Networks and Distributed Systems

Intelligent Agents in Ad Hoc Networks

by

Anthony Winters, B.A.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Anthony Winters

September 10, 2008

Permission to Lend and/or Copy

Acknowledgments

First and foremost I would like to thank my supervisor Dr. Stefan Weber for his patience and guidance throughout the course of my dissertation.

Secondly I would like to thank our course supervisor Dr. Siobhán Clarke for giving me the oppourtunity to take on the NDS challenge.

Thirdly I would like to thank my family, Angie, me Da, Kitty and J for their support throughout the year.

Next I would like to thank the entire NDS class, especially the lads in the lab, Shane, Eric and Cian, it's nice to know insanity is contagious.

Finally I would like to thank Bits, I couldn't have done it without her.

Anthony Winters

University of Dublin, Trinity College
September 2008

“Whether you think that you can, or that you can’t, you are usually right”

- Henry Ford

Networks and Distributed Systems

Intelligent Agents in Ad Hoc Networks

Anthony Winters, M.Sc.
University of Dublin, Trinity College, 2008

Supervisor: Dr. Stefan Weber

Ad hoc networks provide a rapidly deployable, "bring your own network" solution for situations where the use of infrastructure networks is not possible. Groups who could benefit from such a network solution include "search and rescue" teams and mobile military operations. One key limitation however, in the use of ad hoc networks is the efficiency in which data may be successfully routed through them. Routing protocols such as AODV [1] and GPSR [2] have provided usable solutions, however such protocols' performance is greatly reduced when faced with erroneous scenarios such as node failure and lossy wireless links. Multi Agent Systems (MAS) is a field of computer science that is concerned with producing environment aware software agents that use intelligent algorithms to allow them to sense and reason about their environment, and take decisive action. Using the MAS paradigm, an efficient solution to successful routing of data in ad hoc networks was devised. The proposed solution, entitled the Agent Distribution (AD) mechanism is an opportunistic, on-demand mechanism for distributing software agents (and by extension the data which they carry) throughout an ad hoc network. The AD mechanism was implemented in a message delivery system entitled Shoulder Monkey. This system was then performance tested and compared to a simple TCP routing protocol. Results showed that the Shoulder Monkey system out performed the simple TCP routing protocol on a number of network topologies. This indicated that the AD mechanism could be used as a viable solution for efficient routing of data in ad hoc networks.

Contents

List of Figures

Chapter 1

Introduction

Ad hoc networks and Multi Agent Systems (MAS) are two highly active research areas in the field of computer science. Ad hoc networks provide an alternative to wired infrastructure networks and managed wireless networks, whereas MAS allow systems to be designed and implemented with system optimisation at the forefront of the development process.
This chapter discusses the motivation for this work and defines the research questions the author hopes to answer. This chapter then gives a brief introduction into the two main research areas studied in this work, ad hoc networks and multi agent systems. The chapter is then concluded with a road-map for the remainder of this dissertation.

1.1 Motivation

The use of ad hoc networks in real world situations is less than optimal as said networks are hampered with problems, some of which are discussed in chapter 2 (section 2.1 Ad Hoc Networks). By combining the paradigms of ad hoc networking and Multi Agent Systems (MAS), ad hoc networks could reach their full potential. This dissertation is therefore concerned with answering the following research questions:

1. Can ad hoc networks be used in real-world real-time situations?
2. Is ad hoc network performance enhanced through the use of the MAS paradigm?
3. Can the MAS paradigm be used to route data in ad hoc networks?

1.1.1 Ad Hoc Networks

"Ad Hoc: For the purpose of" - Oxford Dictionary [4]

An ad hoc network is a wireless network set up for a certain purpose. Ad hoc networks have no set infrastructure so they can be rapidly deployed. Participant devices in the ad hoc network are called nodes. Nodes act as transmitters, receivers and routers for data within the network. Therefore data can be routed in a multi-hop fashion through the network; much like the way a frog might traverse a pond using multiple lily-pads i.e. nodes, and hops i.e. transmissions as seen in Fig. 1.1.
There are two types of ad hoc network, namely:

1. Static Ad Hoc Networks
2. Mobile Ad Hoc Networks (MANETs)

Nodes in a static ad hoc network do not move (hence the keyword static) once they are deployed. Examples of static ad hoc networks include urban roof top networks. Nodes in a mobile ad hoc network (MANET) can move around (hence the keyword mobile) once they are deployed. Examples of MANETs include networks comprised of personal mobile computing devices e.g. PDAs.

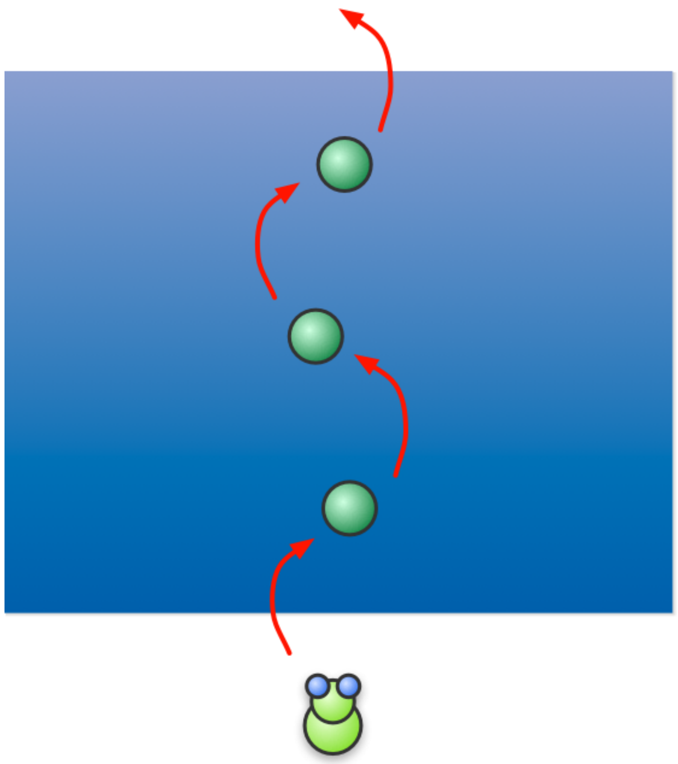


Figure 1.1: Metaphorical representation of an ad hoc network

1.1.2 Multi Agent Systems

"an agent is an artificial, computational entity that can perform certain tasks with a certain degree of autonomy or initiative whilst intelligently adapting to its environment" - Woodbridge [5]

The paradigm of Multi Agent Systems (MAS) is by no means a new concept to the world of computer science; however in recent years due to the growth of distributed computing practices for complex tasks it has seen somewhat a revival. Agents can ultimately be seen as helpers, which enhance a systems performance. The fundamental idea behind the MAS is to have individual agents that can display "intelligent" qualities by being:

- Environmentally aware - Agents should be able to sense the environment in which they operate and dynamically react to situations that may arise
- Interactive - Software agents should be able to interact with other software agents in order to acquire information and collaboratively complete assigned tasks.
- Autonomous - Software agents should be able to make self-dependant decisions and take decisive actions.

Agents can be implemented as software entities that operate on a heterogeneous system consisting of numerous devices.

1.2 Road-map

The remainder of this dissertation is broken up as follows.
Chapter 2 discusses the state of the art and background information relating to ad hoc networks, multi-agent systems (MAS), the use of MAS in ad hoc networks and technologies used in the areas of MAS and ad hoc networking.
Each of the subsections explains the current state of the art and background research in the given field, or combination of fields.
Chapter 3 discusses the design of the Agent Distribution(AD) mechanism. This chapter explains the need for such a mechanism in the given domain and gives a detailed account of the operational semantics of the mechanism.
Chapter 4 discusses the implementation of the AD mechanism into a personal messenger system called Shoulder Monkey. This chapter discusses the design and implementation of the system in detail and gives reasons behind the need for such a system.
Chapter 5 discusses the evaluation of the implemented Shoulder Monkey system and, by extension, the evaluation of the AD mechanism. This chapter compares the performance

of the Shoulder Monkey system to communicating using TCP. The evaluation of the system was carried out using a number of topologies on a purpose build testbed, namely a real world ad hoc network. A complete analysis of results obtained from both testbeds is then discussed.
Chapter 6 discusses the main conclusions obtained from work carried out during the course of this dissertation.
Chapter 7 concludes the dissertation with future work that may be carried out building on the work completed for this dissertation.

1.3 Summary

In this chapter, we discussed the author's motivation for completing this work, and set out the research questions that the author wishes to answer during the course of this study. A brief introduction has been given into the key research areas of this work. Finally a road-map to this work has been defined to inform readers of the layout of this dissertation.

Chapter 2

State of the Art

The two research areas this work is concerned with are ad hoc networks and MAS. As both fields are highly active research topics, a lot of information is available in said research areas. An understanding of each research area as an individual discipline is required before research into a study which combines the areas of ad hoc networks and MAS is undertaken.
The remainder of this chapter discusses background research and the state of the art for ad hoc networks and MAS. Ad hoc network limitations, routing protocols and security issues are first discussed. This discussion is followed by an evaluation of MAS architectures, communication, collaboration and standardisation. These two sections are followed by a review of research which combines ad hoc networks and MAS. Finally a brief overview of technologies currently being used in ad hoc networks and MAS is given.

2.1 Ad Hoc Networks

Potential uses for ad hoc networks include providing rapidly deployable networks for organisations such as search and rescue teams and mobile military operations alike. However making potential uses of ad hoc networks a reality seems to be ultimately limited by the efficiency in the performance of such networks. In situations where data delivery in a network can literally mean life or death, effective real time data routing protocols need to be in operation. Therefore current uses of ad hoc networks are somewhat limited as performance is not always efficient or reliable.
The use of multi agent systems could provide a way to manage ad hoc networks without the use of a central control point, and increase efficiency, reliability and performance. This could ultimately lead to a more stable network environment which would lead to greater success in the routing of data through the network. By combining the two technologies

of ad hoc networks and MAS, potential uses for ad hoc networks could become a reality. Much research is being carried out into the use of ad hoc networks for search and rescue personnel and mobile military operations. Ultimately the practical use of ad hoc networks is limited only by the success and efficiency in which data can be routed through the network. As such, the topic of routing protocols in ad hoc networks is an active research area.

2.1.1 Routing Protocols in Ad Hoc Networks

In an ad hoc network there is the basic case of routing data from a source node S to a destination node D, through intermediate nodes I...I+n. This section looks at two approaches to routing in ad hoc networks, namely the topology-based routing approach and the location-based routing approach.

Topology-Based Routing

Topology-based routing uses information about links between nodes in an ad hoc network to forward data packets. There are three main types of topological approaches to routing data [6]:

1. Proactive Routing - Whereby nodes in the ad hoc network continually make their presence known to other nodes in the network to ensure data can be routed to them.
2. Reactive/On-Demand Routing - Whereby nodes only acquire information relating to network routing when data packets are injected into the network.
3. Hybrid Routing - Which encompasses one or more of the characteristics of both proactive and reactive/on-demand routing approaches.

Flooding is the simplest form of topoloical data routing in an ad hoc network [7]. The idea behind this mechanism is to propagate the data packet to all the nodes within range of the source node S. Then each node that received this data packet will forward the data packet to all nodes within range of it, and so on until the destination node D is reached. Theoretically if all nodes in the network forward the same data packet to multiple nodes, multiple but identical data packets will ultimately find different routes to the destination node D, therefore increasing the chances of successful data packet delivery.
There are, however, two fundamental problems with this approach to routing data packets in an ad hoc network:

1. Excessive Bandwidth Utilisation

2. Broadcast Storm Problem

The first problem encountered is excessive bandwidth utilisation. As each node that receives the data packet will forward the data packet, excessive use of the transmission medium results. Therefore, in theory, a single data packet routed in this fashion could saturate the entire network, as it will be retransmitted by every node that receives it. The second problem encountered by flooding, again, stems from the retransmission of data packets by each node that received the data packet. This causes a problem known as a broadcast storm [7], whereby every node, upon receiving the data packet, will try to re-transmit the data packet to all nodes within its transmission range. As many of these nodes will be within transmission range of each other, this will ultimately cause multiple data packet collisions. Nodes will then need to retry their re-transmit, again at roughly the same times, causing further data packet collisions.
A number of approaches to alleviate the broadcast storm problem exist such as; staggering initial re-transmission time of receiving nodes [7], thus decreasing collision possibilities, and having only certain receiving nodes re-transmit the data packet [7].
Due to the limitations of flooding as a routing protocol in ad hoc networks different routing protocols have been devised.
Dynamic Source Routing (DSR) [8] is a reactive routing protocol used in ad hoc networks. DSR has two parts to its operation; the first part being route discovery and the second part being route maintenance. In DSR only once a route from a source node S to a destination node D through intermediate nodes I...I+n has been discovered, will a data packet be routed from S to D. DSR routes data packets by appending routes to the actual data packet header thus allowing each intermediate node to ascertain the next hop the data packet should be routed along. This approach is less than optimal as long routes will ultimately lead to longer headers thus increasing the actual data packet size that must be transmitted. DSR also makes use of caches to store routes to nodes that are in use in the network. During the route discovery phase intermediate nodes can return routes to sought destinations directly to querying nodes. This saves route discovery time, however it also introduces the possibility of intermediate nodes returning out of date information about routes to querying nodes.
Ad Hoc On-Demand Distance Vector Routing (AODV) [1] is another routing protocol that is similar to DSR. It is also a reactive routing protocol and only maintains routes to destinations that are in use. However the main difference between DSR and AODV is that AODV does not encapsulate the route in a data packet but makes use of routing tables stored at each node in an ad hoc network to route data to specific locations. Intermediate nodes can also respond to route discovery messages as in DSR, however the introduction of sequence numbers somewhat alleviates the of problem out of date information being

returned to querying nodes. Due to routing tables AODV does not scale very well, as even though it is a reactive protocol, if a large network has many active nodes routing tables can quite quickly become very large.
Ad Hoc On-Demand Distance Vector Routing with Backward Routing (AODV-BR) [9] is an adaptation to the AODV routing protocol. AODV does not use multiple paths to destinations, so if a node fails the route discovery phase of AODV must be re-initiated to find a new route to a destination node D. This can lead to a high overhead in the network. AODV-BR makes use of backward paths when a failed node is encountered. The protocol allows data packets to "hop back" two nodes in order to find alternate routes to destinations. If, however, a route cannot be found from the use of a two hop backward hop then a route discovery must be re-initiated. When compared to normal AODV, AODV-BR performed better than regular AODV, especially when node mobility was introduced into the ad hoc network, i.e. when a MANET was created. However in static ad hoc networks it was seen as over kill as an AODV route discovery would cause less network overhead than maintaining the backward routes AODV-BR requires.

Position-Based Routing

Position based routing protocols try to address some of the problems that exist in routing data in ad hoc networks using topology based routing protocols [6]. For position-based routing protocols the physical geographic location of nodes must be known, therefore it is assumed that each node is equipped with a location device e.g. a GPS locator, or that an external service will provide nodes with the physical geographic location of all nodes within the ad hoc network.
Within position-based routing protocols there is the concept of greedy forwarding [7]. Imagine the following scenario, source node S is trying to route a data packet to destination node D via intermediate nodes I...I+n. If greedy forwarding is used, S will route the data packet to the node within its transmission range which is geographically closest to D. If this procedure is followed by all nodes within the network, the amount of hops that must be taken for the data packet to reach destination node D will be minimised. On the surface this is a very efficient approach to routing data in ad hoc networks; however it has one major flaw, local maximums or dead ends. If source node S routes a data packet to an intermediate node I that is geographically closest to destination node D, there is no guarantee that intermediate node I has a route to destination node D. Therefore once data has been routed to intermediate node I it hits a local maximum or dead end and can go no further.
Ad Hoc On-Demand Distance Vector Routing with Directional Forward Routing (AODV-

DFR) [10] is a protocol that makes use of greedy forwarding to route data packets. When AODV- DFR encounters a local maximum it uses backup paths that are set up using a beacon system to find an alternate route. This beacon system uses fisheye scaling [11]; whereby beacons closer to the destination node D are transmitted more frequently, therefore as a data packet gets closer to the destination node D information on how to get there, i.e. backup paths, gets better. In simulations AODV-DFR outperforms regular AODV, however due to the use of a beaconing system, it does introduce a greater overhead into the network.
The Greedy Perimeter Stateless Routing (GPSR) [2] protocol is another protocol that uses greedy forwarding. When GPSR hits a local maximum it uses the concept of perimeter forwarding to recover. Perimeter forwarding uses the right hand rule to traverse around the local maximum thus allowing the data packet to be delivered.
More environment-specific routing protocols have also been researched. The Greedy Perimeter Coordinator Routing (GPCR) [12] protocol is used to route data packets in urban environments where concrete buildings can act as obstacles for radio transmissions. GPCR does not use global or external information (i.e. maps) about the environment; instead the protocol uses restricted greedy forwarding to ensure data packets are not routed past street junctions in urban environments. GPCR does this by having nodes act as coordinators at junctions. Data packets are always routed to a coordinator if a coordinator exists, before any other node in the network. Therefore in between junctions, greedy forwarding is used for routing until a coordinator node is reached.
It is evident that position-based routing has its advantages as an approach to routing data in ad hoc networks. However it does require the additional incorporation of location information for nodes (e.g. GPS coordinates), the additional fitting of nodes with location devices (e.g. GPS locators) or an external service that provides such information, and the added overhead that is needed to reason about routing data packets through the network.

