Towards flexible and scalable distributed monitoring with mobile agents

Towards Flexible and Scalable Distributed Monitoring with Mobile Agents

Antonio Liotta

A dissertation submitted in partial fulfilment of
the requirements for the degree of

Doctor of Philosophy

of the

University of London

Department of Computer Science

University College London

July 2001

Abstract

The tremendous success of the Internet has made possible and even encouraged the realisation of systems characterised by very large scale and high level of distribution. Managing such systems requires systematic communication between a centralised station and distributed components. In many cases, a pure ‘centralised’ model is adopted in which the management station retrieves information directly from the managed elements and performs all the required computation. For large, distributed systems this model is prone to information implosion, which tends to cause congestion both to the management station and to its attached network. Therefore, pure centralised management, despite having the advantage of simplicity, inherits the intrinsic limitations of centralised systems, namely their limited responsiveness, accuracy, and scalability.

The approach traditionally followed to address those limitations is to decentralise management intelligence. A natural approach is to introduce several ‘area’ managers in charge of collecting and pre-processing raw data from different portions of the system. When possible, the event-driven model of computation is adopted. In this case, the area managers or even the managed elements are equipped with logic that performs semantic compression of local information and sends notifications to the manager only under particular circumstances. In this way, the central station is alleviated and the traffic incurred in its vicinity can be dramatically reduced. Event-driven and hierarchical management system organization can cope with scalability problems only to a limited extent. Such approaches are inherently more complex that centralised ones, while they still lack the flexibility, adaptability and relatively loose system organization, which are desirable in large-scale, highly dynamic networked systems.

This thesis is focused on the design and evaluation of a *dynamic distributed monitoring system* based on the use of mobile software agents. Agents act in the role of area managers but, differently from the case of *static distributed monitoring systems*, they can be placed in strategic locations within the network by accounting for the network state and for the type of task to be performed. In addition, at run time, agents can *migrate* to other locations or *clone* other agents in order to provide adaptation to changes in the underlying network.

The core of the thesis addresses the problem of efficiently computing the agent locations. In graph theoretical terms, this is the problem of optimally placing *p* servers within a network of *N* nodes, which falls in the class of the *p-centre* and *p-median* problems. These are NP-complete problems when striving for optimality. The thesis proves that existing approximate solutions, computable in polynomial time, are not viable. Consequently, a novel approximate solution, aiming at minimising the overall traffic and delay incurred by the agent-based distributed monitoring system, is proposed. The proposed algorithm is proved O(*N***R(u)*) (where *R(u)* is the network radius and *N* the number of monitored nodes). Moreover, it is demonstrated that the computed agent locations are near-optimal.

The agent location algorithm is solved in a distributed fashion making use of *agent weak mobility*, which is the ability of agents to move around the network from node to node carrying their code and data. The algorithm relies also on *agent cloning*, the ability of an agent to create and dispatch copies of itself. Both weak mobility and agent cloning are properties that so far have not been exploited to the full extent of their potential in the field of management. A distributed monitoring system based on this algorithm is assessed by simulation and it is shown that significant reductions in both network traffic and response time can be achieved in addition to the increased flexibility and adaptability offered by agent mobility.

To Maria, obviously!

Acknowledgements

I would like to thank my two supervisors, Graham Knight and George Pavlou, for their invaluable support, encouragement, constructive criticism, advice, and friendship throughout these years. Very useful directions have been provided by Jon Crowcroft and Stevie Hailes, who have examined my work on the way. Many thanks to other members of the Computer Science Department of UCL, in particular David Griffin, Saleem Bhatti, Jose' Borges, Jorge Ortega-Arjona, Tom Quick, Rafael Bordini, Nadav Zin, and Adil Qureshi.

Special thanks go to Alberto Ferreira de Souza, a colleague, a friend, and a constant point of reference. Without the long, broad-spectrum discussions I had with him, my PhD period wouldn't have been so enriching and stimulating.

I kindly acknowledge Chris Bohoris from the University of Surrey for providing measurements of Mobile Agents overheads in the context of the Grasshopper Mobile Agent platform. Figure 3-1, Figure 3-2, and Figure 3-3 are based on those measurements.

The PhD adventure started when Lorenzo Coslovi, from Hewlett-Packard, believed in my capabilities and decided to make a case to his company to fund my PhD. In a way, his decision to 'bet' on me may have changed the course of my life. He introduced me to the right people at Hewlett-Packard Labs Bristol and followed me through the difficulties of the initial period. I am indebted with my Industrial supervisors, Keith Harrison and Jon Manley, for initiating me to the avenues of the research world. Finally, I kindly acknowledge Hewlett-Packard for its generous sponsorship.

Invaluable has been the support I have received by my friends, in particular Nickie Coleman, Jon Wells, Pilar Sepulveda, and Cynthia Shaw (all from Imperial College). I am in dept with Claudio Catania and Antonio Barili for encouraging me to step beyond the border of the Italian University. Their constant friendship was a real support during this transition. Clearly my partner and my family have played the key role of keeping me alive by providing first-class oxygen.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

API	Application Programming Interface
CE	Computational Environment
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
COD	Code on Demand
CORBA	Common Object Request Broker Architecture
CS	Client-Server
DAI	Distributed Artificial Intelligence
DISMAN	The IETF Distributed Management working group
DMTF	Desktop Management Task Force
EU	Executing Unit
FIPA	Foundation for Intelligent Physical Agents
HTTP	Hypertest Transfer Protocol
IA	Intelligent Agent
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IN	Intelligent Network
IP	Internet Protocol
ISO	International Organisation for Standardisation
ITU	International Telecommunication Union

ITU-T	ITU Telecommunication Standardisation Sector
J-DMK	Java Dynamic Management Kit
JIDM	Joint Inter-Domain Management
JMAPI	Java Management API Architecture
LAN	Local Area Network
MA	Mobile Agent
MAS	Multi-Agent System
MbD	Management by Delegation
MCS	Mobile Code System
MIB	Management Information Base
MO	Managed Object
N&SM	Network and System Management
ODMA	Open Distributed Management Architecture
OMG	Object Management Group
OSI	Open Systems Interconnection
REV	Remote Evaluation
RPC	Remote Procedure Call
RMI	Remote Method Invocation
RM-ODP	Reference Model of the ISO Open Distributed Processing
RMON	Remote Monitoring (an MIB)
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TINA-C	Telecommunications Information Networking Architecture Consortium
TMN	Telecommunications Management Network
WBEM	Web-based Enterprise Management

Thesis Related Publications

Journal Papers

[Liotta 01b] **A. Liotta**, G. Pavlou, G. Knight, *Reducing the Cost of Large-Scale Network Monitoring with Mobile Code*, submitted to IEEE Network.

[Pavlou 98b] G. Pavlou, **A. Liotta**, P. Abbi, S. Ceri, *CMIS/P++: Extensions to CMIS/P for Increased Expressiveness and Efficiency in the Manipulation of Management Information*, IEEE Network, Special Issue on Network Management - Today and Tomorrow, Vol. 12, No. 5, pp.10-20, IEEE, (September/October 1998).

