && (TOS_LOCAL_ADDRESS >= (FADET_MAX_MEM -
FADET_FAIL)))
            isFailed = TRUE;
#endif

      for (i=0;i < FADET_MAX_MEM; i++) {
            memberList[i] = i;
            pingList[i] = 0;
            failList[i] = FALSE;
      }


#if SINK_NODE
      call UARTControl.init();
#endif
```

```
      call Random.init();
      call Leds.init();
      call ADCControl.init();
      return call MHopControl.init();
   }

   command result_t Control.start() {
      call MHopControl.start();
      call ADCControl.start();
      call Timer.start(TIMER_REPEAT, CLOCK_SCALE);
#if SINK_NODE
      call UARTControl.start();
#endif
      return SUCCESS;
   }

   command result_t Control.stop() {
      call Timer.stop();
      call ADCControl.stop();
#if SINK_NODE
      call UARTControl.stop();
#endif
      return call MHopControl.stop();
   }

   default command result_t Piggyback.receivePiggyback(wsnAddr addr,
                                                       uint8_t
*buf,
      uint8_t len) {
      return SUCCESS;
   }

   default command result_t Piggyback.fillPiggyback(wsnAddr addr,
                                                    uint8_t *buf,
                                                    uint8_t len)
{
      return SUCCESS;
   }

   default event PacketResult_t Intercept.intercept(TOS_MsgPtr m, void
*payload,
                                                    uint16_t len) {
      return SUCCESS;
   }

   // to be called by the Intercept - Replaced before calling
Intercept.intercept
//   command uint8_t PluginPayloadlinkPayload(TOS_MsgPtr m, uint8_t **
buf) {
//      TraceRoute_MsgPtr tr;
//      uint8_t len = call MultiHopPayload.linkPayload(m, (uint8_t **)
&tr);
//      *buf = &(tr->trace[len - TR_PIGGYBACK_LEN - TR_PLUGIN_LEN
//                                    - offsetof(TraceRoute_Msg,
trace)]);
//      return TR_PLUGIN_LEN;
//   }
```

```
   task void sendMessage() {
      Fadet_MsgPtr tr;
      uint16_t len;
      uint8_t payload_len;
      int rndTries;
      uint8_t targetIndex;


      tr = call Send.getBuffer(msg, &len);
      payload_len = (uint8_t)len;

      tr->sourceMoteID = (wsnAddr)TOS_LOCAL_ADDRESS;
      tr->action = FADET_PING;
      tr->hops = 0;

      rndTries = 0;
      targetIndex = (call Random.rand() & 0xff) % FADET_MAX_MEM;
      while ((failList[targetIndex] == TRUE) || (pingList[targetIndex]
!= 0) || (memberList[targetIndex]==TOS_LOCAL_ADDRESS))
      {
            rndTries++;
            if (rndTries >= FADET_MAX_MEM)
            {
                  sendTaskPending = TASK_DONE;
                  return;
            }
            targetIndex = (call Random.rand() & 0xff) % FADET_MAX_MEM;
      }

      targetID = memberList[targetIndex];


      payload_len = 3;

      dbg(DBG_USR1, "Fadet: sending ping to %d \n",targetID);
      if (!send_pending && call
SendMHopMsgPing.sendTTL((uint16_t)targetID,payload_len,msg,(uint8_t)32)
) {
        dbg(DBG_USR1, "Traceroute: Send for new packet\n");
        send_pending = TRUE;
        sendTaskPending = TASK_DONE;
        pingList[targetIndex] = pingList[targetIndex] + 1;
      }
      else {
        pingNumTries = PING_NUM_TRIES;
        sendTaskPending = TASK_REPOSTREQ;
        return;
      }
   }


      task void sendAck() {
         Fadet_MsgPtr tr;
         uint16_t len;
         uint8_t payload_len;
```

```
         tr = call Send.getBuffer(ack, &len);
         payload_len = (uint8_t)len;

         tr->sourceMoteID = (wsnAddr)TOS_LOCAL_ADDRESS;
         tr->action = FADET_ACK;
         tr->hops = 0;