2.1.2 Security Considerations for Ad Hoc Networks

Nodes in ad hoc networks were originally intended to operate in a friendly environment. However this may not always be the case and nodes may need to be capable of operating in both hostile and friendly enviroments. As each node in an ad hoc network can act as a data router for other nodes the fundamental question is raised; how does one guarantee data transmission from a security point of view, in an ad hoc environment?
When considering security issues in ad hoc networks two types of attack must be taken into account namely:

1. External attacks - attacks that originate from outside the ad hoc network.

2. Internal attacks - attacks that originate from a compromised node within the ad hoc network.

Denial of Service (DoS) Attack

DoS attacks can originate from either external or internal attackers. External attacks could introduce interference, thus impeding ad hoc networks from operating properly. Internal attacks could create a DoS attack via packet blasting or the injection of considerable amounts of junk packets into the network, thus creating congestion in the network and degrading its performance. An internal attack could even consist of a malicious node continually transmitting control packets, thus starving other nodes of network utilisation. If two or more network nodes were compromised a “wormhole” [13] could be formed, which would enable the flow of data to propogate between compromised nodes. Hu et al. [13] discussed the use of packet leashes as a possible defense against such attacks.

Packet Authentication

An issue raised by the possibility of network nodes becoming compromised is that of data authenticity and integrity. In the Dynamic Source Routing (DSR) protocol [8] for example, a change to the RREQ or RREP control packets could hide certain routes or route data along an intended or non-existent route. This situation could easily be avoided if data authenticity and integrity were introduced, however this is not a easy task to accomplish in ad hoc networks. The problem of packet authentication could be solved using cryptographic keys, however this raises the questions of who issues said keys, and where are they stored. Ramkumar et al. [14] proposed a scheme for predistribution of keys within an ad hoc network which could be used to solve this problem.

Possible Solutions

Hao et al. [15] identified that security solutions in ad hoc networks should provide complete protection by spanning the complete protocol stack. The also concluded that two kinds of approaches to security in ad hoc networks should be taken; proactive and reactive, and that each of these approachs should encompass prevention of attacks, detection of attacks and reaction to attacks.
Reasearch carried out by Manikopoulos et al. [16] has produced an architecture for security in ad hoc networks. When tested this architecture was shown to withstand attacks of both an internal and external nature. It is therefore evident from the works mentioned that effective security in an ad hoc networked environment is a non-trivial problem and

requires much consideration. Ad hoc-specific security solutions need to be further researched as current solutions for wired computer networks may not be transferrable to wireless networks and those that are may not be transferrable to an ad hoc environment.

2.2 Multi Agent Systems (MAS)

The computer science paradigm of Multi Agent Systems (MAS) is by no means a new concept in the field of computer science. MAS has a number of key idioms which are needed to fully understand the field.

2.2.1 Belief Desire Intention (BDI) Architecture

In MAS the BDI architecture [17] is used to help model the behaviour of an agent. The architecture is comprised of three parts:

- Belief - What an agent knows or does not know about its environment.
- Desire - What goal(s) the agent wants to achieve.
- Intention - How the agent plans to achieve its goal(s).

The BDI architecture is based on practical reasoning or reasoning towards actions. The following example explains how the BDI architecture may be used.
A villager is given the task of going to the well to fetch some water. The villager wants to achieve the goal of bringing water back from the well, so in the BDI architecture his desire is to bring water back from the well. How the villager might achieve the goal of bringing water back from the well is as follows:

a) Walk to the well.

b) Get the water out of the the well.

c) Carry the water back.

These three actions relate to the villager's intentions in the BDI architecture. They are the steps he must take in order to achieve his main goal of fetching water from the well. The villager's beliefs in the BDI architecture relate to him knowing about his environment, i.e. being environmentally aware. To complete the goal of fetching water from the well the villager must have the following beliefs:

a) The villager must know how to walk.

b) The villager must know where the well is.

c) The villager must know how to get water out of the well.

d) The villager must know how to carry water back from the well.

Note that this knowledge is referred to as a belief as it can be wrong, for example the villager might think he knows where the well is but may be mistaken. Suppose the villager did not know where the well was located, then he would have the belief that:

a) He did not know where the well was.

In order to complete his goal of fetching water from the well, the villager must acquire this information. To do this the villager would have to ask a fellow villager where the well was, and interact with his environment, before he could proceed.
The above example shows how the BDI architecture can be used to model agent behaviour. Ronald et al. [18] used an agent-based simulation to model pedestrian behaviour in an urban environment. Each pedestrian in the simulation was modelled using an autonomous agent. The agents used a BDI architecture to help model their behaviour. Agents were chosen to model pedestrians as they could easily encompass the behaviour of a pedestrian.

2.2.2 Agent Communication Languages

A number of communication languages such KQML [19] and FIPA ACL [20] are used to allow agents to communicate. These languages are based on speech-act theory, whereby the speaker makes a declarative statement, e.g. "I declare that...", and in doing so completes an action. Agent communication languages are essential to enable agents to interact with each other and to collaboratively complete tasks.
Figure 2.1 shows an example of a FIPA ACL message used to communicate the price of a bid in an auction between two agents. The first key word in the message is *inform*, this sets the declarative statement, or meaning of the message. The message is then made up of a number of keywords followed by values. For example the keyword *:sender* is followed by the value of the agent sending this message, in this case the sender being *Agent-A*. Other keywords are used to specify which agent to send the message to; in this example the *:receiver* is *Agent-B*, and what kind of reply the sender is expecting; in this case Agent-A is expecting Agent-B to *:reply-with* a bid value of the type *bid02*. Other keywords such as *:language* and *:ontology* allow the language and ontology of the message to be defined.

```
(inform
       :sender Agent-A
       :receiver Agent-B
       :reply-with bid02
       :content (price (bid good01) 100)
       :language fipa-sl
       :ontology auction
)
```

Figure 2.1: FIPA ACL example taken from Ahmad et al. [3]

2.2.3 Collaboration

Having agents work in collaboration is essential if agent technology is to reach its full potential. Agents need to be able to work together to complete complex tasks that are too labour-intensive for a single agent to complete.

Rosenschein et al. [21] proposed a framework for modelling communications between agents which allows agents to "keep promises" with each other. The framework makes the benevolent agent assumption, whereby all agents are seen to want to help one another. The proposed framework enables promises made by agents to become binding i.e. they have to be carried out. The framework introduces the concept of group offers; a group of tasks an agent is willing to collaborate on, and deals; all the agents' group offers combined. The framework also introduces a number of theorems which help agent collaboration to take place. The framework was used to solve the prisoners' dilemma problem, which showed that agents could successfully communicate and negotiate in order to complete a complex task.

Grosz et al. [22] presented a model that uses a group of agents to complete a complex plan without using the notion of jointly held intentions i.e. we-intentions. The model is based on the notion that, ultimately, all collaborative actions are based on the individual action of an agent. The model provides intention operators and potential intention operators in order to allow agents to collaborate on completing tasks.

Agent collaboration allows agents to complete complex tasks in a group-like work structure. However, there is still the case where agents may have many non-complex tasks to complete, where an abundance of such tasks could cause a single agent to become overloaded. Therefore, there is the need for a mechanism to allow agents to pass non-complex tasks off to other agents who are less resource constrained than they are. Shehory et al. [23] discussed such an approach whereby overloaded agents may pass tasks off to other agents. When no appropriate agents exist, an agent may clone itself to create the necessary resources for task completion. To enable this cloning approach an agent must be able to

reason about its current state i.e. load and available resources. In this study simulations were run to compare systems in which cloning of agents was allowed and where cloning of agents was not allowed. Results showed that for small numbers of tasks, cloning enabled systems and non-cloning enabled systems performances were almost equal. However when the number of tasks increased the cloning enabled system performed better than the non-cloning enabled system. Beyond a certain threshold of tasks to be completed however, even cloning-enabled systems showed poor performance. Therefore it is evident that the decision on whether or when to clone is no simple task, and system-specific algorithms to enable such behaviour seem to be the standard approach.
The opposite behaviour of merging two agents or self-extinction is also an important control mechanism for agent proliferation and must be studied if agent cloning is to be used in MAS.

2.2.4 Standardisation

Standardisation of agent technology took a long time to come about as many agent technologies were easier to design and implement if they were system specific. Standardisation of agent technology allows interoperability between agents from different platforms which is essential for successful growth of agent technology.
In 1997 the Foundation for Intelligent Physical Agents (FIPA) released a set of specifications [24] relating to agent technology standards. This paper was reviewed at the AMAAS 98 conference [25], which lead to a further FIPA specification being released at the end of 1998 [26]. The FIPA specification was finalised and released in 2000 [27]. This specification contains information needed to develop FIPA compliant agent based systems. These standardisation efforts by FIPA ultimately lead to agent interoperability becoming possible.

2.3 Combining Multi Agent Systems (MAS) and Ad Hoc Networks

Ad hoc networks may benefit from the introduction of the MAS paradigm into the field. Cicirello et al. [28] proposed that MAS can be used to help coordinate ad hoc networks. The Drextel University MANET [29] architecture is made up of a two tier architecture with agents being placed on the top tier and the MANET being placed on the bottom tier. A number of experiments were carried out using this MANET, including intruder detection in the network [28], an ant nest relocation problem [28] and an agent ecosystem [28] where agents had to complete tasks to stay "alive".

A lot of research in this area is being carried out using a MAS approach to distributed problem solving. Macker et al. [30] proposed a three tiered design approach for developed cross-layer MAS and MANET systems. This approach separates out the different sections of the system, with agents being on the top layer, middleware, e.g. routing protocols, being on the middle layer and the physical network, i.e. MANET, being on the bottom layer. This design divides out the MANET routing problem from the agent task problem. It also is suggested that using pro-active routing protocols to route data in a MANET may yield better results than using reactive routing protocols, as even though pro-active routing protocols introduce a higher overhead into the network it would lead to a) a lower average delay in transmissions between nodes and b) the overhead being predictable, especially during stressed conditions. Coordination of agents was tested using the predator prey problem with agents being required to work together in order to complete their assigned tasks. However as agents are supposed to be environmentally aware to aid them in carrying out their tasks this approach may seem counter-intuitive. A knowledge of network conditions at the agent level may better equip agents to carry out their tasks especially if the tasks involve traversing the network. Ultimately an agent based approach to routing may facilitate better efficiency in relation to task completion. Traditionally system optimisation techniques have relied heavily on prior knowledge of a system's environment and some form of centrally managed runtime knowledge. However with the use of agent technology some concepts from the field of machine learning can be used in a distributed fashion. Dowling et al. [31] proposed a collaborative reinforcement learning model (CRL) to achieve system optimisation. The CRL model was evaluated by using it to build a MANET routing protocol called SAMPLE. SAMPLE was compared against DSR and AODV for efficiency. DSR and AODV seemed to outperform SAMPLE when perfect radio links were simulated, however when more adverse network conditions were introduced SAMPLE outperformed both DSR and AODV.
Boukerche et al. [32] proposed the use of agents to secure routing in MANETs. Anonymous Routing Protocol Based on Mobile Agent (ARMA) encapsulates all routing information in an agent, which then migrates from node to node in the network. This approach removes any routing processing from the node and places it on the mobile agent. ARMA uses a uni-directional trust value relationship system to provide security against malicious nodes in the network. However for this protocol to function correctly an external certificate authority (CA) is required which may not be possible in an ad hoc environment especially where node mobility is introduced. The ARMA protocol also makes use of a Twice-ID Provided (TIP) Mechanism [32] whereby the nodes in a route discovery path and route reply path are compared for consistency. If any of the nodes are out of place it is presumed that the out of place node is potentially malicious. This approach however

does not allow different routes to be taken for route discovery and route replies, which may be a bit naive when considered for use in MANETs. Finally the ARMA protocol makes certain assumptions about what a malicious host will try to do, however such assumptions cannot be made in real world environments, as the intentions of a malicious host can never be fully known.
Misker et al. [33] put forth the concept that computer system users should not only interact with individual computer devices but with a computer system as a whole. Agent technology allows this concept to become a reality as agents can be used to enhance user experience by traversing networks to carry out complex tasks. When one considers that every individual in the modern world carries some sort of electronic device around with them and these devices could be used to make up a very extensive MANET the possibilities of agent technologies coupled with this MANETs are vast.
To allow the successful operation of ad hoc networks in the real world careful design considerations need to be taken into account. For example all agents will need to be interoperable with communication and collaboration between agents being at the forefront of the system design. Much work has been carried out in the individual areas of ad hoc networks and MAS. It seems the next logical step has been taken in combining these two areas, but more research on the potential benefits of this combined research area is needed.

2.4 Technology Overview

There are also a number of technologies which allow simulations to take place in ad hoc networks of both a static and mobile nature. These simulators allow complex scenarios to be run without the need for real world ad hoc networks to be deployed.
Agent technology can be used to enhance capabilities in complex computer systems. Before 1997 many of these agent technologies were system independent. With the introduction of the Foundation for Intelligent Physical Agents (FIPA) [34] a number of specifications were decided upon which enabled these technologies to become interoperable. Since this introduction of interoperable agents, a number of agent technologies which allow easy development of agents in agent based systems, have become available.

2.4.1 Ad Hoc Networks

Potential uses for ad hoc networks are currently being studied by a number of researchers. There are two approaches to testing scenarios in ad hoc networks, firstly via the use of simulators, and secondly via the use of real world deployed ad hoc networks.

Simulators

The most popular simulator used to simulate ad hoc network conditions is the ns-2 simulator [ts1]. The ns-2 simulator is used primarily to test routing protocols in ad hoc networks [2] [12], however some research has been carried out with the ns-2 simulator being used to test agent based systems [30].
Other possible choices for simulators include OPNET [ts2], OMNET++ [ts3], Glomosim [ts4], Qualnet [ts5]. OPNET provides a suite of commercial network simulation tools which could be used to model ad hoc network conditions. Qualnet is also a commercial product for ad hoc network simulation, however a free version of Qualnet called Glomosim is available to educational bodies. A knowledge of parsec and c is required to setup and run simulations using Glomosim. OMNET++ is an open source network simulator which is free for non-profit bodies and educational bodies alike.

Real World Deployment

Researchers at Drextel University, Philadelphia, USA, have not used simulations but have instead carried out real world experiments using the Philadelphia Area Urban Wireless Network Testbed (PA-UWNT) [28]. Research carried out by Cicirello et al. [28, 35], showed that complex agent systems can be built and tested using real world testbeds as opposed to simulations as is discussed in the previous sub-section.
For testing agent-based technologies there are a number of arguments for the use of a simulator such as; ease of testing, i.e. a fully operable MANET would not have to be deployed, and guaranteed results, i.e. even an incorrectly configured simulation would produce some results. However some simulators, such as ns-2, have a steep learning curve and require a lot of man-hours for a developer to become confident with its use. Also simulations are just that, simulations, and should only be used as a stepping stone for future real world testing.

2.4.2 Agent Frameworks

A number of agent technology frameworks are available for the development of agent-based systems, the vast majority of which are FIPA compliant.
Ahmad et al. [3] discussed the use of the Java Agent Development Envirmonment (JADE) [36] as a fundamental java-based framework for the develoment of agent technologies. JADE is fully FIPA compliant, which ensures ease of interoperability between agents on different platforms. Research carried out [37] [33], successfully used JADE to develop agents for their respective agent-based systems.

Ronald et al. [18] used JACK Intelligent Agents [38] to develop agents for their agent based system, as JACK was specifically designed for use in creating agent simulations. This enabled the system in question to be successfully designed in an efficient timeframe. O'Hare et al. [39] used the Agent Factory framework [40] to develop agents for agent-based systems they designed. Agent Factory is fully FIPA compliant and was used to successfully develop software agents used in the Gulliver's Genie [41] agent based system. Memebers of Drextel University, Philadelphia, USA, used the Extendable Mobile Agent Architecture (EMAA) framework [42] to develop agents used in agent-based systems in [28] and [35].
For the development of agents in an agent-based system a FIPA compliant framework such as Agent Factory or JADE would be an asset as it would remove FIPA compliancy issues from system development.

2.5 Summary

In this chapter, we discussed background research and the state of the art for the two research areas this work is concerned with; ad hoc networks and MAS. An overview of the limitations, routing protocols and security concerns for ad hoc networks has been discussed. Key concepts and research relating to MAS have been reviewed. Existing research which combines ad hoc networks and MAS has also been discussed. Finally an overview of the technologies used in developing systems which use ad hoc networks or MAS is given.