Conference Papers

[Liotta 01c] **A. Liotta**, G. Pavlou, G. Knight, *A Self-adaptable Agent System for Efficient Information Gathering*, Proceedings of the 3rd International Workshop on Mobile Agents for Telecommunication Applications (MATA'01), Montreal, Canada, Springer-Verlag (August 2001).

[Liotta 01a] **A. Liotta**, G. Pavlou, G. Knight, *Active Distributed Monitoring for Dynamic Large-scale Networks*, Proceedings of the IEEE International Conference on Communications (ICC'01), Helsinki, Finland, IEEE, (June 2001).

[Bohoris 00c] C. Bohoris, **A. Liotta**, G. Pavlou, *Evaluation of Constrained Mobility for Programmability in Network Management*, To appear in the proceedings of the 11th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2000), Austin, Texas, USA, (December 2000).

[Bohoris 00b] C. Bohoris, **A. Liotta**, G. Pavlou, *Software Agent Constrained Mobility for Network Performance Monitoring*, Proc. of the 6th IFIP Conference on Intelligence in Networks (SmartNet 2000), Vienna, Austria, ed. H.R. van As, pp. 367-387, Kluwer, (September 2000).

[Pavlou 00a] G. Pavlou, **A. Liotta**, C. Bohoris, D. Griffin, P. Georgatsos, *Providing Customisable Remote Management Sevices Using Mobile Agents*. In proc. of HP-OVUA, The Hewlett-Packard Openview University Association Plenary Workshop 2000, Santorini, Greece, (June 12-14, 2000).

[Liotta 99c] **A. Liotta**, G. Knight, G. Pavlou, *A Simulation-based Assessment of Information Gathering Systems based on Mobile Agents*. In proc. of Simulation’99, London, UK, (October 1999).

[Liotta 99b] **A. Liotta**, G. Knight, G. Pavlou, *On the Performance and Scalability of Decentralised Monitoring Using Mobile Agents*, Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations Management (DSOM'99), (October 1999).

[Liotta 99a] **A. Liotta**, G. Knight, G. Pavlou, *On the Efficiency of Decentralised Monitoring using Mobile Agents.* In proc. of HP-OVUA, The Hewlett-Packard Openview University Association Plenary Workshop 1999, Bologna, Italy, (June 1999).

[Liotta 98b] **A. Liotta**, G. Knight, *Decomposition Patterns for Mobile Code-based Management*. In proc. of HP-OVUA, The Hewlett-Packard Openview University Association Plenary Workshop 1998, ENST de Bretagne, Rennes, France, (April 1998).

[Liotta 98a] **A. Liotta**, G. Knight, G. Pavlou, *Modelling Network and System Monitoring Over the Internet Using Mobile Agents*, Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS '98), New Orleans, USA, Vol. 2, pp. 300-312, (February1998).

[Pavlou 98a] G. Pavlou, **A. Liotta**, P. Abbi, S. Ceri, *CMIS/P++: Extensions to CMIS/P for Increased Expressiveness and Efficiency in the Manipulation of Management Information.* In proc. of IEEE Infocom'98, Vol. 2, pp. 430-438, IEEE, San Francisco, USA, (29 March - 2 April 1998).

PART I

THESIS BACKGROUND

Chapter 1

Introduction

1.1 Thesis Overview

The tremendous success of the Internet has made possible and even encouraged the realisation of systems characterised by very large scale, and high level of distribution and dynamics. ‘Network-centric’ approaches such as Sun’s Jini architecture [Waldo 99] envisage large numbers of comparatively simple devices (cellular phones, televisions, thermostats) all accessible across the net. Management systems in the future will need to keep track of these devices and determine which are present, which are functioning correctly and so on. Scalability and high levels of dynamics are also key requirements of the third generation mobile networks or 3G.

More generally, the ability to monitor a network or distributed system accurately and effectively is of paramount importance for its operation, maintenance, and control. Network monitoring, for instance, entails the collection of traffic information used for a variety of performance management activities *e.g.* capacity planning and traffic flow predictions, bottleneck and congestion identification, quality of service monitoring for services based on service level agreements, etc. A key aspect is that collection of traffic information should be supported in a timely manner, so that reaction to performance problems is possible, and without incurring too much additional traffic on the managed network.

Given these motivations and constraints, efficient network or system monitoring is an interesting research problem. The conventional approach is to poll managed devices from a *centralised* management station where most of the intelligence is concentrated. An event-driven approach, in which managed devices notify the management station, is also possible but requires complex functionality to be built into monitored elements. This increases their complexity and cost and, more importantly, it is fixed and cannot be customized and augmented as requirements change. As such, the polling model is being widely used because of its simplicity and flexibility. An example is represented by the Simple Network Management (SNMP) protocol which promotes polling of simple network elements [Stallings 93].

For large distributed systems, this model is prone to information implosion, which tends to cause congestion both to the management station and to its attached network. Therefore, pure centralised management, despite having the advantage of simplicity, inherits the intrinsic limitations of centralised systems, namely their limited responsiveness, accuracy, and scalability.

The approach traditionally followed to address those limitations is to decentralise management intelligence by dividing the system into smaller "areas" and deploying one polling-based monitoring station per area. We term this the *static decentralised* approach as the locations of the 'area' monitoring stations are typically computed off-line and do not change after deployment.

Area managers are in charge of collecting and pre-processing raw data from different portions of the system. When possible, the event-driven model of computation is adopted. In this case, the area managers or even the managed elements are equipped with logic that performs semantic compression of local information and sends notifications to the manager only under particular circumstances. In this way, the central station is alleviated and the traffic incurred in its vicinity can be dramatically reduced. The static decentralised management system organization can cope with scalability problems only to a limited extent. Such approaches are inherently more complex that centralised ones, while they still lack the flexibility, adaptability and relatively loose system organization, which are desirable in systems relying on large, dynamic networks.

The ideal solution is dynamic or *active distributed* monitoring, with stations computing their optimal location based on the target monitored objects. A station will move to that location and will possibly adapt to network changes and move again when conditions change in order to maintain optimality. Given the required mobility, proactivity and reactivity properties, a monitoring station could be realised through a mobile software agent.

A key issue in the static decentralised approach is the calculation of the optimal locations for the monitoring stations. Optimality in this case concerns the minimisation of network traffic incurred due to (localised) polling and the minimisation of the required latency in collecting the necessary information. The problem regarding the optimal placement of a number of servers in a large network has been studied in the literature. It is equivalent to the *p-median* and *p-centre* location problems studied in the context of graph theory and location theory, as discussed in Chapters 2 and 3. These problems are both *NP-complete* when striving for optimality. Approximate algorithms exist but they are characterised by polynomial complexity of high degree. An additional problem is that these algorithms are centralised, requiring the network distance matrix at the monitoring station.