         //targetID = ackTarget;

      payload_len = 3;

      dbg(DBG_USR1, "Fadet: sending ackn to %d\n",targetID);
      if (!send_pending && call
SendMHopMsgAck.sendTTL((uint16_t)ackTarget,payload_len,ack,(uint8_t)32)
) {
        send_pending = TRUE;
        ackTaskPending = TASK_DONE;
        dbg(DBG_USR1, " sendAck sent from %d\n",targetID);
      }
      else {
        ackNumTries = ACK_NUM_TRIES;
        ackTaskPending = TASK_REPOSTREQ;
        dbg(DBG_USR1, " sendAck failed from %d\n",targetID);
        return;
      }
   }

       task void resendAck(){
            uint8_t payload_len = 3;
       dbg(DBG_USR1, " in resendAck from %d\n",ackTarget);
       if(ackNumTries <= 0){
      ackTaskPending = TASK_DONE;
      return;
         }
         if (!send_pending && call
SendMHopMsgAck.sendTTL((uint16_t)ackTarget,payload_len,ack,(uint8_t)32)
) {
      send_pending = TRUE;
      ackTaskPending = TASK_DONE;
      dbg(DBG_USR1, " resendAck sent from %d\n",ackTarget);
      }
         else{
      ackNumTries--;
      ackTaskPending = TASK_REPOSTREQ;
      dbg(DBG_USR1, " resendAck failed from %d\n",ackTarget);
         }
    }

      task void resendPing(){
            uint8_t payload_len = 3;
            int i;
       dbg(DBG_USR1, " in resendPing from %d\n",targetID);
       if(pingNumTries <= 0){
      sendTaskPending = TASK_DONE;
      return;
         }
```

```
         if (!send_pending && call
SendMHopMsgPing.sendTTL((uint16_t)targetID,payload_len,msg,(uint8_t)32)
) {
      send_pending = TRUE;
      sendTaskPending = TASK_DONE;

      for (i=0;i<FADET_MAX_MEM;i++)
      {     if (memberList[i]==targetID)
                  break;
      }
      pingList[i]++;

      dbg(DBG_USR1, " resendPing sent from %d\n",targetID);
      }
         else{
      pingNumTries--;
      sendTaskPending = TASK_REPOSTREQ;
      dbg(DBG_USR1, " resendPing failed from %d\n",targetID);
         }
    }


   event PacketResult_t MultiHopIntercept.intercept(TOS_MsgPtr
forward_msg,
                                    void *payload, uint16_t len) {
      Fadet_MsgPtr tr = (Fadet_MsgPtr)payload;


      tr->hops++;
      ///return signal Intercept.intercept(forward_msg, data,
(uint16_t)TR_PLUGIN_LEN);
      return SUCCESS;
   }

#if PLATFORM_PC
   void print_Fadet(TOS_MsgPtr m) {
      uint8_t i;
      uint8_t len;
      uint16_t payload_len;
      Fadet_MsgPtr tr;
      char buf[160];
      char *b = buf;
      tr = call Send.getBuffer(m, &payload_len);
      len = (uint8_t)payload_len;

         sprintf(b, " %02d", tr->sourceMoteID);
         b+=3;
         sprintf(b, " %02d", tr->action);
         b+=3;
         sprintf(b, " %02d", tr->hops);
         b+=3;

      dbg(DBG_USR1, "Delivering packet from node %d: %s\n",call
MultiHopMsg.getSource(m), buf);
      {
            dbg(DBG_USR1,"Fadet2 : got packet from %d\n ",tr-
>sourceMoteID);
```

```
      }
   }
#endif

   task void deliverMessage() {
      // deliver locally
      if (call UARTSend.send(delivery_msg) != SUCCESS) {
         dbg(DBG_USR1, "Delivery on UART failed: \n");
         delivery_pending = FALSE;
      }
#if PLATFORM_PC
      print_Fadet(delivery_msg);
#endif
   }

   event result_t UARTSend.sendDone(TOS_MsgPtr sentMsg, result_t
success) {
      if (sentMsg == delivery_msg) {
         delivery_pending = FALSE;
         return SUCCESS;
      }
      return FAIL;
   }


   result_t display()
     {
      if (isMember == TRUE)
      {
        call Leds.yellowOn();
        call Leds.greenOn();
        call Leds.redOn();
      }
        return SUCCESS;
  }


  void checkLists()
  {
      int i,countf;

      for (i=0;i<FADET_MAX_MEM;i++)
      {
         if (pingList[i]!=0)
            pingList[i]++;


         if (pingList[i] > xmitRate)
          {
            failList[i]=TRUE;
            pingList[i] =0;
            dbg(DBG_USR1,"Mote %d has died\n",memberList[i]);
          }
       }

      countf=0;
      for (i=0;i<FADET_MAX_MEM;i++)
```

```
      {
        if (failList[i] == TRUE)
            countf++;
      }

      if (countf >= FADET_FAIL)
      {
            dbg(DBG_USR1,"Found all failures in %d rounds \n",rounds);
            display();
      }
  }

  void alive(uint8_t mote)
  {

    int i;

     for (i=0;i<FADET_MAX_MEM;i++)
     {
      if (memberList[i]==mote)
            break;
     }

     pingList[i]=0;
     failList[i]=FALSE;