Chapter 3

Design

Ad hoc networks provide a “bring your own network” solution for situations where infrastructure networks cannot be used. Examples of uses range from search and rescue operations to mobile military operations. In ad hoc networks of both a static and mobile nature a key research question is how to provide successful and efficient routing of data through said networks. Many current ad hoc routing protocols such as AODV [1] and DSR [8] try to establish end-to-end routes between source and destination nodes before routing data packets. This approach places the decision of what route a data packet should take on the source node, thus not allowing dynamic adjustments to selected routes to be made during data transfer.
Other ad hoc routing protocols such as GPSR [2], AODV-DFR [10] and GSCR [12] use physical geographic locations to help route data packets efficiently. These protocols do not require an end-to-end route between source and destination to be known before the transmission of data packets begins. However, geographic routing protocols do require the physical location of the destination to be known i.e. via the use of GPS devices or external services that provide such information. Therefore the route taken by data packets in geographic routing protocols is decided by intermediate nodes along the route.
The research area of Multi Agent Systems (MAS) uses autonomous software agents to provide advanced functionality to software systems. Boukerche et al. [32] used software agents to provide routing anyonmity in ad hoc networks. This approach takes the responsibility off source and intermediate nodes of data packet routing and places it solely on the software agent.
This chapter discusses an optimistic on-demand/reactive routing mechanism for software agents entitled the Agent Distribution (AD) mechanism, which is used to distribute agents throughout an ad hoc network.

3.1 AD Mechanism

As mentioned above, the AD mechanism is an optimistic on-demand/reactive routing mechanism for software agents used in an ad hoc networked environment. The AD mechanism removes the responsibility of routing data packets from nodes in an ad hoc network and places it solely on network aware software agents that can migrate freely around the ad hoc network in question. The AD mechanism does not fit into any of the routing protocol approaches discussed in chapter 2 (section 2.1.1 Routing Protocols in Ad Hoc Networks). It does not require an end-to-end to route from source to destination to be known before it begins transmission of data as in topology-based routing approaches, and it does not use location devices to ascertain destination locations; as in position-based routing. The AD mechanism can, therefore, be seen as a fresh approach to routing in ad hoc networks.

3.1.1 Naming of Network Nodes

The proposed AD mechanism uses a static naming convention for naming network nodes. Each network aware software agent has a node from which it originates, in the AD mechanism these origin nodes are called *hosts*. Each host is given a globally unique identifier or *host-name*. The concept used to assign host-names is similar to the naming concept used in friend-to-friend networks such as SKYPE [43]. In such networks a static name is applied to a users account and some external means is used to propagate the account name of a given user to other system users (e.g. via simple person to person communication) in order to allow two users to communicate. In the AD mechanism the host-name of a certain host will be propagated using similar external means.
An implementation of this naming concept might involve a host registering their host-name with a central authority that has records of all host-names for a given network. This would halt the same host-name being assigned to multiple hosts. To aid system efficiency names could then be hashed using a hash function (e.g. SHA-1). Therefore as each host can be assigned its own globally unique identifier in the form of a host-name, naming concepts, such as IP addresses, for underlying protocols, such as the Internet Protocol (IP), for a host can dynamically change over time as a hosts' IP address is not used by the AD mechanism to identify it. This allows hosts to be completely mobile and the need to register with new networks and unregister with old networks is not required by the AD mechanism.

3.1.2 Agent Distribution Mechanism Message Types

Within the AD mechanism there are four types of message which may be sent by network agents to assist in agent routing within the ad hoc network in question. These messages types are as follows:

1. Agent-Query
2. Population-Query
3. Population-Reply
4. Agent-Reply

The following subsections explain each of these message types in turn.

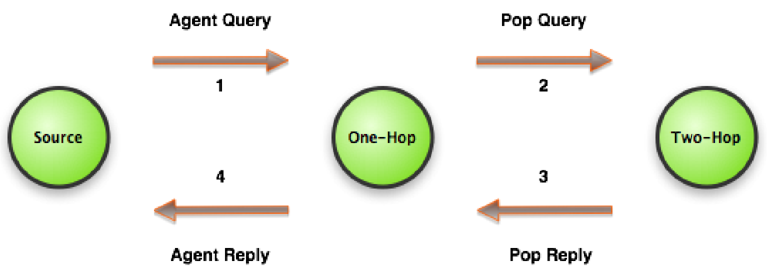


Figure 3.1: AD mechanism overview

Agent-Query

Agent-query messages are sent by a software agent to hosts that are currently its one-hop neighbour. In Fig 3.1 the Source host/node contains the querying software agent who sends an agent-query message to its one-hop neighbour. Agent-query messages contain the following information:

- The host-name of the current host on which the software agent is operating.
- The host-name of the destination host.
- The IP address of the current host on which the software agent is operating, if IP is used as the underlying transport protocol.

Population-Query

Population-query messages are sent by the one-hop neighbour(s) of the querying software agent to its current one-hop neighbours (or the two-hop neighbours of the querying software agent). In Fig 3.1 the One-Hop host/node sends a population-query message to its one-hop neighbour, host/node Two-Hop. A population-query message can be seen as a light weight ping message that is only used to ascertain host/node populations for a given area. Population-query messages contain the following information:

- The host-name of the agent's one-hop neighbour sending the population query message.
- The IP address of the host sending the population-query messages, if IP is used as the underlying transport protocol.

Population-Reply

Population-reply messages are sent by the querying software agent's two-hop neighbours in receipt of a population-query message. In Fig 3.1 the Two-Hop host/node sends a population-reply message back to the One-Hop host/node after receipt of a population-query message. A population-reply message allows the querying software agent's one-hop neighbour to ascertain a host/node population/neighbourhood count and to alert the querying software agent as to whether the required destination host is within it's population/neighbourhood group. Population-reply messages contain the following information:

- The host-name of the host sending the population-reply message.
- The IP address of the host sending the population-reply message, if IP is used as the underlying transport protocol.

Agent-Reply

Agent-reply messages are sent by a querying software agent's one-hop neighbour in receipt of an agent-query message, after the one-hop neighbour has performed a population/neighbourhood count for its population/neighbourhood group. In Fig 3.1 the One-Hop host/node sends a agent-reply message back to the Source host/node after steps 1 - 3 have been performed. Agent-reply messages contain the following information:

- The host name of the agent's one-hop neighbour.
- The population count of the agent's one-hop neighbour.

- A boolean value indicating whether or not the destination host is directly reachable via this one-hop neighbour, i.e. via two-hops with the host that sent the agent-reply message being the first hop.
- The IP address of the host sending the population-reply message, if IP is used as the underlying transport protocol.

The above message types for the ABR protocol are exchanged between agents operating on network hosts/nodes, and as such, depending on the performance constraints of the network hosts/nodes, should be implemented in an agent communication language such as KQML [19] or FIPA ACL [20].

3.1.3 AD Mechanism Operation

There are five stages to successful operation of the AD Mechanism. Fig 3.2 gives an overview of this operation.

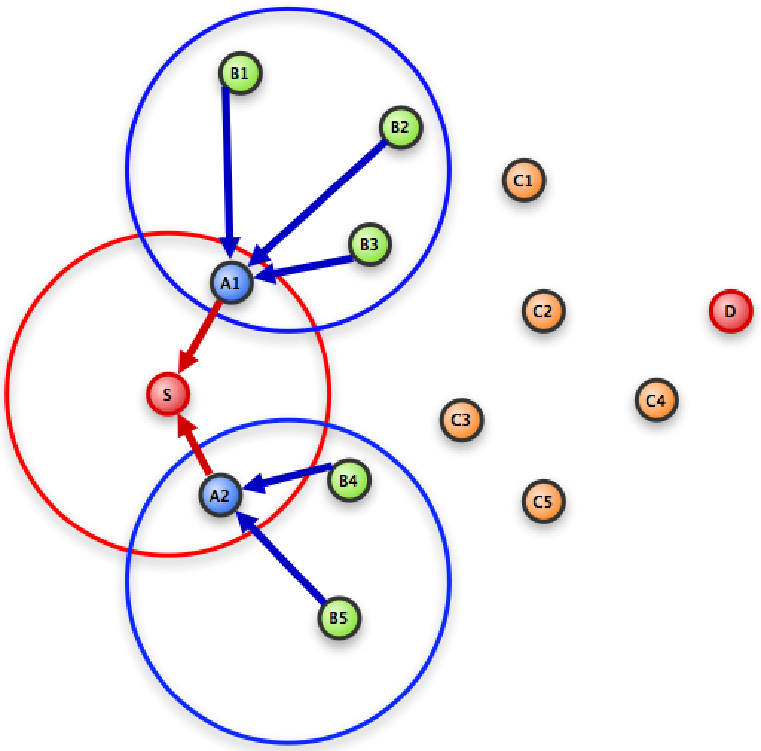


Figure 3.2: AD mechanism operation

1) As seen in Fig 3.2 host/node S wishes to find host/node D. A software agent operating on host/node S broadcasts an agent-query message to its one-hop neighbours, depicted in Fig 3.2 by a red ring around host/node S. This agent-query message is received by software agents operating hosts/nodes A1 and A2.

2a) If host/node A1 or host/node A2 were the required destination host/node then an agent-reply message would be sent straight back to the querying software agent operating on host/node S informing it of this. The querying software agent would then migrate to either host/node A1 or host/node A2, depending on the value of required destination host/node.

2b) In this scenario the required destination host/node is D. Therefore software agents operating on hosts/nodes A1 and A2 broadcast a population-query message to their one-hop neighbours, depicted in Fig 3.2 by a blue ring around hosts/nodes A1 and A2.

3) Software agents operating on hosts/nodes B1, B2 and B3 receive host/node A1's population-query message broadcast, and software agents operating on hosts/nodes B4 and B5 receive host/node A2's population-query message broadcast. Software agents operating on hosts/nodes B1, B2, B3 then reply directly to the software agent operating on host/node A1, and software agents operating on hosts/nodes B4 and B5 reply directly to the the software agent operating on host/node A2, with a population-reply message, as depicted by blue arrows in Fig 3.2. This informs the software agents operating on both host/node A1 and A2 of their population/neighbourhood count.

4) Software agents operating on both host/node A1 and A2 then reply to the querying software agent operating on host/node S with a agent-reply message which contains the population/neighbourhood count for each host/node and a boolean indicating whether or not either host/node A1 or A2 can act as a gateway or stepping-stone to the required destination host/node D.

5) As Fig 3.2 shows neither host/node A1 or A2 is a gateway or stepping-stone to the required destination host/node D, so the software agent operating on host/node S will then migrate to the one-hop neighbour with the highest population/neighbourhood count, that being host/node A1 in this scenario. Host/node A1 is chosen as the greater the number of nodes in an area of the network the greater the chance of successful routing of the software agent.

3.1.4 Software Agent Replication

Software agents that use the AD mechanism have the ability to replicate themselves, thus allowing multiple agents with the same or different tasks to traverse an ad hoc network. The keyword *replicate* was used, as opposed to the keyword *clone*, to describe such behaviour as replicated agents may not be an exact copy of the software agents that they were replicated from. For example, replicated software agents may only contain a subset of the resources, i.e. data and capabilities, of their replicant agent. Replication may be desirable in two cases, firstly if rapid routing of software agents is required, and secondly if a task is overly complex and the resources of single software agent are not great enough to complete it.

Rapid Routing

As described in the AD mechanism above if a software agent's one-hop neighbours return the same population values then the agent must make a choice. They can either:

a) Choose to ignore the problem and go with the neighbour who returned the highest neighbour population value first or last depending on the nature of the ad hoc network (i.e. static or mobile).

b) Choose to replicate itself and have each replicated software agent migrate to one of the possible one-hop neighbours with population values equal to the maximum.

Choice b) increases the possibility of one of the software agents finding the required destination host/node in the best time possible, especially if the software agent has more than one destination to locate.
For example, if a software agent has to route to multiple locations it may decide to replicate itself multiple times before it starts to seek any of the locations, and have each software agent route to a single location. This increases overall system efficiency. In Fig 3.3 this scenario is depicted, with software agent A1 operating on host/node S. The software agent is given the task of routing to hosts/nodes D and Q. In this scenario the software agent replicates itself, thus creating software agent A2. Each software agent is then given a single location to route to.

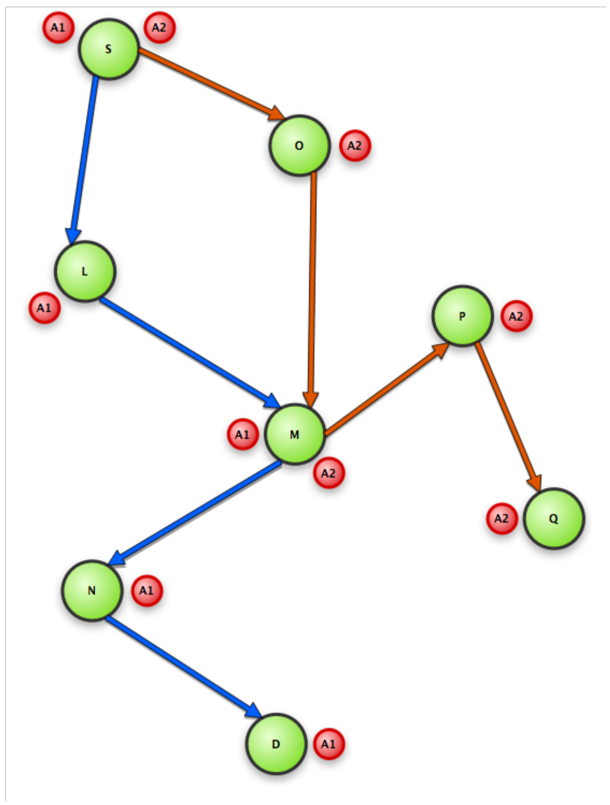


Figure 3.3: Software Agent Replication for Rapid Routing

Complex Task

If a software agent is given a complex task which can be divided into individual sub-tasks then a software agent may "choose" to replicate itself in order to create more resources to complete the given task.

Fig 3.4 shows such a scenario, where a software agent A operating on host/node S is given a complex task that may be broken down into two or more logical sub-tasks. The result is the software agent A replicating itself and creating the software agent B. A and B are then given separate tasks. In this scenario the task given to software agent A involves routing to host/node D and performing some action and the task given to software agent B involves routing to host/node Q and performing some different action. This division of software agents and assignment of individual tasks increases the overall performance of the system.

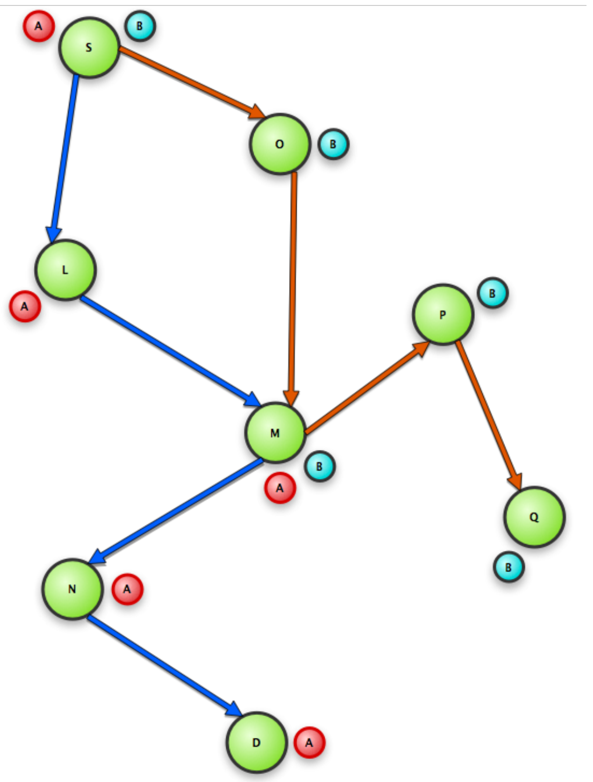


Figure 3.4: Software Agent Replication for Complex Task

3.1.5 Software Agent Merging

As software agents in the ABR protocol have the ability to replicate themselves, the opposite behaviour is required for the ABR protocol to work efficiently. Merging of software agents or “self-extinction” allows multiple agents to become one. An example of where this behaviour might be desirable would be when resources in the system become constrained and an abundance of active software agents has a detrimental effect on system performance.

Post Complex Task/ Re-Finding S

If multiple software agents or an *agent swarm* have completed a complex task they may be required to return to a central host/node in order to relay the acquired information. If the end location needs to wait for all agents in the agent swarm to return before taking decisive action, enabling software agents to merge and return as a single entity would increase system efficiency, as it reduces the possibility of single agents “getting lost”.
Fig 3.5 depicts this scenario whereby software agents A and B operating on hosts/nodes D and Q respectively, both have to route to host/node S to complete their assigned task. The software agents meet up at host/node M. Upon the realisation that both software

agents are in the final stages of execution and must return to host/node S to complete their tasks, the two software agents may decide to merge, thus creating a single software agent X. X can then then route to host/node S as a single software agent. The system must wait for both software agents to return before it can proceed. If the merged software agent X takes a non-optimal route to host/node S, i.e. "gets lost", for example via the hosts/nodes M - P - O - L - S, then the maximum time waited would be equal the time taken if one of the non-merged software agents had "gotten lost". Therefore merging of software agents can be seen as an optimistic approach to increase system performance.