While this is less of a problem in off-line calculations for medium to long-term optimal locations, it becomes an important problem for active distributed solutions in which network conditions may vary rapidly. In this case, existing location algorithms are not suitable to solve the problem of placing the area managers near-optimally. Therefore, given our proposal to use mobile software agents as area monitoring stations, a new distributed algorithm is required. This thesis proposes such an algorithm that relies on agents learning about the network topology through node routing table information which is accessed through standard management interfaces. The monitoring system is deployed through a "clone and send" process starting at the centralised network-wide station. The same algorithm adopted for the initial agent deployment is also used for agents to adapt to network changes through migration. Key features of this algorithm are its distributed nature, *i.e.* each agent carries and runs the algorithm, and its low computational complexity.

The proposed algorithm is proved O(*N***R(u)*), whereby *N* is the number of network nodes, *u* is the location of the monitoring station, and *R(u)* is the network radius. The agent location algorithm is solved in a distributed fashion making use of *agent weak mobility* – the ability of agents to move around the network from node to node carrying code and data. The algorithm relies also on *agent cloning* – the ability of an agent to create and dispatch copies of itself. Both weak mobility and agent cloning are properties that, despite their potential, have previously been exploited only marginally in the particular field of management. A distributed monitoring system based on this algorithm is assessed by simulation and it is shown that significant reductions in both network traffic and response time can be achieved.

1.2 Research Motivation

The original motivation which sparked the thesis was to investigate means to manage large-scale, dynamic networks more efficiently than was offered by more conventional centralised or hierarchical management systems. Code mobility was seen as the candidate enabler. The Management by Delegation (MbD) work carried out in the Columbia University since the early nineties had already shown the potentiality of the 'push' model, a mechanism to dynamically enhance the capabilities of remote servers and delegate simple management tasks [Yemini 91, Goldszmidt 95a, Goldszmidt 96b]. After the success of the Java language [Java 95] and the proliferation of Java-based Mobile Agent (MA) frameworks, the MbD idea was followed by intensive research aimed at establishing the use of MAs into the network management arena.

One of the reasons why so many researchers started looking at MA-based management was that MAs were already being successfully applied to domains as diverse as autonomous vehicles, industrial process control, e-commerce, networking etc. Many advantages are commonly associated with MAs; particularly relevant to this thesis is their ability to result in bandwidth savings and reduced latency. Limited bandwidth and excessive latency are, in fact, the major barriers of large-scale management systems.

Soon it became clear that, while most effort was directed towards the attempt to use MAs to find improved solutions to specific network management problems, the real essence of mobile software agents was not being fully exploited. In fact, the situation in which agents may move around the network in a reactive or proactive adaptive manner, carry their code and data, and clone/destroy themselves according to their intelligence has not yet been shown to achieve better results than static approaches in network management.

Most of the applications found in the literature tend to adopt simpler degenerate forms of mobility, such as *Remote Evaluation*, *Code on Demand*, *constrained mobility*, or the so called *push* model. These models of mobility and their application to management systems will be discussed in the next two chapters. It will be shown that those forms of mobility provide the vehicle for programmability through the enhancement of pre-existing functionality. This represents a first step ahead of static distributed management. A second step towards dynamic, distributed management is represented by the introduction of weak mobility, which (in the context of management) has not undergone in-depth study in recent years and whose potentiality has not been fully measured.

Further motivation to the work is provided in Chapters 2 and 3 which highlight the issues and literature gaps addressed herein.

1.3 Thesis Objective

The objective of this thesis is to fill some of the gaps mentioned in the previous section, beyond the boundaries of network monitoring and in the more general context of monitoring of large-scale, dynamic networked systems. Therefore, the main objectives are:

1. the investigation of agent mobility in the context of distributed monitoring;

2. the proposal of a scalable, adaptive, and active distributed monitoring approach based on the MA paradigm;
3. a comparative, quantitative evaluation based on performance, scalability, and flexibility between the proposed agent-based monitoring and the more conventional ‘centralised’ or ‘static distributed’ monitoring;
4. The assessment of the hypothesis stated in the next section.

1.4 The Hypothesis

This thesis examines the following hypothesis:

> *The application of the ‘weak agent mobility’ paradigm to distributed monitoring represents an effective complement to more conventional ‘centralised’ and ‘static distributed’ monitoring approaches. In particular, the agent approach:*
>
> 1. *can lead to significant improvements in performance and scalability;*
> 2. *can be used to realise near-optimal distributed monitoring systems;*
> 3. *can be used to realise distributed monitoring systems which can adapt effectively to changes in the network state;*

1.5 Research Methodology

The hypothesis is evaluated partly by mathematical modelling and partly through simulation. The former is adopted to study the more theoretical aspects of the work, such as:

- the asymptotic complexity of the proposed agent deployment algorithm, *i.e.* its scalability;
- the typical order of magnitude of agent deployment overheads, *i.e.* deployment time and traffic;
- the sufficient conditions on network topology for which the proposed agent location algorithm places the agents near-optimally in the network;
- the study of the agent system under those near-optimal conditions and its comparison with the static monitoring approach;

Simulations are adopted to study the agent-based monitoring system under general conditions in the case of realistic internetworks, covering aspects such as:

- the quantitative comparative evaluation of performance and scalability between static monitoring and active, distributed monitoring;
- the assessment of the goodness of the agent locations computed by the proposed algorithm, *i.e.*, its distance from optimality;
- the preliminary assessment of the ability of the agent system to adapt to network changes;

1.6 Overview of the Contributions

This thesis makes the following main contributions:

- the proposal and assessment of a novel approach to distributed monitoring, termed *active distributed monitoring*, which is based on autonomous mobile software agents;
- the comparative quantitative evaluation of the proposed agent-based monitoring approach against the more conventional 'centralised' or 'static distributed' monitoring;
- an initial study of the ability of such agent system to adapt to network changes to maintain location near-optimality;
- a mathematical-based study of the *p-median* location problem (an NP-complete problem when striving for optimality) and the formulation and evaluation of a novel near-optimal solution to it, which is computed in polynomial time and is also viable for the proposed agent system;
- extensions to the UCB/LBNL/VINT NS network simulator in order to support mobile code capabilities [NS]. The extended simulator can be used to study agent-based systems by simulation.

1.7 What the Thesis is not About

The are a number of other open issues that are strongly related to the thesis and worth investigation but were either left out of the scope of the thesis or treated more marginally. Some of them are listed below:

- security and safety issues introduced by code mobility;
- search for the killer application of MAs to distributed monitoring;
- broad application of the same concepts involved in the proposed agent solution to the more general area of management, *i.e.* involving other management functional areas such as fault management, configuration management, accounting management, performance management, and security management; it should be mentioned, though, that 'monitoring' is a fundamental part of any of those functional areas;
- platform- language- or implementation-specific issues;

1.8 A Road Map of this Thesis

The thesis is composed of two parts. The first one includes this introductory chapter together with Chapters 2 and 3. Chapter 2 provides some background information on the various topics touched by this interdisciplinary thesis. These include a description of the various approaches to performing monitoring, *i.e.* centralised, static distributed, dynamic or active distributed monitoring; a survey of MAs theories, applications, benefits, problems, types of mobility, etc.; and the presentation of the server location problem as it has been dealt with in the area of transportation theory from a graph theoretical point of view.