  }


   event result_t Timer.fired() {
      timeCounter++;
      pingcounter++;



#if PLATFORM_PC
      if ((TOS_LOCAL_ADDRESS == FADET_ROOT) && (isMember) &&
(!isFailed))
            checkLists();
#else
      if ((isMember) && (!isFailed))
            checkLists();
#endif

      if ((pingcounter<(xmitRate / PING_NUM_TRIES)) && (timeCounter <
xmitRate)) {
                  return SUCCESS;
            }
      pingcounter=0;

      if (isMember)
            call ADC.getData();

      if(ackTaskPending == TASK_REPOSTREQ){
                  ackTaskPending = TASK_PENDING;
                  post resendAck();
            }
```

```
      if(sendTaskPending == TASK_REPOSTREQ){
                  sendTaskPending = TASK_PENDING;
                  post resendPing();
            }


      if (timeCounter < xmitRate) {
         return SUCCESS;
      }

      rounds++;
      dbg(DBG_USR1,"Round %d started\n",rounds);
      timeCounter = 0;

      if (sendTaskPending == TASK_DONE) {
#if PLATFORM_PC
      if ((TOS_LOCAL_ADDRESS == FADET_ROOT) && (isMember) &&
(!isFailed)) {
#else
      if ((isMember) && (!isFailed)) {
#endif
           sendTaskPending = TASK_PENDING;
             post sendMessage();
         }
     }


      return SUCCESS;
   }

   event TOS_MsgPtr ReceivePing.receive(TOS_MsgPtr received_msg, void
*payload, uint16_t len) {
      TOS_MsgPtr ret = received_msg;
      Fadet_MsgPtr tr = (Fadet_MsgPtr)payload;
      void *data;

      dbg(DBG_USR1," Fadet : got ping from %d\n",tr->sourceMoteID);

      if (isFailed)
            return ret;

      if (ackTaskPending == TASK_DONE)
      {
            ackTarget = tr->sourceMoteID;
            ackTaskPending = TASK_PENDING;
            post sendAck();
      }

      return ret;
   }

   event TOS_MsgPtr ReceiveAck.receive(TOS_MsgPtr received_msg, void
*payload, uint16_t len) {
      TOS_MsgPtr ret = received_msg;
```

```
      Fadet_MsgPtr tr = (Fadet_MsgPtr)payload;


      dbg(DBG_USR1, "Fadet : %d is alive\n", tr->sourceMoteID);

      alive(tr->sourceMoteID);

      return ret;
   }



   event result_t Send.sendDone(TOS_MsgPtr sentMsg, result_t success) {
      dbg(DBG_USR1, "Traceroute: sendDone (%s)\n",
                    (success == SUCCESS ? "SUCCESS": "FAIL"));
      if ((send_pending == TRUE) && (sentMsg == msg))
      {
#if USE_SYNC_ACK
         if ((success == SUCCESS) && sentMsg->ack) {
#else
         if (success == SUCCESS) {
#endif
            timeCounter = 0;
         }
         send_pending = FALSE;
         return SUCCESS;
      }


      if ((send_pending == TRUE) && (sentMsg == ack)) {
                send_pending = FALSE;
          return SUCCESS;
      }

      return FAIL;
   }


   async event result_t ADC.dataReady(uint16_t data) {
       if (data < ADC_THRESHOLD)
                isFailed = TRUE;

       if (data >= ADC_THRESHOLD)
                call Leds.yellowOn();
       else
                call Leds.yellowOff();

       return SUCCESS;
     }




#if 0
   command result_t Settings.updateSetting(uint8_t *buf, uint8_t *len)
{
      xmitRate = *buf;
```

```
      *len = 1;
      return SUCCESS;
   }

   command result_t Settings.fillSetting(uint8_t *buf, uint8_t *len) {
      *buf = xmitRate;
      *len = 1;
      return SUCCESS;
      }
#endif

   }
```