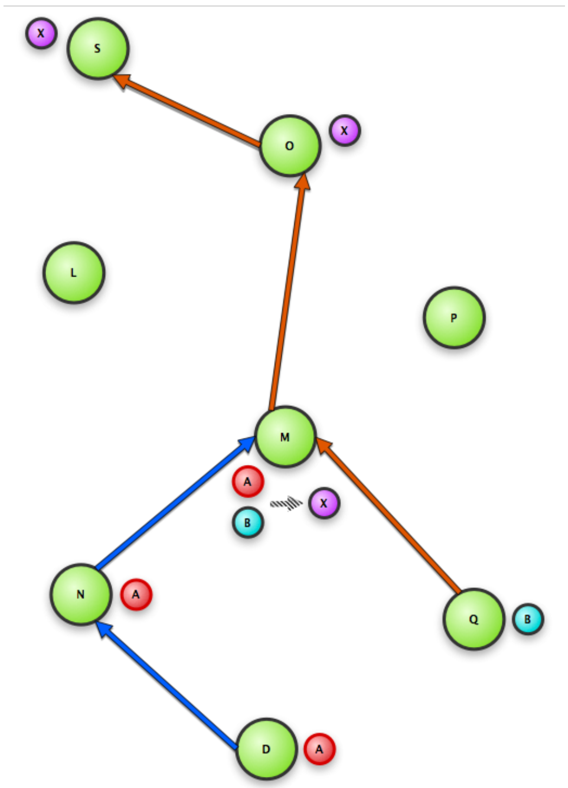


Figure 3.5: Software Agent Merging

3.2 Summary

In this chapter, we discussed an optimistic on-demand/reactive routing mechanism for distributing software agents and by extension, the data which they carry, throughout an ad hoc network. The AD mechanism operational semantics were formalised and discussed thus allowing the design of this mechanism to be conveyed to readers.

Chapter 4

Implementation

To prove the AD mechanism could be successfully used to route data in an ad hoc network, a simple message delivery system entitled Shoulder Monkey was developed. This system was agent-based and used the AD mechanism as the method of allowing software agents to jump from node to node in an ad hoc network. The operation of the system consisted of a software agent been given a message, which it then had to relay to a given node in an ad hoc network. Once this message was relayed, the software agent waited for a reply and then re-routed to its originating node to relay this reply.
This chapter discusses the design and implementation of the Shoulder Monkey system in detail.

4.1 Shoulder Monkey System Design

The design architecture for the Shoulder Monkey system needed to be modular; allowing easy separation of system concerns, easy to understand; allowing future researchers to add new features to the system and as light weight as possible; keeping the use of system resources to a minimum. The design architecture for the system went through many iterations and the result is shown in Figure 4.1. A high level view of the design architecture for the Shoulder Monkey system shows that the system met the criteria set out above.

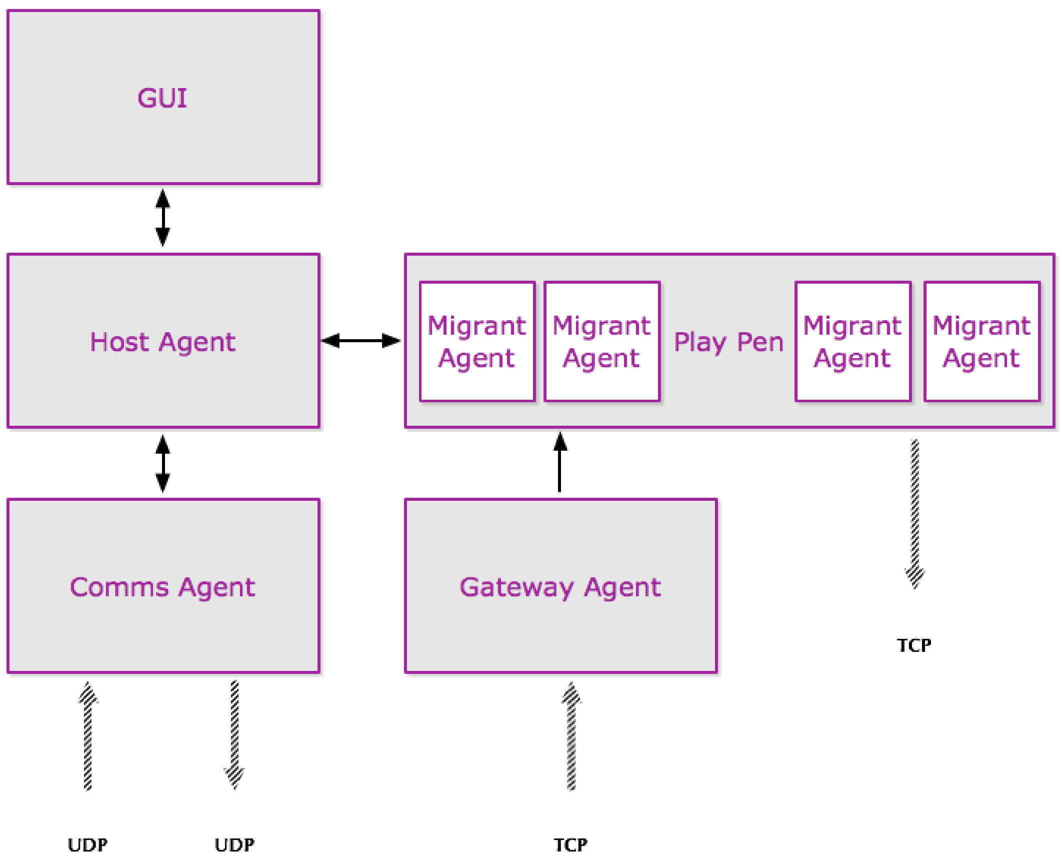


Figure 4.1: Design Architecture for Shoulder Monkey system

4.1.1 Host Agent

The Host Agent component in the Shoulder Monkey system was responsible for maintaining data relating to the node on which the system operated. The Host Agent contained the following information:

- The host name of the node.
- The IP address of the node.

This information could be accessed by components such as the Play Pen, the Comms Agent and Migrant Agents as was needed.
Creation of Migrant Agents was the responsibility of the Host Agent component. Once a Migrant Agent was created by the Host Agent, it was passed to the Play Pen where it began executing. The Host Agent also acted as the intermediary component to enable messages to be passed from Migrant Agents operating in the Play Pen component to the GUI component. This design decision avoided concurrency issues which may have arisen from allowing Migrant Agents direct access to the GUI component.

4.1.2 Play Pen

The Play Pen component was designed to act as a holding area for Migrant Agents. Newly created Migrant Agents were passed to the Play Pen component by the Host Agent component. Migrant Agents that migrated from other nodes were passed to the Play Pen component via the Gateway Agent component. Once a Migrant Agent began executing it was required to register with the Play Pen component. If a Migrant Agent self-terminated, merged with another Migrant Agent or migrated to another node, it was required to unregister with the Play Pen component before doing so. This allowed the Play Pen component to keep count of how many Migrant Agents were operating within it.

4.1.3 Migrant Agent

Once a Migrant Agent had been created by the Host Agent it executed entirely in the Play Pen component of the system. The Migrant Agent was equipped with an implementation of the AD mechanism to enable it to traverse an ad hoc network. The Migrant Agent contained algorithms to enable it to ascertain which network nodes it must migrate to, and whether or not it was currently operating on said nodes. Migrant Agents also had the ability to replicate themselves, merge with other Migrant Agents and self-terminate. The Migrant Agent could leave the Play Pen at any time and of its own merit, but was required to enter Play Pen components on other nodes via their Gateway Agent component.

4.1.4 Gateway Agent

The Gateway Agent component acted as the doorman of the Shoulder Monkey system. As discussed in Chapter 2 (section 2.1.2 Security Considerations for Ad Hoc Networks) security considerations are coming to the forefront of research in ad hoc networks. Many ad hoc networks and protocols trust that all nodes operating within an ad hoc network are benevolent, which may not be the case. As the Shoulder Monkey system operated by transferring executable code bases from node to node; i.e. the systems migrant agents, security concerns had to be taken into account in the design architecture. Even though the security considerations for an MAS approach to ad hoc routing were beyond the scope of this dissertation, it was the opinion of the author that any system that was designed to be potentially deployable in real world situations should show a knowledge of security issues in their design. Therefore the Gateway Agent component was designed to allow the easy addition of security checks such as digital signatures and cryptographic keys, to be run on Migrant Agents before they are allowed to begin executing, thus impeding malicious agent attacks on the system.

4.1.5 Comms Agent

The Comms Agent component was the most unusual design feature of the Shoulder Monkey system. This component provided an automatic reply system for broadcasted Agent-Query and Pop-Query messages. When a Migrant Agent broadcasted an Agent-Query message, the Comms Agent component received it, and sent back a reply automatically, without the need to communicate with other system components. The Comms Agent component was also responsible for sending and responding to Pop-Query messages, sent to/from Comms Agent components operating on other nodes. This enabled population counts to be ascertained for nodes.

4.1.6 GUI

The GUI component for the Shoulder Monkey system was designed to be as user friendly and informative as possible. As a proper evaluation of the AD mechanism was the reason for developing the Shoulder Monkey system, AD mechanism operation needed to be shown in the finished GUI component. Also as many academic research projects are easily forgettable to say the least, an attractive GUI that "catches one's eye" was seen a key feature to attract the attention of fellow researchers, much like the way magpies would be attracted to shiny objects.

4.2 Shoulder Monkey System Implementation

The Shoulder Monkey system was implemented entirely in java. A number of java based frameworks [36] [40] were considered for the development of the system, however these frameworks came with an inherent level of complexity, and as the Shoulder Monkey system was designed to be as light weight as possible, it was decided not to use a framework to develop the system.

4.2.1 AD mechanism

The AD mechanism was implemented at application layer in a single java class aptly named AgentDistributionMechanism. This class enabled the AD mechanism to operate. The class consisted of four methods which:

1. Sent an Agent-Query message.

2. Sent a Pop-Query message.

3. Sent a Pop-Reply message.

4. Sent an Agent-Reply message.

Agent-Query and Pop-Query messages were broadcasted via the use of a UDP multicast group. Figure 4.2 shows a code snippet of the constructor method for the AgentDistributionMechanism class. This code snippet shows the UDP multicast group being setup thus allowing broadcasting of messages.
Four more methods from the AgentDistributionMechanism class allowed the payload for all message types to be set. For example, the method used to set the payload for an Agent-Query message took the host name of the node sending the Agent-Query and the host name of the node being sought as arguments. These values were then concatenated into a single string; separated by a space and broadcasted to the UDP group.

```
public AgentDistributionMechanism()
{
       group = "230.0.0.5";

       try
       {
              myIP = InetAddress.getLocalHost().getHostAddress();

       }       catch(UnknownHostException e)       {

               e.printStackTrace();
       }
}
```

Figure 4.2: Code snippet of AgentDistributionMechanism class constuctor

4.2.2 Comms Agent and Stop Clocks

The CommsAgent class was used to implement the Comms Agent component of the Shoulder Monkey system. Figure 4.3 shows a code snippet from the CommsAgent run() method. This snippet shows that both Agent-Query and Pop-Query messages were continually listened for by the run() method. Upon receiving a message of either type a new thread was started which sent an appropriate message i.e. an Agent-Reply or Pop-Reply message, in reply.
For example, when an Agent-Query message was received, a new thread was started which

broadcasted a Pop-Query message via a UDP multicast. A stop clock was then started to give nodes a time frame of 50 ms to reply to the broadcast. A code snippet for this stop clock is shown in Figure 4.4. Once the stop clock was up an Agent-Query message was sent to the *locahost* to inform the system to proceed. All Pop-Reply messages were then processed and an Agent-Reply message, containing information on located destinations, neighbourhood sizes, etc, was sent to the node the Agent-Query originated from. Pop-Query broadcasts were also replied to in the same manner, however stop clocks were not used. This enabled Pop-Reply messages to be sent directly back to a querying node, via a UDP unicast, upon receipt of a Pop-Query message.

```
public void run()
{
        //       Listen for PopQuery msg's
        Thread pop_query_listener = new Thread(new PopQueryListener());
        pop_query_listener.start();

        System.out.println("COMMS AGENT : ACTIVE");

        //       Listen for AgentQuery msg's
        AgentQueryListener dave = new AgentQueryListener();

        while(true)
        {
                dave.listen();

                //       Start a thread to send a reply
                Thread AgentQueryListener = new Thread(
                               new AgentQueryListener());
                AgentQueryListener.start();
        }
}
```

Figure 4.3: Code snippet of CommsAgent class run() method

```
int current_time = (int) System.currentTimeMillis();
int end_time = (int) System.currentTimeMillis();

while(current_time < end_time)
{

try
{
        Thread.sleep(10);

        }        catch(InterruptedException e)        {

                 e.printStackTrace();
        }

        current_time = (int) System.currentTimeMillis();
}

AgentReplySender.setWait(false);
stephen.sendPopulationReply("localhost", HostData.getHostID());
```

Figure 4.4: Code snippet of stop clock used by CommsAgent class

4.2.3 Migrant Agent

The migrant agent was implemented in the MonkeyAgent class. This class allowed migrant agents to traverse an ad hoc network. The implemented migrant agent had two operation modes, namely:

1. findD mode
2. findS mode

When a migrant agent was in findD mode it sought the destination it had been given to relay a message to. When a migrant agent was in findS mode it sought it's source node, or the node which it originated from, in order to relay a reply to the message sent. Migrant agents operated differently depending on which operation mode they were operating in. To ensure that migrant agents successfully jumped from node to node all migrant agent migration was performed using TCP.

Scout class

The scout class removed the AD mechanism logic from the MonkeyAgent class. This allowed the the MonkeyAgent class to be concerned with where it was and where it was going and not how to get there. Whenever a MonkeyAgent object wished to locate a new node to jump to it created a Scout object and then called its seek() method. This method returned a ArrayList of destinations to the MonkeyAgent object. Depending on whether or not the MonkeyAgent object was in replication mode and on whether multiple destinations had been located, the first entry of this ArrayList, or multiple entries were chosen as the next hop for the MonkeyAgent object. The seek() method in the Scout class followed a number of steps in order to locate nodes that a MonkeyAgent object could jump to, namely:

1. Sending an Agent-Query message.
2. Receiving back Agent-Reply messages.
3. Processing Agent-Reply messages.
4. Using a built in algorithm to decide which node(s) to jump to next.

The first of these steps broadcasted an Agent-Query message to nodes within range of the querying node. A hop-timer, similar to the stop clock timer used by the CommsAgent class, allowed a time frame of 125 ms for nodes to ascertain a population count for their neighbourhoods and send back an Agent-Reply message via a UDP unicast, thus completing step two. Step three then processed all Agent-Reply messages, as can be seen in Figure 4.5. The payload for all Agent-Reply messages was divided into separate strings which were stored in a string array. This allowed easy access to the data contained in each payload during step four. To complete step four an algorithm was used to run a number of checks on the Agent-Query messages received. Using the data processed in step three the algorithm checked the following for each Agent-Query message:

1. If the IP address was 127.0.0.1.
2. If a one-hop neighbour was the required destination.
3. If the required destination was found by a one-hop neighbour.
4. Which one-hop neighbour returned the largest neighbourhood count.

Depending on which of these conditions were met, different choices were made for the next node a migrant agent jumped to.

```
tuples = new String[agentReplies.size()][4];
String alan;


for(int i = 0; i < agentReplies.size(); i++)
```

```
{
        alan = agentReplies.get(i);

        StringTokenizer carol = new StringTokenizer(alan);

        //       Hostname of one hop neighbour
        tuples[i][0] = carol.nextToken();

        //       Destination found boolean
        tuples[i][1] = carol.nextToken();

        //       Population count
        tuples[i][2] = carol.nextToken();

        //       Sender IP
        tuples[i][3] = carol.nextToken();
}
```

Figure 4.5: Code snippet of Agent-Reply message processing

Agent Replication

As discussed in Chapter 3, software agents which use the AD mechanism to traverse an ad hoc network should have the ability to replicate themselves, thus allowing the creation of more resources as is needed. This feature was implemented in the Shoulder Monkey system and could be turned on or off by a system user from the GUI. When this feature was turned on, an agent was said to be in cross-roads mode. When an agent was created with the ability to replicate itself, it did so when two or more possible routes to a destination node were discovered. Upon the discovery of possible multiple routes to a destination an agent would replicate itself the exact number of times needed to send an agent on each route. To avoid the network becoming completely flooded all replicated agents were created without the ability to replicate, therefore only the original agent could replicate itself. Agents did not replicate when searching for their source node. Figure 4.6 shows a code snippet for the method used to replicate migrant agents.