Chapter 3 complements the previous one by focusing more closely on the review of work which is related to the thesis. Management by Delegation (MbD) is thoroughly surveyed because it represents the first concrete attempt to employ code mobility in the field of management. We follow the evolution of MbD since 1991 up to more recent times. The result is that the MA paradigm adopted in this thesis is much more powerful than the MbD idea. We then focus on work which aimed at exploiting MAs in the particular field of management, dedicating more space to the specific area of distributed monitoring. Finally, location algorithms are surveyed and the need for novel solutions suitable to the problem of locating MAs optimally is highlighted.

The second part of the thesis includes the description of the proposed approach and its evaluation and discussion. This part is initiated in Chapter 4 which introduces the proposed dynamic or active distributed monitoring approach. A simpler solution, which is computed in a centralised fashion, allows focusing on the basic algorithmic ideas behind the proposal. The same principles apply to the distributed version of that algorithm, which is described afterwards. The main features of the proposed agent-based monitoring system are finally summarised. These include its dynamic behaviour, *i.e.* agent locations are not pre-determined at design time but are computed on-the-fly depending on the state of the network and of the monitoring task; its increased scalability with respect to network diameter, number of nodes, and number of MAs; and its ability to adapt to network changes through agent migration and cloning.

The method adopted to assess the proposed agent-based approach is described in Chapter 5. A mixed approach is adopted. The hypothesis is assessed partly by mathematical modelling and partly by simulations. The chapter goes through the method in detail, explaining and motivating the various choices by relating them to the hypothesis under examination.

The assessment of the proposed approach starts with Chapter 6, which presents a theoretical evaluation of the agent deployment process for general network topologies (the evaluation is based on mathematical modelling). The asymptotic complexity of the proposed deployment algorithm is studied in order to assess its scalability and typical overheads, including agent deployment time and incurred traffic. These are overheads because they are not present in conventional centralised or static distributed monitoring. The monitoring system does not operate properly during agent deployment; therefore, deployment time is an issue. Upper bounds on deployment time are calculated and are shown to increase linearly with the network radius.

Chapter 7 elaborates more on the agent system optimality. It first theorises about the topological conditions under which agents end up in near-optimal locations, which is to say they result in near-minimal total traffic incurred by the monitoring system. Upon giving sufficient conditions for location near-optimality, the chapter covers a detailed mathematical study of the agent system both at transient time (*i.e.* during agent deployment time) and at steady state (*i.e.* during the execution of the monitoring task) under the stated near-optimal conditions. Mathematical models are given for the agent system and for two flavours of the centralised polling approach, namely *naïve centralised polling* and *optimal centralised polling* and a comparative study of the various approaches is presented.

Chapter 8 is dedicated to the study of the agent system at steady state, under general conditions. The purpose is to concentrate on the phenomena that follow the initial agent deployment process and achieve a quantitative evaluation of the performance benefits of the agent approach in comparison to both centralised and static distributed monitoring. This simulation-based analysis includes also the assessment of the goodness of the agent location algorithm by evaluating their distance from optimality, and an initial study of the ability of the agent system to adapt to variations in the network status through agent migration. The computed agent locations are shown to be near-optimal whilst the algorithmic computational complexity is polynomial. This is a key result of the thesis work. In fact, in Chapter 7 we found restraining conditions on the network topology for near-optimality. Whereas the simulations show that the agent locations computed by the proposed algorithm are near-optimal even for general, Internet-like networks.

In Chapter 8 we also carry out a preliminary study of the adaptability of the proposed agent system with respect to changing network conditions. Simulation results show a particular ability of the agent system to adapt to link failure or congestion.

Chapter 9 concludes this thesis, summarising the main results and contributions. It discusses the whole work, drawing the conclusions, and elaborating on future developments of this research.

Chapter 2

Background

This chapter provides an overview of the various topics involved in this interdisciplinary thesis. Since the thesis examines a novel approach to realising *distributed monitoring*, the first area involved is the one of *management* of distributed systems (Section 2.1); monitoring is, in fact, a fundamental function of management systems. We first describe the three commonly agreed management architectures; namely, *centralised*, *hierarchical*, and *distributed* architectures. The focus is then shifted towards a more detailed taxonomy of management paradigms that will lead to a categorisation of more innovative approaches to network and system management. The *active distributed monitoring* system proposed in this thesis follows one of those approaches, namely the *strong distributed hierarchical* paradigm.

The thesis is about applying Software Agent concepts to Network and System Management (N&SM) and, more specifically, examines the use of *mobile autonomous agents* for distributed monitoring. The second area reviewed in this chapter is therefore that of mobile agents (MAs). (Section 2.2.)

It should be noted that in the context of this thesis the term ‘agent’ is overloaded with two completely different meanings, depending on whether it is used in the context of traditional management or in the one of software agent theories. In the Network and System Management community the word ‘agent’ is inherited from the *manager-agent* paradigm, one of the building blocks of the OSI and SNMP management framework. Conversely, the *software agents* community refers to the term ‘agent’ as an autonomous entity containing the logic to perform a given task and migrate under its own control from machine to machine.

The third area involved in the thesis work is the one of *Location Theory*, which involves the study of algorithms to find the location of *p* service facilities in a network with *N* nodes, that is to solve so called *facility location problems* (Section 2.3). An extensive literature is available on this classic problem which has been tackled since the early 60's in the context of *transportation theory* and of *computer networks*. We shall focus on the *centre problem* and on the *median problem*, which are strongly connected with our *agent location problem*, that is the one of locating our MAs centrally in the network in order to minimise traffic and/or response time involved in the monitoring process. Those problems are very complex to solve in general and are usually *NP*-complete when striving for optimality. We shall formulate those problems here, whereas some of the approximate algorithms available in the literature will be reviewed in Chapter 3. Therein, we shall reach the conclusion that none of those algorithms satisfies the requirements of the agent location problem.

It should be stressed that the purpose of the present chapter is to introduce the main concepts and issues involved in each of the above three research areas and identify some of the gaps that motivate the need for further work. A detailed and extensive review of those disciplines is already present in the literature and is, hence, beyond the scope of this thesis. What will be surveyed in greater detail is the work which is strongly related to the approach proposed in this thesis, this is the subject of Chapter 3.

2.1 Management of Networked Systems

2.1.1 What is Monitoring?

The concepts involved in the process of monitoring distributed systems have been described extensively in the literature. An annotated bibliography on network management is, for instance, reported in [Znaty 94]. Hence, the purpose of this section is to recall some of those fundamental concepts to set the scene of the thesis work rather than providing and exhaustive background on distributed monitoring. This section is based on [Sloman 94, Hegering 98], and [Leinwand 96] to which the interested reader may refer for further details.

Monitoring referred to by Sloman as <<the essential means for obtaining the information required about the components of a distributed system in order to make management decisions and subsequently control their behaviours>> [Sloman 94]. Whereas Joyce *et al* define monitoring as <<the process of dynamic collection, interpretation and presentation of information concerning objects or software processes under scrutiny>> [Joyce 87].

Monitoring is needed to perform a large variety of tasks; for instance, for program debugging, testing, visualisation and animation. In the context of this thesis, monitoring is seen as an important part of general management activities, which have a more permanent and continuous nature such as, performance management, configuration management, fault management or security management. Moreover, particular reference to monitoring in the context of network management is given, though the thesis intends to be applicable to the more general field of monitoring of large-scale distributed, networked systems.

Sloman also identifies the following four monitoring activities performed in a loosely coupled, object-based distributed system [Sloman 94]:

1. *Generation:* Important events are detected and event and status reports are generated. These monitoring reports are used to construct monitoring traces, which represent historical views of system activity.

2. *Processing:* A generalised monitoring service provides common processing functionalities such as merging of traces, validation, database updating, combination, correlation, and filtering of monitoring information. They convert the raw and low-level monitoring data to the required format and level of detail.

3. *Dissemination:* Monitoring reports are disseminated to the users, managers or processing agents who require them.

4. *Presentation:* Gathered and processed information is displayed to the users in an appropriate form.

This thesis addresses more directly the first two activities by considering the use of MAs for generating and processing monitoring information. Nevertheless, agents are naturally excellent candidates to provide dissemination and presentation of monitoring information as well. For instance, agents may be dual-role components receiving reports and passing processed information ‘upwards’.

Monitoring data is generated in the form of *status* and *event reports*, and according to different modalities. For example status reporting can be either *periodic* or *on request*, and events can be detected by either software of hardware *probes* and reported in a variety of different *formats*. A sequence of such reports is used to generate a *monitoring trace*. Finally, monitoring data can be generated according to two different models: the *polling model* and the *event-driven* model.

As far as processing of monitoring data is concerned, herein we shall consider some of the basic operations like *merging* of monitoring traces, *combination* of monitoring information (*i.e.* to increase the level of abstraction of data), *filtering* of monitoring information (*i.e.* to reduce the amount of data), and *analysis* of monitoring information (*e.g.* to determine average or mean variance values of particular status variables, trend analysis, diagnosis etc.). Finally, we shall consider only a simple dissemination scheme based on broadcasting of all reports to all users. We should notice that this scheme only works if there are relatively few managers. In general more sophisticated dissemination schemes based on *subscriptions* are necessary ([Sloman 94] page 321). In that case, specific reports are sent only to those management entities who have preliminary expressed an interest in (*i.e.* they have subscribed to) those reports.

An important characteristic of monitoring systems is their *intrusiveness*, which can be defined as <<the effect that monitoring may have on the behaviour of the monitored system>> [Sloman 94]. Intrusiveness results from the monitoring system sharing resources with the observed system (*e.g.* processing power, communication channels, storage space). Intrusive monitors may alter the timing of events in the system in an arbitrary manner and can lead to problems such as: degradation of system performance; a change of the global ordering of these events; incorrect results; an increase in the execution time of the application; masking or creating deadlock situations. Delays in transferring information from the place it is generated to the place it is used means that it may be out of date. For this reason it is very difficult to obtain a global, consistent view of all components in a distributed system.

A fundamental property of monitoring systems is therefore their *scalability*. Scalability is defined in [Casavant 94] as <<the ability to increase the size of the problem domain with a small or negligible increase in the solution's time and space complexity>>. Hence, in the context of this thesis scalability can be defined as the ability to increase the size of the monitored system and the accuracy of the monitoring system, with a small or negligible decrease in performance.

It should be mentioned that accuracy is related to scale because a higher level of accuracy usually results in larger resource consumption. For instance, if polling is used to collect monitoring information, higher polling rates are necessary to increase the system accuracy.

Scalability is strongly dependent on the architectural features of the monitoring system that is, in turn, part of a more general management system. Hence, the next sections review the key management architectures and management paradigms.

2.1.2 Classic Management Architectures

Management systems can use various architectures to provide functionality. The three most common ones are [Leinwand 96]:

- Centralised;
- Hierarchical;
- Distributed.

A *centralised* architecture has the management platform on one computer system, at a location that is responsible for all management duties. This system uses a single centralised database. For instance, in the case of a network management system, the single location of a centralised architecture is used to collect and process all network alerts and events, to retain all network information, and to access all management applications.

Having all the management applications and information at one point is advantageous because is useful for troubleshooting and problem correlation and provides convenience, accessibility and security for the manager. However, this architecture is weak for various reasons, as pointed out in [Goldszmidt 96b, Martin-Flatin 00], and [Pavlou 96]. First, it *does not scale*. As the number of elements of the monitored system grows, monitoring traffic, in turn, increases and tends to overload the network resources located in the proximity of the management station. In addition, the processing load at the management station increases. Thus, this model suffers from both communication and processing bottlenecks caused by the need to transmit and process large amounts of data at the management station. There can also be an 'implosion effect', with all the responses traversing the small area of the network adjacent to the management station.

Another problem is that the centralised architecture *is not robust* because the management station is a single point of failure. If the connection from the management station to the network gets severed, all management capabilities are lost. In addition, this approach can be *expensive* because it requires powerful management stations in terms of memory and processing capability.

Furthermore, centralised management tends to be *static* and *inflexible* for different reasons. One is that it may not be able to respond rapidly to dynamic changes in the state of the underlying network infrastructure. It is in practice unfeasible for a central station to have an accurate snapshot of the network status especially if the system is characterised by high-frequency variations.

The other reason is that, in practice, systems following this approach concentrate all the management intelligence in the management station and rely on pre-defined management functionalities which require complicated software update procedure to be changed. In network management this approach is exemplified by protocol-based SNMP management ([Stallings 93] and [Stallings 96]). For instance, SNMP only supports basic operations such as 'get' and 'set' for the manipulation of network parameters. More sophisticated operators are provided by RMOM; however, these are predefined and still limited in functionality.

One way to pursue increased performance and scalability is to adopt a *hierarchical* management architecture, which uses multiple systems with one system acting as a central server (the main management station) and the others working as clients. Some of the functions of the management system reside within the server; others run on the clients, which act in the role of ‘area managers’. For instance, in network management separate client systems can be configured to collect and pre-process raw data from different portions of the network. Hierarchical monitoring can be realised in the Telecommunications Management Network (TMN) [M3010 91], which uses currently OSI Systems Management (OSI-SM) as the base management technology [Yemini 93]. In a similar fashion, in the context of SNMP simple monitoring and statistical probes can be introduced using RMON [Waldbusser 95, Stallings 96], which is equivalent to an area manager that collects monitoring information about a number of elements within a subnetwork.