This method took a string, which is the node the replicated agent should migrate to immediately upon creation, as an argument. This method then created a name for the replicated agent, which was a concatenation of the original agent's *agent name*, the letter

R to indicate that this agent was a replica, and an *integer* relating to the number of replicas the original agent had created. The method then created a new agent, using a different constructor than the one used by the HostAgent class and added it to the Play Pen. Once this was completed the original agent set its self-replicated status to true, indicating that it had created at least one replica.

```
private void selfReplicate(String nextHop)
{
        String repName = agentID + "R" + replicaCounter;
        replicaCounter++;

        PlayPen.add(new MonkeyAgent(repName, homeHostID,
             desHostID, desHostIDList, msgList.get(0), nextHop));

        hasSelfReplicated = true;
}
```

Figure 4.6: Code snippet of MonkeyAgent self-replication method

Agent Merging

As was also discussed in Chapter 3, agents using the AD mechanism should have the ability to merge with other agents to allow resources to be saved and/or freed up. The feature of agent merging was implemented in the Shoulder Monkey system. Two situations called for agents to merge when:

1. Agents replicated from the migrant agent, or said migrant agent, encountered each other before finding their destination.
2. Agents returning to the same source node encountered each other en route.

To enable this merging behaviour, an algorithm was developed, which consisted of four steps:

1. Check for other migrant agents.
2. Message said agents.
3. Pass on payload.
4. Self-terminate.

The following scenario explains how agent merging works in the Shoulder Monkey system. Imagine two migrant agents, agent X and agent Y operating on the same node. When agent X arrived at a new node and began executing in the Play Pen, it began the merging algorithm if one of the two conditions were met:

1. The agent had replicated or was a replica.

2. The agent was in findS mode.

If either of these conditions were met, Agent X checked if any other migrant agents were operating in the same Play Pen. It did this by calling the preMerge() method in the MonkeyAgent class. A message was sent to agent Y as it was operating in the Play Pen, using the chat() method in the MonkeyAgent class. This method enabled agent Y to check if it was eligible to be merged with. If agent Y was eligible it returned a boolean value of true. Agent X then called the merge() method to pass on its payload to the agent Y. Once the payload was passed, agent X self-terminated.
Figure 4.7 shows a code snippet from the chat() method. The first if loop checked if agent Y was a replica of agent X. It did this by checking if agent Y was in findD mode and then whether or not agent names were of the same form i.e. if they both started *MonkeyAgent_Z@W*. If they did, agent Y set its can merge status to false and self-terminated. This resulted in agent X being the only operating agent in the Play Pen and was then allowed to continue searching for its destination. If the condition for the first if loop was not met, the next if loop checked if both agents were in findS mode and whether or not they originated from the same node. If these conditions were met, agent Y set its can merge status to true and returned. If none of the above conditions were met agent Y set its can merge status to false and returned.
Once a true value had been returned from the chat() method, agent X could then take steps to merge with agent Y. Figure 4.8 shows a code snippet of the merge method from the MonkeyAgent class. This method enabled agent X to pass on its payload to agent Y before self-terminating. This approach of *shallow merging*, whereby agent X only passed on some information to agent Y, was chosen over a *deep merging* approach. A deep merging approach would entail passing the entire agent X object to agent Y. This was seen as a waste of resources as both agents were equipped with the same operational algorithms. This would have increased the overhead of transferring agents from node to node without any benefit to the Shoulder Monkey system. The shallow merging approach allowed agents to only pass on relevant information when merging with other agents and was therefore seen as a more optimal approach.

```
boolean canMerge;
```

```
//       If in findD mode and of the same MonkeyAgent/Replica
if(findD && localName.equals(externalName))
{
       canMerge = false;

       System.out.println();
       System.out.println("MONKEYAGENT " + agentID + "
               : Replica in findD mode exists on host - self-terminating");

       this.selfTerminate();
}
//       If in findS mode and from the same host
else if((findD != true) && homeHostID.equals(hostName))
{
       canMerge = true;
}
       else
{
       canMerge = false;
}

return canMerge;
```

Figure 4.7: Code snippet of MonkeyAgent class chat() method

```
private void merge(MonkeyAgent abu)
{
       abu.lastRights(msgList);
       System.out.println();
       System.out.println("MONKEYAGENT " + agentID + " :
       MERGED WITH : " + abu.getAgentID());
                    this.selfTerminate();
}
```

Figure 4.8: Code snippet of MonkeyAgent class merge() method

Msg class

The Msg class contained the payload that a migrant agent carried. Each Msg object contained a message, a message-reply and a message-id. The class also contained getter methods for this data, and a setter method to allow a message-reply to be added to a Msg object. Figure 4.9 shows the methods of the Msg class. Msg objects were created by passing an integer containing the message id number and a string containing the message to be relayed to the Msg class constructor.

```
public Msg(int id, String s)
{
       msgID = id;
       content = s;
}

public String getContent()
{
       return content;
}

public void setReply(String hostID, String msg)
{
       destinations = destinations + " " + hostID;
       destinationReply = destinationReply + "*" + msg;}

public String getReply()
{
       String all = destinations + "\%" + destinationReply;

       return all;
}

public int getID()
{
       return msgID;
}
```

Figure 4.9: Code snippet of Msg class methods

MonkeyAgent run() Method (run() Monkey run())

The behaviour of the migrant agent was implemented in the run() method of the MonkeyAgent class. This method enabled multiple migrant agents to operate on the same platform simultaneously in their own thread. As discussed previously, the operation mode; findD or findS, a migrant agent was in, would have an effect on its behaviour. The first task a migrant agent carried out, regardless of what mode it was in, was to add the current node it was operating on to an ArrayList which served as a record of all the nodes the migrant agent had visited. If the migrant agent was in findD mode it would then check if the node it was currently operating on was the required destination. If it was, the migrant agent would relay its message to the Host Agent. Figure 4.10 shows a code snippet of how this was achieved. The migrant agent would pass its Msg object and a reference to itself to the Host Agent. The Host Agent would then process the message while the migrant agent waited. If the message had already been relayed to the Host Agent by a replica migrant agent, the Host Agent would terminate the current migrant agent. Once the migrant agent received a reply it would enter findS mode and route to its original source node. The same manner of routing was used by the migrant agent to route to its source node as was used to route to the destination node. However once the migrant agent located it's source node and relayed it's reply, it self-terminated.
Figure 4.11 shows a code snippet for the algorithm used by the migrant agent to traverse an ad hoc network. A migrant agent first checked if agent merging was possible, as discussed in previous sections. Once this had been completed, the migrant agent called its scout method, which used an instance of the Scout class, as discussed in previous sections, to find a node to migrate to. Once a node was found, the migrant agent used the jump() method to migrate to the selected node. Fault tolerance was built into the implemented migrant agent jump behaviour using a boolean value, which ensured the process of finding a node to migrate to and jumping to it via TCP was re-tried until it was completed successfully.

```
if(this.isCurrentHostDes())
{
        //      Relay MSG
        HostAgent.relayMsg(this, msgList.get(0));


        //      Where did we visit
        System.out.println();
        System.out.println("MONKEYAGENT " + agentID + " : Hop-List");
        for(int u = 0; u < visitedHostIPList.size(); u++)
        {
                System.out.println("HOP " + u + " : " +
                           visitedHostIPList.get(u));
        }

        //      Wait for reply
        while(waitReply)
        {
                try
        {
                Thread.sleep(1000);

        }       catch(InterruptedException e)       {

                e.printStackTrace();
        }
}
        //      No longer looking for D
        //      Set home destination to S
        this.getNextDestination();
}
```

Figure 4.10: Code snippet of MonkeyAgent class Msg object passing

```
this.preMerge();
if(alive != true)
{
        break;
}
//      If not @ home
this.scout();
this.jump();
```

```
//      If TCP connection fails try until it is successful
while(trapped)
{
        this.scout();
        this.jump();
}
```

Figure 4.11: Code snippet of MonkeyAgent routing algorithm

4.2.4 Host Agent

The HostAgent class was used to implement the Host Agent component in the Shoulder Monkey design architecture. This class consisted of methods to allow creation of new migrant agents, importing of created agents to the Play Pen and passing of Msg objects between the GUI component and migrant agents operating in the Play Pen. Figure 4.12 shows a code snippet of the methods used for migrant agent creation and importing migrant agents into the Play Pen. The first method; createNewMonkeyAgent(), created a new migrant agent. The method achieved this by taking an ArrayList, which contained the destination(s) that a migrant agent had to route to, and a string, which contained the message the migrant agent must relay to said destination(s), as arguments. The method then created a Msg object containing the message string. This Msg object was given an ID consisting of a concatenation of the host name of the node in question and a message count number, which was then hashed using a hashing function. The method then created a new migrant agent which it imported into the Play Pen. Once the agent was in the Play Pen its run() method was started and it began executing.
Passing of messages from migrant agents in the Play Pen to the GUI was achieved by a migrant agent passing a reference to themselves as well as an Msg object as arguments to the Host Agent. Then when the GUI contacted the Host Agent with a reply, the reference to the migrant agent was used to ensure the newly modified Msg object was passed to the correct migrant agent operating in the Play Pen.
The Host Agent component was also responsible for maintaining information relating directly to the node on which the Shoulder Monkey system was operating. This information was contained in a class called HostData. The HostData class contained information such as the IP address of the node and the nodes host-name and provided methods to allow access to this information.

```
//      Creates a new MonkeyAgent
```

```
public static void createNewMonkeyAgent(ArrayList<String> des, String msg)
{
       int msgID = (HostData.getHostID() + Integer.toString
                   (msgCounter)).hashCode();
       Msg m = new Msg(msgID, msg);
       msgCounter++;

       PlayPen.add(new MonkeyAgent(HostAgent.generateNewAgentName(),
                   des, m, GUI.crossRoadsMode()));
}

//        Generates a new MonkeyAgent name
private static String generateNewAgentName()
{
       String name = Integer.toString(monkeyCounter);
       monkeyCounter++;

       return name;
}
```

Figure 4.12: Code snippet of HostAgent methods

4.2.5 Gateway

The Gateway Component of the Shoulder Monkey system was implemented in the GatewayAgent class. This class provided a method called standGuard(), which simply listened on a TCP port for MonkeyAgent objects and then passed them to the Play Pen to begin executing once they were successfully received. As the security considerations for MAS in ad hoc networks were beyond the scope of this dissertation, no security measures were implemented in the GatewayAgent class. However, as can be seen in Figure 4.13, a number of security checks could be easily implemented after the MonkeyAgent object is received and before it is added to the Play Pen where it begins executing. This is an example of how new features could be easily added to the Shoulder Monkey system due to its modular design architecture.

```
jane = new ServerSocket(PortNumbers.MIGRATING_AGENT_PORT);

Socket claire;
ObjectInputStream john;

while(true)
{
       claire = jane.accept();
       claire.setTcpNoDelay(true);
       john = new ObjectInputStream(claire.getInputStream());

       MonkeyAgent newbie = (MonkeyAgent) john.readObject();

       PlayPen.add(newbie);

       john.close();
}
```

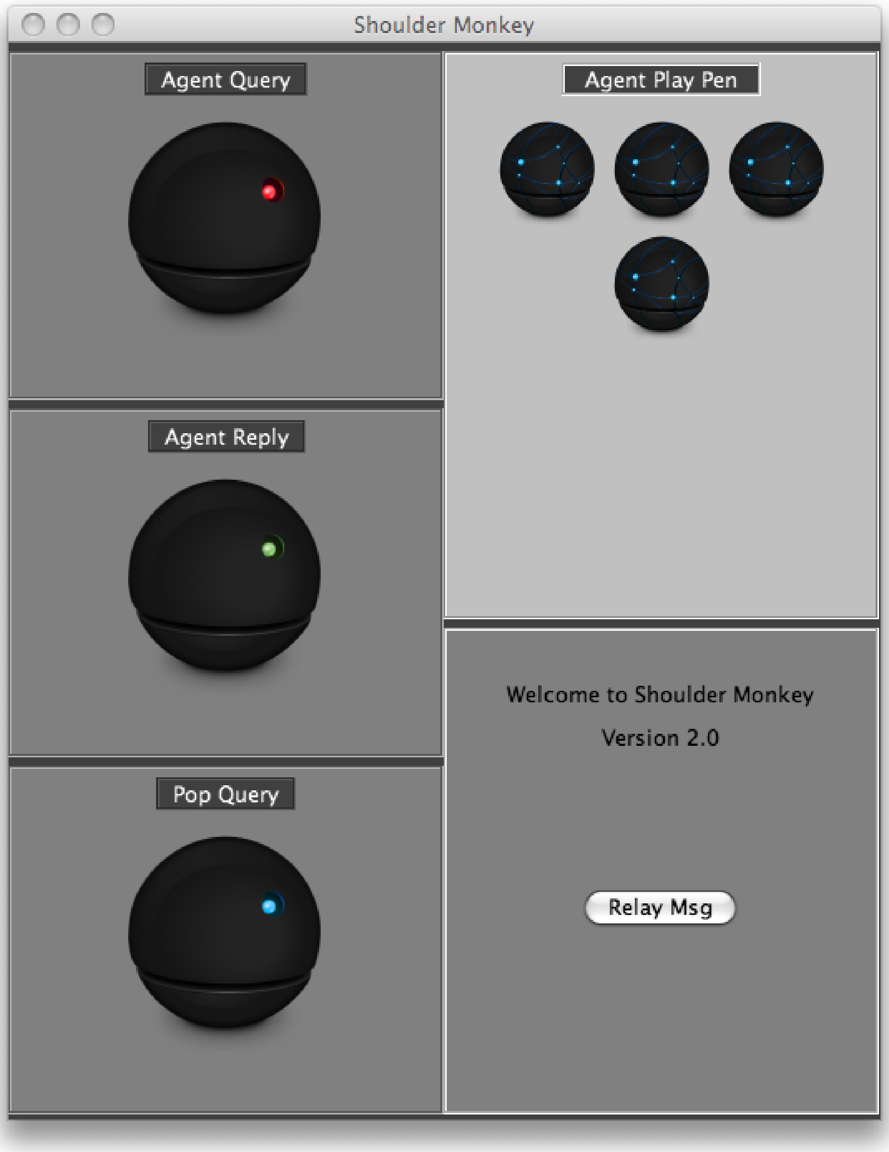
Figure 4.13: Code snippet of gateway agent operation

4.2.6 GUI

The GUI was implemented using java swing, and went through two stages of development. A rudimentary GUI was first developed to aid in the testing of a prototype of the Shoulder Monkey system. This GUI was then used to aid in the further development of the Shoulder Monkey system. Once the Shoulder Monkey system was operating efficiently a new GUI was developed, as can be seen in Figure 4.14. This new GUI allowed system users to create agents to send messages, turn the cross-roads replicating mode on/off, see at a glance how many migrant agents were present in the Play Pen and view AD mechanism operation.

As can be seen in Figure 4.14, on the left hand side of the GUI there are three spheres. Each of these spheres rotated when an AD mechanism message was sent. For example, the top sphere rotated when an Agent-Query message was sent, the middle sphere rotated when an Agent-Reply message was sent and the bottom sphere rotated when a Pop-Reply message was sent. This allowed system users to witness the AD mechanism in operation. In the top right hand corner of the GUI, as can also be seen in Figure 4.14, there is a graphical representation of the Play Pen component. The small black spheres in this section indicate at a glance how many agents are operating in the PlayPen. As migrant agents migrate to or leave the Play Pen these small black spheres increase and decrease in quantity.

The bottom right hand corner of the GUI allowed users to create migrant agents to send messages to other system users, as can be seen in Figure 4.14. Once a user clicked the button labeled “Relay Msg”, they were prompted with a text box to enter their message. Once a user entered a message and clicked a button to continue they were prompted with another text box to enter the destination or destinations of the network nodes they wished to relay the message to. Multiple destinations were inputed one after another, separated by blank spaces. Once a user entered the destination(s) a new migrant was created and displayed graphically as a small black sphere in the Play Pen section of the GUI. The migrant agent then began sending AD mechanism messages, which were represented graphically by the rotating spheres on the right hand side of the GUI.



Welcome to Shoulder Monkey

Version 2.0

Figure 4.14: Screen shot of Shoulder Monkey system GUI

4.3 Summary

In this chapter, we discussed the implementation of the AD mechanism in a simple message delivery system entitled Shoulder Monkey. A top level design architecture was given which outlined key components in the system. This was followed by a discussion of the implementation of the system.

Chapter 5

Evaluation

To properly evaluate any system testing must be carried out. This testing gives an insight into the performance abilities and constraints of the system in question. Therefore, to evaluate the Shoulder Monkey system discussed in chapter 4, a real world ad hoc network was deployed and the Shoulder Monkey system was run on this network. Quantitative data was collected during testing of the Shoulder Monkey system to enable an accurate evaluation of the system.
This chapter discusses the setup of the ad hoc network used as a test-bed to evaluate the Shoulder Monkey system and by extension the AD mechanism. This is followed by a discussion of the evaluation process and the results produced.