The common denominator of these approaches is the adoption of simple, pre-defined functionality that can result only in a limited level of decentralisation of management intelligence. Monitoring functionality that can actually be decentralised is restrained to operations such as low-level filtering of monitoring data, generation of alarms on the basis of simple conditions, and collection of rudimentary statistical information. In addition, these decentralised area managers operate in pre-defined network locations, which means that they cannot easily adapt to network changes. Therefore, conventional hierarchical schemes, despite coping with the scalability problem to a certain extent, inherit the other problems of centralised management and cannot easily cope with frequently changing, dynamic environments. They are still inflexible and static solutions. For instance, once a task has been defined in an agent (via RMON, or CMIP/S with M_ACTION), there is no way to modify it dynamically; it remains static. Moreover, those system have a limited range of pre-defined actions which can be invoked but cannot be programmed in an arbitrary way.

The *distributed* architecture combines the centralised and hierarchical approaches. Instead of having one centralised system or a hierarchy of area managers controlled by the main station, the distributed approach uses multiple peer management systems; that is a network of managers in which there is no clear-cut allocation of resources to management systems. The type of allocation depends on many factors. For instance, the same resources can be allocated to several managers if, say, one manager is responsible for security management and the other one for performance management.

Distributed architectures have been the subject of intensive research in recent years and deserve a more detailed categorisation. Hence, a more refined taxonomy of distributed architectures is presented in the next section along with practical examples.

2.1.3 A New Taxonomy of Management Paradigms

A less conservative taxonomy of management paradigms has been proposed by Martin-Flatin *et al* in [Martin-Flatin 97a, Martin-Flatin 97b], and [Martin-Flatin 00]. The main concepts are summarised herein because they help identifying the approach followed in this thesis. To begin with they distinguish between two types of approaches: *centralised* and *distributed* paradigms. The former, reflects the features of the centralised architectures discussed in the previous section. Then, they further divide the distributed paradigms into three categories, leading to the following paradigms:

- Centralised paradigms;
- Weakly distributed hierarchical paradigms;
- Strongly distributed hierarchical paradigms;
- Strongly distributed co-operative paradigms.

The weakly distributed hierarchical paradigms are substantially equivalent to the hierarchical architecture discussed in the previous section. These are characterised by the fact that the management-application processing is concentrated in few managers, whereas the numerous agents are limited to the role of dumb data collectors.

Centralised and weakly distributed paradigms can be regarded as traditional management paradigms, whereas strongly distributed paradigms encompass the more recent approaches to network and system management. We shall describe the latter approaches and identify the thesis scope.

Strongly distributed paradigms decentralise management processing down to every agent. Management tasks are no longer confined to managers: all agents and managers take part in the management application processing. The potential of large-scale distribution over all managers and agents was anticipated by Yemini *et al*, in 1991, when they devised the manager-agent delegation model [Yemini 91]. Those ideas were then fully demonstrated in Network and System Management by Goldszmidt with his Management by Delegation (MbD) framework which sets a milestone in this research field [Goldszmidt 96b]. With MbD, network devices were suddenly promoted from dumb data collectors to the rank of managing entities.

MbD triggered extensive research work on strongly distributed network and system management. In fact, many strongly distributed technologies have been suggested in the recent past. Martin-Flatin *et al* group them into three sets of paradigms [Martin-Flatin 00]: *mobile code*, *distributed objects*, and *intelligent agents*. The first two belong to the category of strongly distributed hierarchical paradigms, which are of major interest in the context of this thesis. The latter, belongs to the category of strongly distributed co-operative paradigms.

Mobile code paradigms encompass a vast collection of very different technologies, which share the idea of providing flexibility by dynamically transferring programs into agents and having these programs executed by the agent. The application of code mobility and particularly of MAs to network and system management is closely related to this thesis and will be discussed in greater detail in Chapter 3. Examples of platforms used to implement this approach include Agent Tcl, Ara, Sumatra, Telescript, Aglets, Facile, Mole, Obliq, and Tacoma (see references in [Fuggetta 97]). In 1996, two working groups were created, one by IETF and another by the ISO, in order to integrate mobile code concepts in their respective management frameworks. This resulted in the definition of the Script MIB ([Levi 96, Levi 99, Schonwalder 97, Schonwalder 99], and [Schonwalder 00]) and the Command Sequencer management function [X753 97].

Distributed object technologies represent a second type of strongly distributed hierarchical paradigms. For brevity, only the main ones are summarised below.

- *The Common Object Request Broker Architecture (CORBA)* [Siegel 96]. The Joint Inter-Domain Management (JIDM) group has been created to provide tools that enable management systems based on CMIP, SNMP, and CORBA to interoperate. CORBA has received wide acceptance in telecommunications, where it is gradually becoming a *de facto* standard. The Telecommunications Information Networking Architecture Consortium (TINA-C) [Barr 93] selected CORBA for its distributed processing environment in 1996 and, more recently, most telecommunication equipment vendors are gradually incorporating it to manage their switches.

- *Distributed Java.* After Java Remote Method Invocation (RMI) [JDK-RMI] was released in 1997 a new way of managing networks and systems gradually emerged. Java RMI makes it possible to program a management application like a distributed object-oriented application. When Java-RMI is combined with Object Serialisation [JDK-OS] (which allows objects to be transferred from host to host) management application designers have a powerful technology that allows exploitation of combination of mobile code and distributed object technologies. The Java Management API Architecture (JMAPI) is a set of tools to build management applets supporting RMI [JMAPI 96]. It supports MIB-II (see [McCloghrie 91] and [McCloghrie 94]) by mapping all managed objects onto Java objects. The Java Dynamic Management Kit (J-DMK) [JDMK 98] is a component-oriented management toolkit based on JavaBeans which comes with a library of core management services and can communicate via

RMI, HTTP, and SNMP. This toolkit allows to both ‘push’ and ‘pull’ code and offers a very powerful way of building a strongly distributed management application.

- *Web-based Enterprise Management (WBEM).* The WBEM consortium was launched by Microsoft in 1996, about the same time that development started on the JMAPI, with the goal of developing an open, vendor-neutral architecture for web-based enterprise management. This initiative is now evolving under the influence of the Desktop Management Task Force (DMTF), and it is difficult to assess whether it will succeed in its attempt to establish new management standards [Thompson 98].
- *The Open Distributed Management Architecture (ODMA)* [ISO-WG4 95]. The purpose of ODMA is to extend the OSI management architecture and, thus, the TMN architecture with the Reference Model of the ISO Open Distributed Processing (RM-ODP) framework, which provides the specification of large-scale, heterogeneous distributed systems. In ODMA, there are no longer managers and agents with fixed roles as in OSI management. Instead, computational objects may offer some interfaces to manage other computational objects (manager role), and other interfaces to be managed (agent role). ODMA also renders the location of computational objects transparent to the management application.

Finally, strongly distributed co-operative paradigms are of marginal relevance to the thesis and are only briefly discussed for the sake of completeness. In this case, the agents implementing the management task originate from the Distributed Artificial Intelligence (DAI) community and, more specifically, from Multi-Agent Systems (MASs), where researchers are modelling complex systems with large groups of intelligent agents (IAs). What has not been achieved is a common understanding of what an IA is. An approach which has encountered a great deal of success outside the realm of DAI is the one proposed in 1994 by Wooldridge and Jennings. Instead of coming up with a precise definition of what an IA should or should not be, they have defined a core of compulsory properties and considered any other property to be application specific. For them, IAs must exhibit four properties (see [Wooldridge 94] and [Wooldridge 96]):

- *Autonomy*. An IA operates without direct human intervention, and has some kind of control over its actions and internal state.
- *Social Ability*. IAs co-operate with other IAs (and possibly people) to achieve their goals, via some kind of agent communication language.
- *Reactivity*. An IA perceives its environment and responds in a timely fashion to changes that occur in it.

- *Pro-activeness*. An IA is able to take the initiative to achieve its goals, as opposed to solely reacting to external events.

Whereas optional properties of IAs include *mobility*, *veracity* (IAs do not knowingly communicate false information), and *rationality* (IAs are not chaotic, they act so as to achieve their goal).

Later on, Franklin and Graesser revisited the mandatory and optional properties of IAs. For them, IAs must be *reactive*, *autonomous*, *goal-oriented* (pro-active and purposeful), and *temporally continuous* (continuously running). They can be optionally *communicative* (*i.e.* able to communicate, co-ordinate, and co-operate with other agents), *able to learn*, *mobile*, and have a *human-like character* [Franklin 96].

When IAs are co-operative, they are exposed to heterogeneity problems and, therefore, need standards for agent management, communication languages, etc. The Foundation for Intelligent Physical Agents (FIPA) consortium is currently working on such standards (see [Chiariglione 98, FIPA] and [FIPA 98]).

Finally, more and more researchers are now trying to use IAs to manage networks and systems. Some examples are described in [Busuioc 94a, Busuioc 94b, El-Darieby 98, Grimes 96, Keller 96, Leckie 97, Lewis 97, Magedanz 95, Magedanz 96b, Mountzia 97a, Mountzia 97b, Mountzia 98, Sahai 97a, Somers 96, Steenekamp 96, Wies 97, Zhang 96, Zhang 97]. A state-of-the-art paper on the application of IA to management is described in [Cheikhrouhou 98].

2.1.4 Approaches to Delegation of Management Responsibility

An important aspect of distributed management is the ability of delegating management responsibility to area managers and peer management entities. In the enterprise world, *delegation* has been defined as the process of transferring *power*, *authority*, *accountability*, and *responsibility* for a specific task to another entity (see [Evans 86] and [Mullins 89]). In distributed N&SM, delegation always goes down the management hierarchy [Martin-Flatin 00]: a manager at level {N} delegates a task to a subordinate at level {N+1}. This is known as *downward delegation*, which is an example of *vertical delegation*, typical of hierarchical paradigms. Contrary to vertical delegation, *horizontal delegation* involves peer entities at the same level and is typical of distributed co-operative paradigms used in DAI.

Delegation is normally a *one-to-one relationship*, between a manager and an agent in hierarchical management, or between two peers in co-operative management. In the latter case, there could also be *one-to-many relationship* where a task is collectively delegated to a group of entities. Many-to-many relationships are really in the realm of co-operation rather than delegation [Martin-Flatin 00].

Mountzia and Dreo-Rodosek suggest that two additional forms of delegation are useful in management: *static* and *dynamic delegation* [Mountzia 96]. With static delegation the functionality is pre-allocated to distributed entities at management design time and cannot be dynamically expanded or changed without re-programming and re-compilation on the remote entity. This approach is usually acceptable for standard, routine management tasks such as low-level monitoring with RMON [Stallings 96].

On the other hand, if dynamic delegation is supported, functions can be delegated during the operational phase of management, which improves the flexibility of the management system. This approach enables a management system to be adjusted to changes and/or new requirements at run time. For instance, in network management, a flexible expansion of the network element functionality is possible, as capability can be modified at any time by simply uploading new executable code to the elements. This capability is, for instance, supported by MbD and is particularly important for managing large-scale heterogeneous networks. These networks may evolve very rapidly making it quite difficult for the management system designer to foresee every single management need at design time.

Mechanisms for dynamic delegation can support both *spatial distribution —i.e.*, distribution over different nodes— and *temporal distribution —i.e.* distribution over time— of management functionality ([Goldszmidt 96c] and [Goldszmidt 98]). Spatial distribution reduces the network bandwidth used for management purposes and the delays involved in accessing remote data. For example, applications that need to evaluate and react to transient events of short duration, such as noise burst in a line, should be distributed to the devices. In general, spatial delegation provides the means to distribute the monitoring functionality, compress data locally, and to send to a manager only the most significant information.

Temporal distribution is the ability to dynamically delegate new management code to a device when it is needed. Temporal distribution assists the developers of management applications to modify their management policies as administrative requirements change, and as the managed environment grows and evolves. Management applications can use dynamic code delegation to address temporal problems, such as the detection of intrusion attempts on a networked workstation, reacting to problem situations or performing load balancing. The network health-checking example described in [Goldszmidt 93] shows the need for this capability. As the parameters defining the health of a network vary among different systems and during different times, health functions should be dynamically bound to agents when needed.

Both spatial and temporal delegation can be initiated either by a manager or by system events. The former is also called *user-driven delegation*, while the latter is usually referred to either as *event-driven* or *data-driven delegation*. In other words, delegation can be caused either by the changing requirements on the applications and services or by changes on the distributed system. In particular, event-driven delegation can provide a very powerful mechanism to increase dramatically the autonomy and survivability of a system, providing a fast adaptability to resource constrains [Goldszmidt 96a, Goldszmidt 96c].

To conclude we can say that this thesis falls under the umbrella of vertical delegation, whereby a main management station distributes the main monitoring task to one or more MAs that, in turn, may distribute their part of the sub-task to other agents. As will become clearer after the proposed agent system will be described, agent location is computed dynamically and can change over time. Hence, dynamic, spatial, and temporal distribution are pursued as well. Finally, agent migration can be triggered by network or system events according to the event-driven delegation concept.

2.2 Mobile Agents Overview

After almost a decade of discussion some concepts and definitions surrounding the agent world are still blurred since no strong agreement has been reached by the various research communities interested in this area. This thesis does not aim at contributing toward those theoretical aspects involved in agent architectures and terminology. As such, we propose a more pragmatic approach to introducing MAs that starts off by focusing on two important features, *code mobility mechanisms* and *distributed systems design paradigms*. That will lead us to the definition of MAs which is used in the thesis. To conclude, a panorama of the MAs main advantages, open issues, and general applications is given. The particular application of MAs to the management realm is of strong relevance to the thesis and is, therefore, covered in greater detail in Chapter 3.

2.2.1 Code Mobility Mechanisms

In this section we introduce the main code mobility mechanisms [MCB] based on the classification proposed in a seminal paper on this topic by Fuggetta *et al* [Fuggetta 97]. The concepts presented there are also elaborated by other authors in [Baldi 97, Cugola 96, Cugola 97, Ghezzi 97], and [Carzaniga 97]. The large number of references to those papers suggests wide acceptance of the concepts and terminology presented therein.