5.1 Test-Bed Network Setup

Each node in the testbed network consisted of a dell laptop D400/C400 [ts6]/[ts7] running Unbuntu 8.04 (hardy heron) [ts8]. Each node was equipped with a Netgear WPN111 wireless USB adapter [ts9] to act as its network connector. Static IP addresses were given to all network nodes in the form 169.254.230.9x, with network node one having the address 169.254.230.91, network node two having the address 169.254.230.92, etc. All network nodes also had the java 6.0 jre [ts10] installed on them. Network traffic for the testbed network was monitored using an apple iBook G4 laptop [ts11] running wireshark [ts12].

5.1.1 Installing Wireless Adaptors

The installation of the Netgear WPN111 wireless USB adaptors was a two part process that consisted of:

1. Installing the necessary software to operate the wireless adaptors.

2. Configuring the wireless adaptors to limit their range.

Netgear WPN111 Software

To enable the Netgear WPN111 wireless USB adaptors the following packages were downloaded and installed on each node:

- build-essential
- ndiswrapper
- windriver

A number of steps were then taken to install ndiswrapper on each node. Once ndiswrapper was installed the Windows drivers for the wireless adaptors could be installed and configured to enable the wireless devices. Finally ndiswrapper had to be added to the following file:

/etc/modules

This ensured that the wireless adaptor was initialised on system startup. A complete tutorial to install a Netgear WPN111 wireless USB adapter is available from the Ubuntu forum website [ts13].

Configuring Wireless Adaptors

As the Netgear WPN111 adaptors had a range of approximately 90 m [ts9] the range had to be greatly reduced. To achieve this, the wireless adaptors were firstly removed from their plastic casing as can be seen in Figure 5.3. An anti-static component bag and a single sheet of tin foil were then combined to limit the range of the wireless adaptors. As can be seen in Figure 5.1 and Figure 5.2 a sheet of tin foil was placed inside the anti-static component bag. The bag was then wrapped around the case-less wireless adaptor, as can be seen in Figure 5.4, which limited the range of the wireless adaptors to 1cm - 10 cm. The wireless adaptors were then connected to the network nodes and the ping command was used to ascertain the range of each node, which could then be physically moved in or out of range of other network nodes.

Figure 5.1: Sheet of tin foil and anti-static component bag

Figure 5.2: Sheet of tin foil and anti-static component bag combined

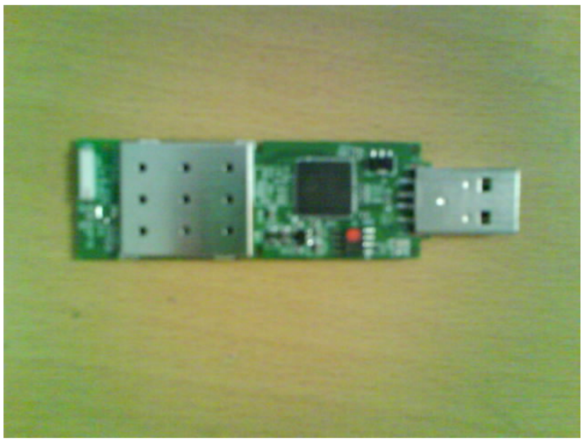


Figure 5.3: Wireless adaptor removed from plastic casing

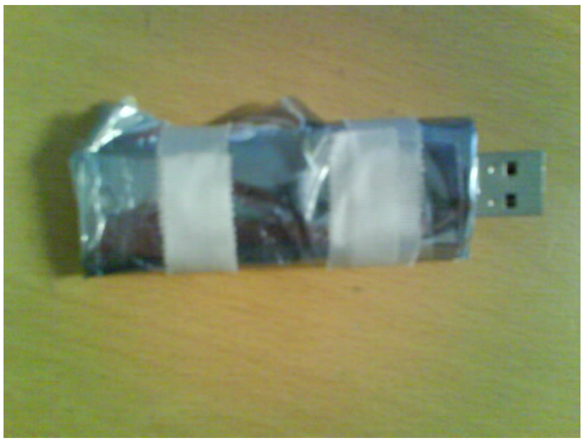


Figure 5.4: Covered wireless adaptor with limited range

5.1.2 Ubuntu Networking Problems

Two main problems were encountered when running tests on the Ubuntu platform:

1. Multicast problem
2. TCP problem

Multicast Problem

The Shoulder Monkey system used a UDP multicast to broadcast Agent-Query and Pop-Query messages to a UDP multicast group. However when the Shoulder Monkey system was executed on network nodes the following exception was thrown.

```
java.net.SocketException: No such device
```

After some research it became evident that the solution to the problem lay in one line of code, which added the default UDP multicast route to the routing table:

```
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev wlan0
```

TCP Problem

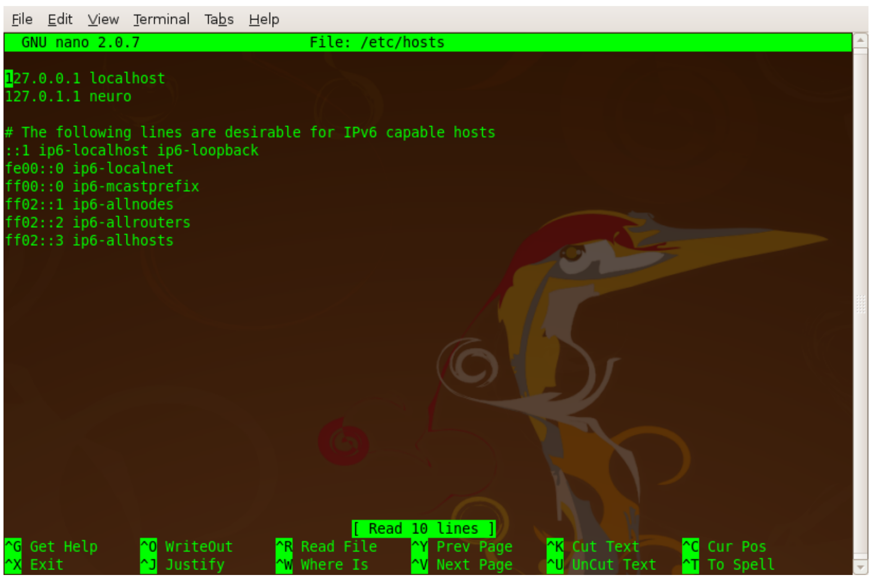
The IP address of each network node was statically assigned using the following command:

```
sudo ifconfig wlan0 169.254.230.9x
```

This however caused an error when the migrant agent had to jump from one node to another using a TCP socket. Once the migrant agent had detected the node which it wanted to jump to next, the migrant agent would set up a TCP connection and then the system would hang until it was restarted. The route of this problem lay in the way Ubuntu assigns two loopback IP addresses in the following file:

/etc/hosts

Fig 5.5 shows a screen shot of this file. On line two the second loopback IP address "127.0.1.1" is assigned to the node in addition to the loopback IP address "127.0.0.1" on the first line. Changing this second loopback IP address to the static IP address of the network node fixed this problem and allowed the Shoulder Monkey system to operate correctly.



```
  GNU nano 2.0.7                    File: /etc/hosts

127.0.0.1 localhost
127.0.1.1 neuro

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

                                         [ Read 10 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figure 5.5: Screenshot of /etc/hosts

5.2 Comparitive Tests

To enable a complete analysis of the Shoulder Monkey system, a number of comparative experiments were run using a simple TCP based routing approach and using an implementation of the AD mechanism in the Shoulder Monkey system.

5.2.1 Simple TCP Comparitive Test

A simple line topology was setup for comparitive testing of a simple TCP based routing approach. Channel 3 was used for radio transmissions in this ad hoc network, with a background noise level of -60 dBm. For this experiment set, each experiment run consisted of TCP data being generated and transmitted using the iperf tool [ts14]. Transmission of data was from Node 1 to Node 5, as seen in Figure 5.6. Five experiment runs were used to obtain results for this experiment set. Each node in the network was given a fixed gateway to pass data through, ultimately creating an ad hoc network with uni-directional links. All experiment runs were carried out for approximately 60 secs.

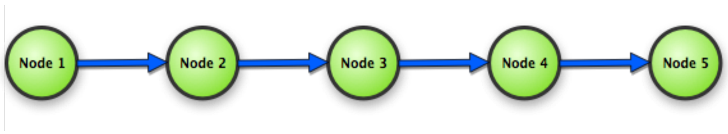


Figure 5.6: TCP line topology for 5 node ad hoc Network

Success Rate

Results showed that the performance of the TCP based routing approach in the created ad hoc network was less than perfect. Figure 5.7 shows the TCP throughput measured in MBytes, for a number of hops in the network. For a single hop transmission, i.e. transmitting from Node 1 - Node 2 as seen in Figure 5.6, TCP showed quite a good performance, averaging 4.63 MBytes. However as the number of transmission hops increased the performance of TCP in the ad hoc network rapidly decreased, averaging 1.96 MBytes for a two hop TCP transmission, i.e. transmitting from Node 1 - Node 2 - Node 3 in Figure 5.6, and just 0.15 MBytes for a three hop TCP transmission, i.e. transmitting from Node 1 - Node 2 - Node 3 - Node 4 as seen in Figure 5.6.
Results for a four hop TCP transmission, i.e. transmitting from Node 1 - Node 2 - Node 3 - Node 4 - Node 5 as seen in Figure 5.6, could not be obtained. However an analysis of network traces taken during the attempted four hop transmissions revealed an insight into a possible reason for this. The network traces showed that Node 1 would send out a TCP

syn packet to Node 5, in some runs it would receive a *syn-ack* packet back from Node 5, and then nothing else, in other runs no *syn-ack* would be received from Node 5. A possible explanation for this phenomenon relates to each node having different transmission and interference ranges. For example Node 1 may have had a transmission range of only 5 cm, however its interference range may been as much as double that, at 10 cm. Therefore as Node 1 sent a TCP *syn* packet it may have experienced interference from Node 2 and Node 3, i.e. experiencing the hidden and exposed terminal problems. This interference would have had an accumulative effect on re-transmissons of *syn* and *syn-ack* packets for all nodes involved. This would ultimately have caused Node 1 to timeout before the complete three way TCP handshake could have been completed, thus impeding any transfer of data. Medium access protocols such as RTS/CTS [44] could possibly have alleviated this problem, however such protocols would have caused extra network overhead and as such, were not used during the evaluation process.
The results obtained for this experiment set are contrary to previously published results of TCP throughput performance in ad hoc networks [45], [46]. This may be due to the fact that these studies provided results obtained from simulated ad hoc conditions. As the results in this work were obtained from experiments which used actual real world ad hoc networks, the results produced can be seen as a more accurate view of actual TCP performance in real world deployed ad hoc networks.

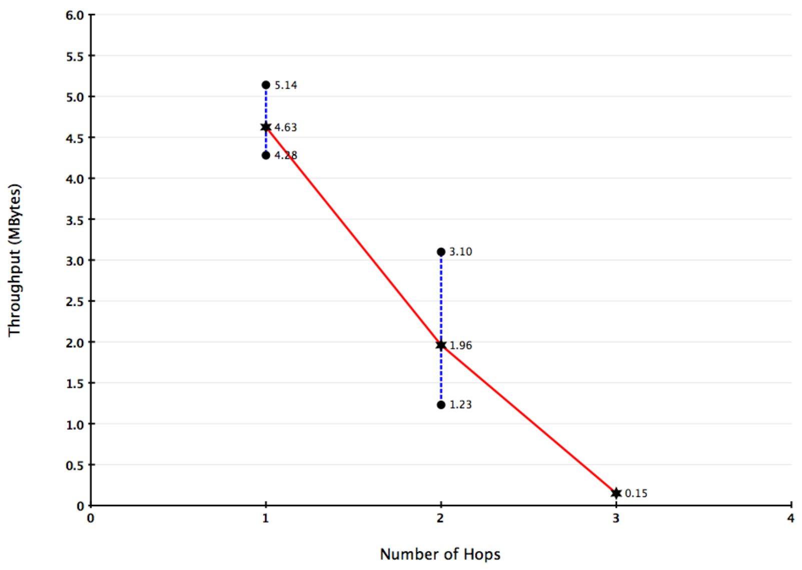


Figure 5.7: TCP throughput for 5 node ad hoc network

Network Overhead

An analysis of traces taken during TCP transmissions in the ad hoc network showed that as the number of transmission hops increased the amount of TCP error packets; comprised of TCP retransmits, TCP lost segments and TCP lost acks, increased dramatically. Figure 5.8 shows that over a one hop TCP transmission the amount of TCP error packets averages 0.91 MBytes, which is small in comparison to the average 4.62 MBytes transmitted. However as the number of transmission hops increase the number of TCP errors is also increased. For example for a three hop TCP transmission TCP error packets average 0.10 MBytes, with only 0.20 MBytes being transmitted 50% of data packets transmitted fall into the TCP error packet category.

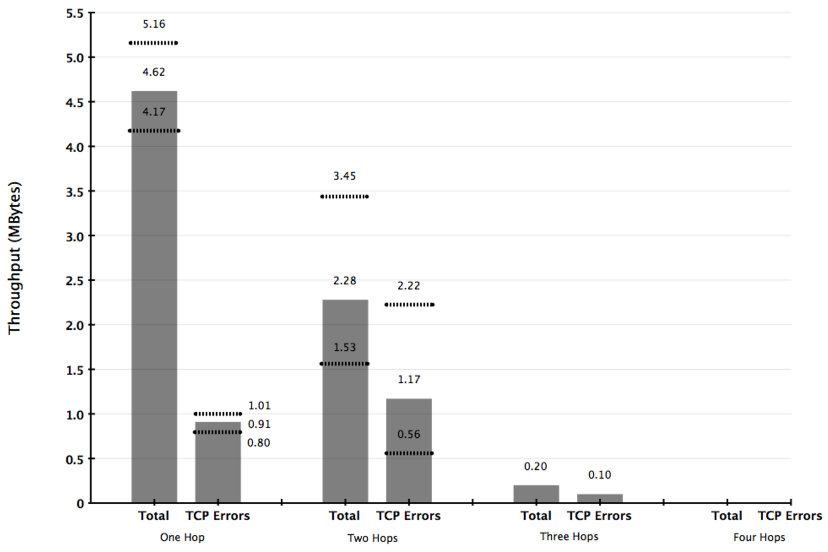


Figure 5.8: TCP network overhead for 5 node ad hoc network

5.2.2 Shoulder Monkey Comparitive Test

As in previous experiments, an ad hoc network with a simple line topology was setup for comparative testing of the Shoulder Monkey system. Channel 3 was used for radio transmissions in this ad hoc network, with a background noise level of -60 dBm. For this experiment set each experiment run consisted of a migrant software agent originating from Node 1, being given the task of routing to Node 5, as seen in Figure 5.11. Five experiment runs were used to obtain results for this experiment set. As no gateways for any node in the network had to be manually set, each node in the ad hoc network had a bi-directional link with any node it was in range of. All experiment runs were carried out for a period of no more than 60 secs.

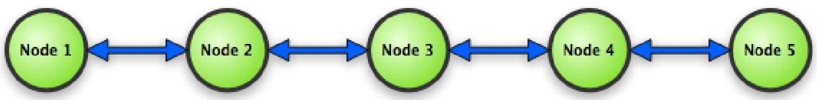


Figure 5.9: Shoulder Monkey line topology for 5 node ad hoc network

Success Rate

For the simple line topology the Shoulder Monkey system was given the task of routing a payload from Node 1 to Node 5, as seen in Figure 5.9, using the AD mechanism. For all experiment runs the Shoulder Monkey system's migrant software agents had a success rate of 100% in delivering data to the given destination node. The only variable in all experiment runs was the time it took for the a migrating software agent to find the destination node, with all destinations being located in a time of no more than 60 seconds, i.e. the length of each TCP transmission for simple TCP routing experiments.
These results showed that the AD mechanism approach to routing in ad hoc networks was susbstantially more reliable than a simple TCP based approach. As each migrant software agent made a jump of no more than one node per transmission, the reliability of both TCP and UDP (the two protocols used to implement the AD mechanism as discussed in chapter 4) at these short ranges allowed for successful operation of the AD mechanism. When routing data via the simple TCP based approach a maximum of 4 nodes could be reached, i.e. Node 1 - Node 2 - Node 3 - Node 4 as seen in Figure 5.6. When routing data via the AD mechanism all five nodes in the ad hoc network could be reached, thus increasing the operational capacity of the ad hoc network.

Network Overhead

The payload for each experiment run was a simple Msg class (as explained in chapter 4), however in systems where the payload would be greater, the size of the payload should be simply added to the size of the migrant software agent, as the migrant software agent would contain the payload to be routed.
An analysis of network traces captured during experiment runs was used to give an insight into network overhead for the AD mechanism. The analysis showed the use of AD mechanism resulted in a minimal network overhead, or mechanism footprint. Figure 5.10 shows that the overall total overhead was in the region of 50.96 KBytes, which was only incurred during times of active data routing as the AD mechanism is a re-active/on-demand

routing mechanism. This showed that the AD mechanism's success created only a small network overhead.

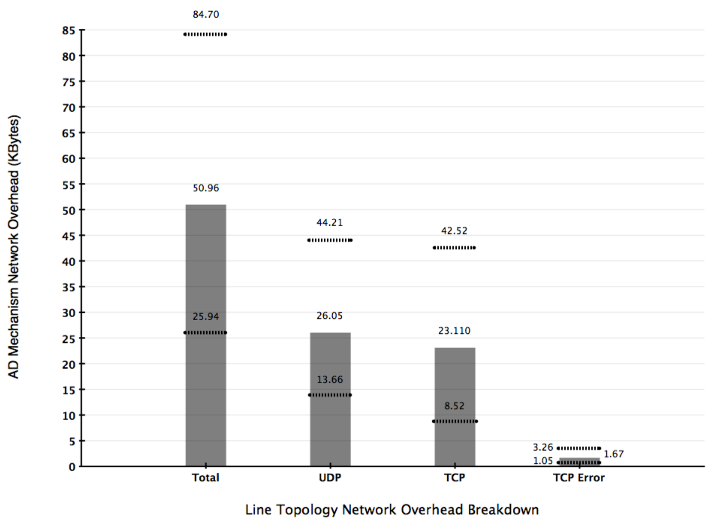


Figure 5.10: Shoulder Monkey network overhead for 5 node line topology

5.3 Shoulder Monkey Stress Test

As the Shoulder Monkey system showed excellent results when compared to a simple TCP data routing approach, it was decided to stress test the application. To measure its performance a number of ad hoc networks with different topologies were setup; a delta topology and a diamond topology. Experiments were then run on these topologies and results obtained and analysed.

5.3.1 Delta Topology

An ad hoc network with a so-called delta topology was setup for the first stress test of the Shoulder Monkey system. Channel 10 was used for radio transmissions in this ad hoc network, with a background noise level of -35 dBm. For this experiment set each experiment run consisted of two migrant software agents; one originating from Node 1 and the other originating from Node 2, being given the task of routing to Node 5, as seen in Figure 5.11. Five experiment runs were used to obtain results for this experiment set. As no gateways for any node in the network had to be manually set, each node in the ad hoc network had a bi-directional link with any node it was in range of. All experiment runs were carried out for a period of no more than 60 secs.

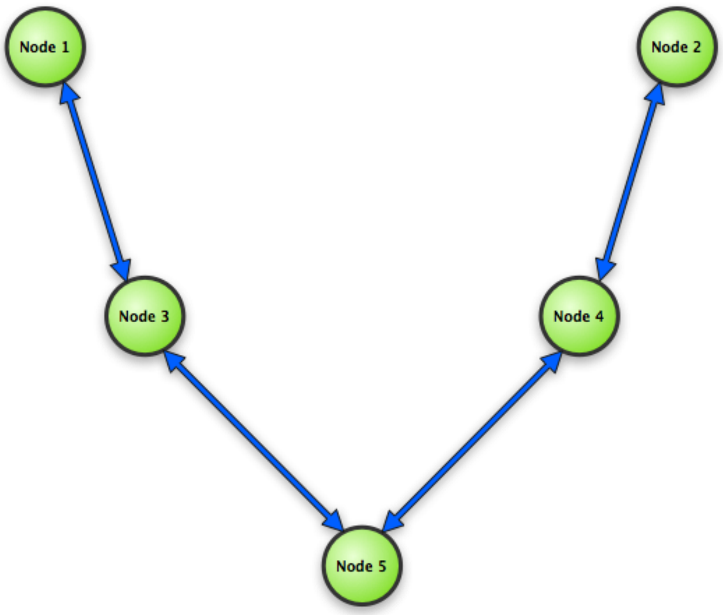


Figure 5.11: Shoulder Monkey delta topology for 5 node ad hoc network

Success Rate/Network Overhead

As with the previous line topology experiment set, the Shoulder Monkey system's migrant software agents had a success rate of 100% in delivering data to the given destination node. An analysis of network traces captured during experiment runs showed that even in a relatively noisy environment such as the one created in the delta ad hoc network, the Shoulder Monkey system performed very well with the AD mechansim creating a mechanism footprint of approximately 50.88 KBytes. Of the network traffic generated, most were UDP control packets used by the Shoulder Monkey system implementation of the AD mechaism.

This experiment set showed that the AD mechanism performed superbly even when multiple migrant software agents were seeking the same destination, and thus competing for medium access, as was the case when the two migrant software agents were operating on Node 3 and Node 4 and had to jump to Node 5, as can be seen in Figure 5.12.

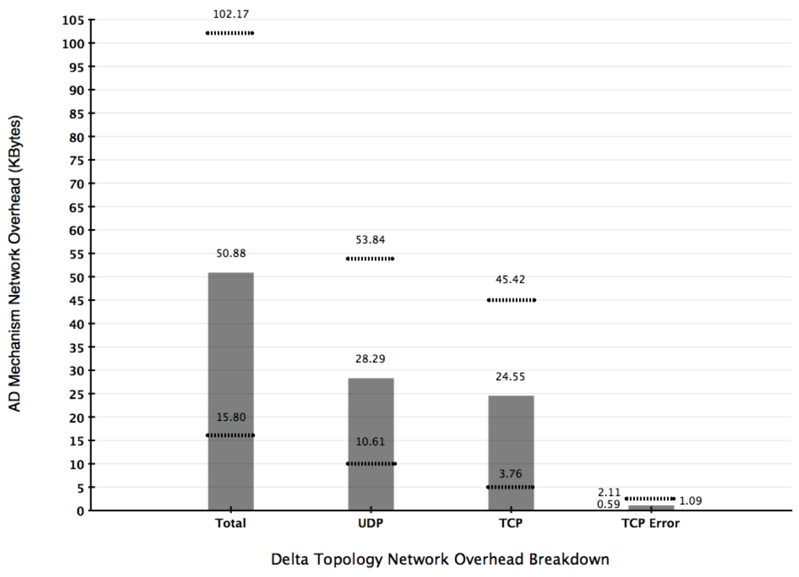


Figure 5.12: Shoulder Monkey network overhead for 5 node delta topology

5.3.2 Diamond Topology

An ad hoc network with a so called diamond topology was set up for the second stress test of the Shoulder Monkey system. Channel 3 was used for radio transmissions in this ad hoc network, with a background noise level of -60 dBm. For this experiment set, each experiment run consisted of a single migrant software agent; originating from Node 1, being given the task of routing to Node 5 via either Node 2 or Node 3, as seen in Figure 5.13. Five experiment runs were used to obtain results for this experiment set. Again as no gateways for any node in the network had to be manually set, each node in the ad hoc network had a bi-directional link with any node it was in range of. All experiment runs were carried out for a period of no more than 60 secs.
For this stress test the Shoulder Monkey system was tested with its migrant agent replication mode or cross roads mode (as discussed in chapter 4), turned on and turned off. A further test, where either Node 3 failed, was also carried out.

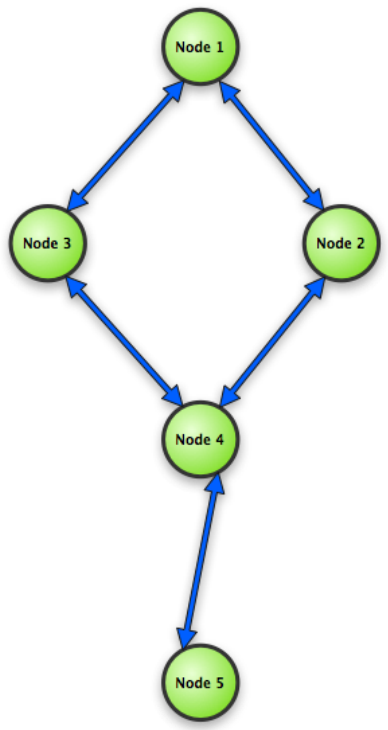


Figure 5.13: Shoulder Monkey diamond topology for 5 node ad hoc network

Success Rate/Network Overhead (Non-Replicating Agents)

As with results obtained from previous experiment sets the Shoulder Monkey system's migrant software agents had a success rate of 100% in delivering data to the given destination node. Figure 5.14 shows the mechanism footprint for the successful routing of software agents through the diamond topology was on average 24.13 KBytes. Again with the majority of the network overhead being created by AD mechanism control packets.

Success Rate/Network Overhead (Replicating Agents)

Results obtained for the diamond topology with replicating migrant software agents also had a success rate of 100% in delivering data to the given destination node. Figure 5.15 shows the mechanism footprint incurred by the use of the AD mechanism. The average mechansim footprint is 45.91 KBytes, less than half of the footprint incurred for the experiment set run on the diamond topology with non-replicating migrant software agents. This can be explained as migrant software agents in the Shoulder Monkey system merge if they end up routing to the same node after being created (as discussed in chapter 4). For the diamond topology on a number of experiment runs, agents that were replicated on Node 1 and sent to Node 2 and Node 3, arrived on Node 4 at the same time and then merged, thus creating a *single* migrant software agent. This single migrant software agent then routed to the destination node creating less network overhead and a smaller mechanism footprint.

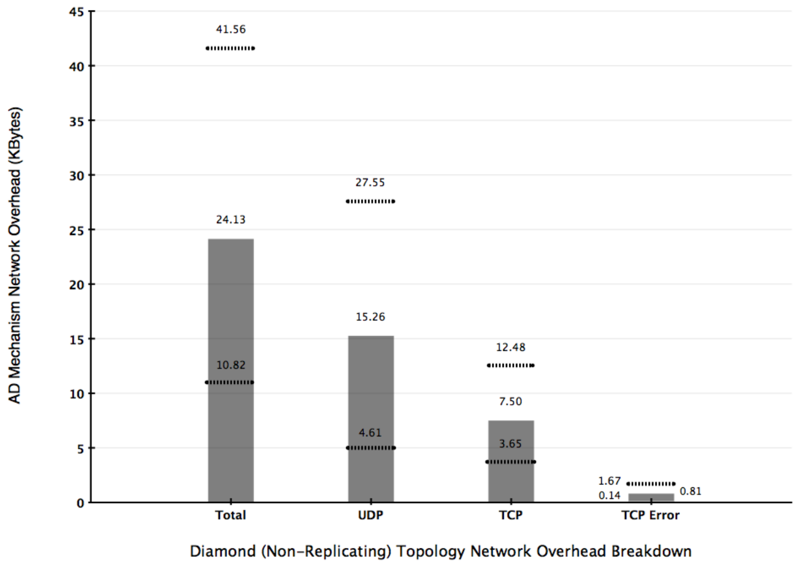


Figure 5.14: Shoulder Monkey network overhead for 5 node diamond topology with non-replication

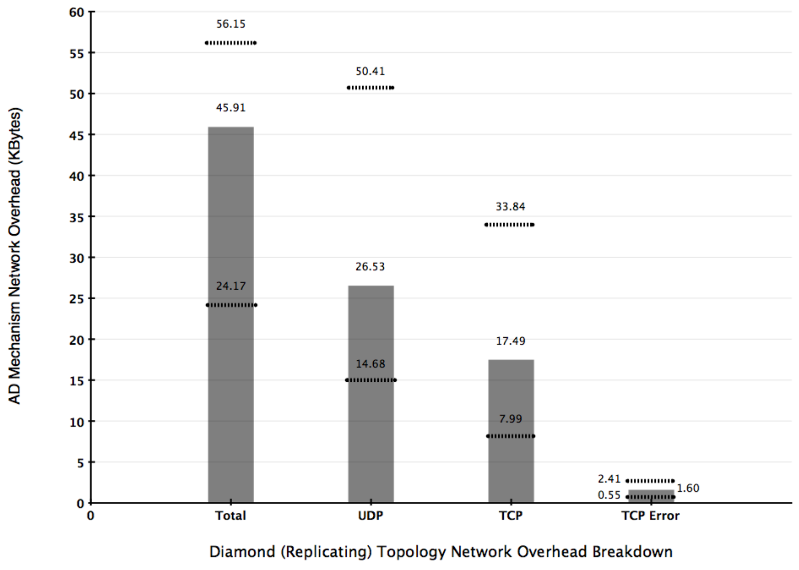


Figure 5.15: Shoulder Monkey network overhead for 5 node line topology with replication

Failed Nodes

The final experiment run on the diamond topology was that of Node 3 failing during network operation. If Node 3 failed, there would still be a route to the destination node, via Node 1 - Node 2 - Node 4 - Node 5, as can be seen in Figure 5.13.
When Node 3 failed before a migrant agent jumped from Node 1, a 100% success rate in delivering data to the given destination node was still upheld for both non-replicating and replicating migrant agents. This was possible as all migrant software agents found the route via Node 2 to the destination node; Node 5.
This sucess rate of 100% was also upheld if Node 3 failed during operation but migrant agents had taken the the route to Node 5 via Node 2.
The success rate however dropped to 50% for non-replicating agents if Node 3 failed at the point when a migrant agent was operating on it. For replicating agents however the 100% success rate was still upheld if Node 3 failed when a migrant agent was operating on it. This was due to another replica agent operating on Node 2 simultaneously, which would be allowed to proceed to Node 5. This showed using replicating agents was a good approach to fault tolerance in ad hoc networks. Migrant agents could traverse an ad hoc network in a swarm like manner replicating when they found mulitple routes to take; for example finding Node 2 and Node 3 in the diamond topology, and merging if they met up again; for example meeting up on Node 4 in the diamond topology. This swarm like migration could act as a built in redundancy system for routing migrant agents in ad hoc networks where network nodes had a tendency to fail.

5.4 Summary

In this chapter, we discussed the evaluation process for Shoulder Monkey system, and by extension, the AD mechanism. An evaluation on successfully deploying a real world ad hoc network was given, to provide readers and possible future researchers an insight into the difficulties of deploying such a test-bed. This was followed by a discussion of the results obtained from quantitive data during the evaluation process. This allowed an accurate analysis of AD mechanism performance to be given to readers.

Chapter 6

Conclusions

Ad hoc networking is an active research area in the field of computer science. By combining this research area with the MAS paradigm, ad hoc networks have potentially substantial practical applications in real-world systems. To achieve this potential, research such as the work carried out for this dissertation; which draws clear and concise conclusions, is required.

This chapter discusses the main conclusions drawn by the author during the course of the work completed for this dissertation.

6.1 Ad Hoc Networks in the Wild

As previously stated, ad hoc networks potentially provide a rapidly deployable, "bring your own network" solution to situations where infrastructure networks cannot be used. In a world where everyday situations rely on communication, the technology which facilitates this communication has to be efficient, reliable and fault tolerant. As ad hoc networks are light weight and easy to deploy, they may possibly become the de facto technology used to enable such activities. That said, ad hoc networks are far from perfect. Many problems in ad hoc networks try to use a wired solution to fix a wireless problem. Similarly managed wireless network solutions are often proposed as solutions to an ad hoc network problem. More often than not these proposed solutions are not transferrable to an ad hoc wireless environment.

Trying to rectify fundamental problems such as efficient data routing in ad hoc networks poses a major problem for ad hoc network performance. This is exacerbated by the fact that testing new ad hoc protocols, mechanisms, etc. is exceptionally difficult. Due to theses difficulties in testing ad hoc network performance, much research has been based on results gathered from simulated ad hoc environments. Work carried out for this dissertation has shown that results from simulated ad hoc networks and results gathered from real-world ad hoc networks differ greatly. Therefore it seems that either more accurate simulators are needed, or the more practical approach of testing ad hoc protocols on real-world deployed networks should be used. Therefore to answer the first research question raised in this work:

Can ad hoc networks be used in real-world, real-time situations?

The author is of the opinion, that yes, ad hoc networks can be use in real-world, real-time situations and it is only through the use of such ad hoc networks "in the wild" that a proper evaluation of the abilities and constraints of ad hoc network protocols can be carried out. As with all new technologies ad hoc networks have some teething problems, however as research in the field enables a deeper analysis of these problems, ad hoc network performance should reach new levels of efficiency, reliability and fault tolerance.

6.2 MAS and Ad Hoc Networks (Together at Last)

The MAS paradigm is primarily used to enhance a system's performance, however this enhancement comes at a price to system resources. As ad hoc networked devices are primarily seen as being particularly resource-constrained, the use of the MAS paradigm may have been viewed as being too resource-intensive for use in ad hoc networks. However, if the MAS paradigm is properly implemented, the positive effects received from using an MAS based approach should greatly outweigh the negative impacts on system resources. Therefore the use of the MAS paradigm can be seen as having a "give a little, get a lot" relationship with ad hoc networks.
As was shown by the work carried out for this dissertation the MAS paradigm does have potential uses in ad hoc networked environments. Therefore to answer the second research question raised in this work:

Is ad hoc network performance enhanced through
the the use of the MAS paradigm?

The author is of the opinion that yes, ad hoc network performance can be greatly enhanced through the use of the MAS paradigm. As with all systems, not paying proper attention to a system's development; during design, implementation, etc, can lead to poor system performance. As ad hoc networked devices may have less resources at their disposal, when using an MAS based approach in a system that is to run in an ad hoc networked environment, extreme vigilance must be taken throughout its development process to ensure the implemented MAS based approach is as efficient as possible.

6.3 The AD mechanism

Successful routing in ad hoc networks is no easy task. Current topology-based routing protocols require that an end-to-end route between source and destination be found before routing of data can take place. Current position-based routing protocols require the exact position; via the use of position devices; such as GPS locators or an external service which provides such information, of a destination to be known before data can be routed to said location. Both of these approaches have their difficulties.
The AD mechanism was developed during the course of this dissertation to answer the third research question raised in this work:

Can the MAS paradigm be used to route data in ad hoc networks?

The author can answer yes to this question, as was proven by the implementation of the agent-based AD mechanism in the Shoulder Monkey system. An evaluation of the AD mechanism showed that two situations in particular would benefit greatly from the use of such a routing mechanism:

1. Rapid routing of small payloads
2. Successful routing of large payloads

As a migrant agent that traverses an ad hoc network carries its payload with it from node to node, the larger the payload the more time it takes for a migrant agent to jump from one node to another. Therefore migrant agents with smaller payloads will be able to traverse an ad hoc network quicker than migrant agents with larger payloads. This enables small payloads to be rapidly routed through ad hoc networks. If agents are allowed to replicate, this routing could be further accelerated as agents would route to a destination in a swarm like manner as discussed in chapter 5 (section 5.5.2 Diamond Topology - Failed Nodes). Migrant agents carrying larger payloads, may take longer to traverse an ad hoc network, however, will successfully do if a path to a given destination exists. Therefore agents carrying larger payloads could act as a delivery system for data that did not need to be delivered to a destination in great haste, so long that it eventually arrived at the given destination. Finally for migrant agents carrying small or large payloads, replication of migrant agents would provide a level fault tolerance for an agent based approach to routing in an ad hoc environment. For example, if any network nodes failed during routing, replicated migrant agents operating on other nodes would still be able to route to a given destination.

6.4 Summary

In this chapter, we discussed the main conclusions drawn from this work. In doing so the author has answered each of the research questions defined in chapter 1 (section 1.1 Motivation). Conclusions for the use of real-world deployed ad hoc networks or ad hoc networks in the wild were first discussed. This was followed by a review of the use of MAS paradigm within the research area of ad hoc networks. Finally a conclusive discussion on the use of the AD mechanism as a technique for data routing in an ad hoc networked environment was given.

Chapter 7

Future Work

A lot of work was carried out in completion of this dissertation, however due to time constraints imposed on the author, all the research avenues considered during the course of this dissertation could not be investigated. Therefore there are a number of interesting ideas relating to this dissertation which have yet to be explored in detail.
This chapter discusses future work that that may be carried out and expand upon the work completed for this dissertation.

7.1 Further Testing

Preliminary testing has shown that the AD mechanism is a successful method for routing data in an ad hoc environment. However further testing of the AD mechanism would give a more in depth view of its abilities and/or constraints.

7.1.1 Larger Testbed

The ad hoc network which was used as the test-bed for experiments conducted was somewhat limited in size. Therefore further testing of the AD mechanism, carried out on a large scale, real world ad hoc network is suggested. Such a test-bed would provide valuable data and allow a full analysis of the AD mechanism to be carried out. The use of a real world deployed ad hoc network is suggested as results gathered from this work conflict with results obtained from previous works [45], [46], where the ad hoc test-bed used for testing was simulated. These simulated results clearly show that transmission of data is achievable for up to 9 hops. In the real world ad hoc network used to gather results for this work, TCP transmission is only achievable for a maximum of 3 hops (as seen in Figure 5.7); without the use of added medium access control protocols such as

RTS/CTS. Therefore to ensure results obtained from experiments in ad hoc networks are accurate, a real world deployment should be used in place of a simulated network.

7.1.2 Introduction of Mobility to Network Nodes

The ad hoc test-bed used to carry out experiments for the AD mechanism was static in nature. The introduction of mobility to the nodes within the test-bed network, thus creating a mobile ad hoc network (MANET), would allow further analysis of the AD mechanism. An analysis of the AD mechanism in such an environment would give a more accurate view of the AD mechanisms performance abilities in real world ad hoc environments. This further analysis would also indicate whether or not the AD mechanism could be successfully deployed and used as a reliable form of routing in ad hoc networks where reliability, fault tolerance and efficiency are paramount.

7.2 Implementation in Lower Layers of the Network Stack

As described in Chapter 3, the AD mechanism was implemented in the Shoulder Monkey system at application level. Even though the system performed extremely well when tested, AD mechanism control packets had to traverse the network stack in order to be processed. By removing this unnecessary stack traversal, the performance of the AD mechanism could be further improved. This may entail implementing the AD mechanism at a lower layer of the network stack, such as the transport or MAC layers. An approach such as this would enable the AD mechanism to be used as the underlying routing protocol in an ad hoc network with a degree autonomy and allow other systems to be easily developed on top of it.

7.3 Dynamic Addition to Agent Behaviour

In the Shoulder Monkey system the implemented behaviour of the migrant software agent was of a monolithic nature, i.e. executed in a sequential fashion. The ability to dynamically add modules of behaviour to a migrant software agent would increase the usability of said agents. In the Shoulder Monkey system the software agents behaviour was contained in a single run method. Allowing dynamic adjustment to this behaviour, by adding and/or removing behaviour modules would enable the Shoulder Monkey system to become more robust. This new feature could ultimately change the Shoulder Monkey system from a

simple message delivery service into an agent based activity platform, where software agents could perform a number of tasks which could be defined by system users.
In the current implementation of migrant agents in the Shoulder Monkey system, agent to agent interaction is kept to a minimum. A further advancement to migrant agents would be to allow them to share information on where they have been and how they have managed to route to their current location. For example if migrant agent X and migrant agent Y encountered each other on a node in an ad hoc network, migrant agent X could query migrant agent Y on whether or it had already visited a destination(s) migrant agent X was seeking, and vice versa. The agents could then exchange data on the best course of action to route to this already visited destination. This behaviour could also be added as an additional behaviour module to existing migrant agents.

7.4 Final Thought

The work carried out for this dissertation has provided a viable working solution for routing data in an ad hoc networked environment. This was achieved by using the MAS paradgim to help overcome problems with efficient data routing in ad hoc networks. The solution proposed acts as a means to minimise the limiting factors of ad hoc networks in real world situations. To achieve this, a new mechansim for distributing agents throughout an ad hoc network and the data which they carry, was developed. This mechanism was entilted the AD mechanism. The AD meachanism was then implemented in a simple message delivery system for use in ad hoc networks called Shoulder Monkey. An evaluation of this system showed that the AD mechanism was, indeed, a practical and efficient solution to the problem of routing data in ad hoc networks. Future work was then proposed to allow researchers to build upon the work produced for the the completion of this dissertation, which could further enhance the usability of ad hoc networks in the wild.

Appendix A

Abbreviations

Short Term	**Expanded Term**
MAS	Multi Agent System
MANET	Mobile Ad Hoc Network
AD Mechanism	Agent Distribution Mechanism
WAND	Wireless Ad Hoc Network for Dublin
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Appendix B

Technical Specifications

[ts1] Official ns-2 wiki, http://nsnam.isi.edu/nsnam/index.php/, 14:08GMT, 19 Oct, 2008.

[ts2] Official opnet website, http://www.opnet.com/, 14:10GMT, 19 Oct, 2008.

[ts3] Official omnett++ website, http://www.omnetpp.org/, 14:15GMT, 19 Oct, 2008.

[ts4] Official glomosim website, http://pcl.cs.ucla.edu/pro jects/glomosim/, 14:18GMT, 19 Oct, 2008.

[ts5] Official qualnet website, http://www.scalable-networks.com/, 14:21GMT, 19 Oct, 2008.

[ts6] Dell d400 specication, http://www.dell.com/downloads/us/products/latit/d400 spec.pdf, 12:30GMT, 4 Sep, 2008.

[ts7] Dell c400 specication, http://www.tkoelectronics.com/tkoeducation/tkostore/c400, 12:29GMT, 4 Sep, 2008.

[ts8] Ubuntu 8.04 (hardy heron), http://www.ubuntu.com/testing/hardy/beta, 12:32GMT, 4 Sep, 2008.

[ts9] Netgear wpn111 wireless usb adaptor specification, http://www.netgear.com/products/adapters/rangemaxadapters/wpn111.aspx, 12:34GMT, 4 Sep, 2008.

[ts10] Java 6.0 jre, http://java.sun.com/javase/downloads/ea.jsp, 12:36GMT, 4 Sep, 2008.

[ts11] Apple ibook g4 specification, http://support.apple.com/specs/ibook/ibook g4.html, 12:39GMT, 4 Sep, 2008.

[ts12] Official wireshark website, http://www.wireshark.org/, 12:41GMT, 4 Sep, 2008.

[ts13] Wpn111 installation tutorial, http://ubuntuforums.org/showthread.php, 12:46GMT, 4 Sep, 2008.

[ts14] iperf tool, http://sourceforge.net/pro jects/iperf, 12:48GMT, 4 Sep, 2008.

Bibliography

[1] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90–100, 25-26 Feb 1999.

[2] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM.

[3] H.F. Ahmad. Multi-agent systems: overview of a new paradigm for distributed systems. *High Assurance Systems Engineering, 2002. Proceedings. 7th IEEE International Symposium on*, pages 101–107, 2002.

[4] Oxford English Dictionary. Definition of ad hoc, http://dictionary.oed.com/, 2001.

[5] M. Wooldridge. *An Introduction to Multiagent Systems*, pages 227–234. Wiley, 2002.

[6] M. Mauve, A. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *Network, IEEE*, 15(6):30–39, Nov/Dec 2001.

[7] C. Siva Ram Murthy and B.S. Manoj. *Ad Hoc Wireless Networks: Architectures and Protocols*, pages 229–278. Prentice Hall, 24 May 2004.

[8] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[9] Hsing-Lung Chen and Chein-Hsin Lee. Two hops backup routing protocol in mobile ad hoc networks. *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on*, 2:600–604 Vol. 2, 20-22 July 2005.

[10] Jiwei Chen, Yeng-Zhong Lee, He Zhou, Mario Gerla, and Yantai Shu. Robust ad hoc routing for lossy wireless environment. *Military Communications Conference, 2006. MILCOM 2006*, pages 1–7, 23-25 Oct. 2006.

[11] Guangyu Pei, Mario Gerla, and Tsu-Wei Chen. Fisheye state routing in mobile ad hoc networks. In *ICDCS Workshop on Wireless Networks and Mobile Computing*, pages D71–D78, 2000.

[12] Christian Lochert, Martin Mauve, Holger Fussler, and Hannes Hartenstein. Geographic routing in city scenarios. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(1):69–72, 2005.

[13] Y.-C. Hu, A. Perrig, and D.B. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 3:1976–1986 vol.3, March-3 April 2003.

[14] M. Ramkumar and N. Memon. An efficient key predistribution scheme for ad hoc network security. *Selected Areas in Communications, IEEE Journal on*, 23(3):611–621, March 2005.

[15] Hao Yang, Haiyun Luo, Fan Ye, Songwu Lu, and Lixia Zhang. Security in mobile ad hoc networks: challenges and solutions. *Wireless Communications, IEEE*, 11(1):38–47, Feb 2004.

[16] C. Manikopoulos and Li Ling. Architecture of the mobile ad-hoc network security (mans) system. *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, 4:3122–3127 vol.4, Oct. 2003.

[17] Alejandro Guerra Hernandez, Amal El Fallah-Seghrouchni, Henry Soldano, and Henry Soldano. Distributed learning in intentional bdi multi-agent systems. In *ENC '04: Proceedings of the Fifth Mexican International Conference in Computer Science*, pages 225–232, Washington, DC, USA, 2004. IEEE Computer Society.

[18] N. Ronald and L. Sterling. Modelling pedestrian behaviour using the bdi architecture. *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 161–164, 19-22 Sept. 2005.

[19] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *CIKM '94: Proceedings of the third international conference on Information and knowledge management*, pages 456–463, New York, NY, USA, 1994. ACM.

[20] FIPA. Communicative act library specification, http://www.fipa.org/specs/fipa00037/xc00037h.html, 2001.

[21] Jeffrey S. Rosenschein and Michael R. Genesereth. *Deals among rational agents*, pages 227–234. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[22] Barbara [J.] Grosz and Sarit Kraus. Collaborative plans for group activities. In *Proceedings IJCAI-93*, pages 367–373, 1993.

[23] O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Agent cloning: an approach to agent mobility and resource allocation. *Communications Magazine, IEEE*, 36(7):58, 63–67, Jul 1998.

[24] Hiroki Suguri. A standardization effort for agent technologies: The foundation for intelligent physical agents and its activities. In *HICSS '99: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8*, page 8061, Washington, DC, USA, 1999. IEEE Computer Society.

[25] P. Charlton, E. Mamdani, R. Cattoni, and A. Potrich. Evaluating the fipa standards and its role in achieving cooperation in multi-agent systems. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8034, Washington, DC, USA, 2000. IEEE Computer Society.

[26] FIPA. Specification 1998, http://www.fipa.org/specs/index.html, 7 May 2008, 15:54 GMT.

[27] FIPA. Specification 2000, http://www.fipa.org/specs/index.html, 7 May 2008, 15:56 GMT.

[28] V. Cicirello, M. Peysakhov, G. Anderson, Gaurav Naik, K. Tsang, W. Regli, and M. Kam. Designing dependable agent systems for mobile wireless networks. *Intelligent Systems, IEEE*, 19(5):39–45, Sept.-Oct. 2004.

[29] J. Kopena, E. Sultanik, Gaurav Naik, I. Howley, M. Peysakhov, V.A. Cicirello, M. Kam, and W. Regli. Service-based computing on manets: enabling dynamic interoperability of first responders. *Intelligent Systems, IEEE*, 20(5):17–25, Sept.-Oct. 2005.

[30] J.P. Macker, W. Chao, R. Mittu, and M. Abramson. Multi-agent systems in mobile ad hoc networks. *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 883–889 Vol. 2, 17-20 Oct. 2005.

[31] J. Dowling, E. Curran, R. Cunningham, and V. Cahill. Using feedback in collaborative reinforcement learning to adaptively optimize manet routing. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 35(3):360–372, May 2005.

[32] Azzedine Boukerche and Yonglin Ren. A novel solution based on mobile agent for anonymity in wireless and mobile ad hoc networks. In *Q2SWinet '07: Proceedings of the 3rd ACM workshop on QoS and security for wireless and mobile networks*, pages 86–94, New York, NY, USA, 2007. ACM.

[33] Jan M. V. Misker, Cor J. Veenman, and Leon J. M. Rothkrantz. Groups of collaborating users and agents in ambient intelligent environments. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1320–1321, Washington, DC, USA, 2004. IEEE Computer Society.

[34] Official fipa website, http://www.fipa.org/, 14:02GMT, 19 Oct, 2008.

[35] V.A. Cicirello, A. Mroczkowski, and W. Regli. Designing decentralized software for a wireless network environment: evaluating patterns of mobility for a mobile agent swarm. *Multi-Agent Security and Survivability, 2005 IEEE 2nd Symposium on*, pages 49–57, 30-31 Aug. 2005.

[36] Jade web site, http://jade.tilab.com, 7 May 2008, 15:52 GMT.

[37] S. Daviet, H. Desmier, H. Briand, F. Guillet, and V. Philippe. A system of emotional agents for decision-support. *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 711–717, 19-22 Sept. 2005.

[38] Aos website (jack info page), http://www.agent-software.com/products/jack/index.html/, 14:45GMT, 19 Oct, 2008.

[39] R.W. Collier, M.J. O'Grady, G.M.P. O'Hare, C. Muldoon, D. Phelan, R. Strahan, and Y. Tong. Self-organisation in agent-based mobile computing. *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on*, pages 764–768, 30 Aug.-3 Sept. 2004.

[40] Prism lab website (agent factory info page), http://www.cs.ucd.ie/csprism/projects.html/, 14:48GMT, 19 Oct, 2008.

[41] R.W. Collier, M.J. O'Grady, G.M.P. O'Hare, C. Muldoon, D. Phelan, R. Strahan, and Y. Tong. Self-organisation in agent-based mobile computing. *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on*, pages 764–768, Aug.-3 Sept. 2004.

[42] J. N. Thies R. Lentini, G. P. Rao and J. Kay. Emaa: An extendable mobile agent architecture. In *In AAAI Workshop on Software Tools for Developing Agents*, July 1998.

[43] Skype web site, http://www.skype.com/intl/en/, 4 Sep 2008, 12:20 GMT.

[44] A. Tanenbaum. *Computer Networks (fourth edition)*, pages 295–299. Prentice Hall, 2003.

[45] Yao-Nan Lien and Yi-Fan Yu. Hop-by-hop tcp over manet. In *The First IEEE International Workshop on Wireless Network Algorithms (WiNA 2008)*, 2008.

[46] Gavin Holland and Nitin Vaidya. Analysis of tcp performance over mobile ad hoc networks. *Wirel. Netw.*, 8(2/3):275–288, 2002.