The code mobility concept originates in the work aiming at supporting the migration of active processes and objects (together with their state and associated code) at the operating system level. For instance, migratory systems such as Locus [Thiel 91] and Cool [Lea 93] support transparent process and object migration respectively, regardless of the underlying hardware substrate. Transparent migration at operating system level addresses the issues that arise when *code* and *state* are moved among the hosts of a loosely coupled, small-scale distributed system. This approach does not tend to be suitable for large-scale networks and systems, particularly those of the scale of the Internet. Papaioannou in his recent PhD thesis builds an argument against *location transparency* in large-scale systems [Papaioannou 00a]. He argues that location transparency, by creating the illusion that all components exist within the same machine, breaks the fundamental computing layers of abstractions.

A more recent approach for large-scale distributed systems is the one of *location-aware programming* based on Mobile Code. This approach, contrasts the philosophy of location transparency and exhibits several innovations [Fuggetta 97]:

- *Code mobility is exploited on a large scale rather than a small scale.* In large-scale systems networks are composed of heterogeneous hosts, managed by heterogeneous authorities, connected by heterogeneous links.
- *Programming is location aware and mobility is under the programmer's control.* Location of computational entities may have a significant impact on heterogeneous, large-scale systems. Location-aware applications may take actions based on the knowledge of the locations of the other application components.
- *Mobility is not performed just for load balancing.* Contrarily to operating system level migration, mobile code systems are oriented towards service customisation, dynamic extension of applications functionality, and support for nomadic computing.

Existing Mobile Code Systems (MCSs) support two forms of code mobility for large-scale systems: weak and strong mobility. A small parenthesis on MCSs needs to be open in order to introduce some important concepts and terminology, before elaborating more on those forms of mobility.

The main difference between traditional systems – *e.g.* those supporting location-transparent migration at operating system level – and MCSs is that the former provide network transparency by means of a *True Distributed System* layer; whereas the latter manifest to the programmer the structure of the underlying computer network. An example of the first case is CORBA in which the programmer is never aware of the network topology and always interacts with a single well-known object broker [Siegel 96]. Conversely, in MCSs a *Computational*

Environment (CE) is layered upon the network operating system of each network host. The CE maintains the identity of the host where it is located and provides the application with the capability to dynamically relocate computation onto different hosts.

Continuing to follow the approach presented in [Fuggetta 97], we can distinguish the components hosted by the CE as *Executing Unit* (EU) and *resources*. EUs represent sequential flows of execution, such as single-threaded processes or individual threads of multi-threaded process. Resources are entities which can be shared among multiple EUs, such as a file or an object shared by different threads. In turn, an EU may be modelled as a composition of a *code segment*, which provides the static description of a computation behaviour, and a *state* composed of a *data space* and an *execution state*. The data space is the set of references to resources that can be accessed by the EU. The execution state contains private data that cannot be shared, as well as control information related to the EU state, such as call stack and instruction pointer.

Considering the above definitions, Fuggetta *et al* define *strong mobility* as the ability of an MCS to allow migration of both the code and execution state of an EU between different CEs. Whereas, *weak mobility* is the ability of an MCS to allow code movement across different CEs. Code may be accompanied by some initialisation data but no migration of execution state is involved in weak mobility.

A mobility mechanism orthogonal to execution state migration relates to *data space management*. Upon migration of an EU to a new CE, its data space – *i.e.*, the set of bindings to resources accessible by the EU – must be rearranged. This may involve voiding bindings to resources, re-establishing new bindings, or migrating some resources to the destination CE along with the EU. Fuggetta *et al* describes this particular aspect of code mobility in detail in [Fuggetta 97]. This is not a central theme in this thesis and is, hence, not discussed in greater detail.

The most comprehensive form of code migration is termed *higher-order mobility*, the ability of an MCS to allow migration of code, execution state, and data space of an EU between different CEs. Higher-order mobility is usually realised over systems based on the *continuation-passing* style. In this context, a function together with its defining scope is termed *closure*; a *continuation* is a closure that represents the state of execution; and a *closing environment* includes the data space. Hence, higher-order mobility is defined as the ability to transfer arbitrary closures and continuations, together with their closing environment.

To summarise, there are five main code mobility mechanisms or paradigms:

- *Process Migration.* Concerns the transfer of an operating system process from the machine where it is running to a different one. Process migration facilities manage the binding between the process and its execution environment and operate at operating system level. This has been used in loosely coupled, small-scale distributed systems to achieve load balancing across network nodes.
- *Object Migration.* Object migration makes it possible to move objects among address spaces, implementing a finer grain mobility with respect to systems providing migration at the process level. In some cases object migration is achieved transparently, that is without user intervention or knowledge.
- *Weak Mobility.* Weak mobility is the ability of an MCS to allow code movement, along with its initialisation data, across different CEs. This is the first type of mobility supported in current MCSs which is meant to target the requirements of large-scale distributed systems. This is the mobility mechanism exploited in the context of this theses.
- *Strong Mobility.* Strong mobility is the ability of an MCS to allow migration of both the code and execution state of an EU between different CEs. This mechanism involves significant migration overheads related to the need to save the execution state along with the code. These overheads may be contained with *discrete migration*, *i.e.*, if migration is triggered only at particular moments when the execution state is relatively small. The work presented in this thesis does not exploit directly strong mobility, although in principle it could exploit *discrete strong mobility*.
- *Higher-order Mobility.* Higher-order mobility is the ability of an MCS to allow migration of code, execution state, and data space of an EU between different CEs. The overheads associated to this mechanism are even heavier than those of strong mobility. Hence, higher-order mobility is not exploited in this thesis. An in-depth study of higher-order mobility is presented in [Halls 97].

It should be noted that a simple degree of data space management will be required also in weak and strong mobility. What differentiates higher-order mobility from weak and strong mobility is the focus on both execution state and data space management during migration.

2.2.2 Design Paradigms

While in the previous section we focused on the mobility mechanisms, here we identify the main software design paradigm behind those mechanisms. The main design paradigms according to the classification presented in [Cugola 96, Baldi 97, Carzaniga 97, Fuggetta 97, Ghezzi 97], and [Baldi 98] are:

- *Client-server (CS).* In the CS paradigm, a server component exports a set of services. The code implementing those services is owned by the server component – *i.e.* the server holds the *know-how*. It is the server who executes the services; thus, it has the *processor* capability. The server also has the *resources* which are accessed by the client through the server. The CS paradigm is widely adopted in network-centric applications but cannot be viewed as a paradigm for mobile computations because no mobility takes place.

- *Remote Evaluation (REV) or Code Pushing.* REV has been introduce by Stamos and Gifford in their pioneering work described in [Stamos 90a] and [Stamos 90b]. In REV, an application in the client role can dynamically enhance the server capability by sending code to the server. Subsequently, clients can remotely initiate the execution of this code that is allowed to access the resources collocated within the server. Therefore, this approach can be seen as an extension of the CS paradigm whereby a client, in addition to the name of the service requested and the input parameters, can also send code implementing new services. Hence the client owns the code needed to perform a service, while the server offers both the computational resources required to execute the service and the access to its local resources. The REV principles have led to the more recent approach usually referred to as *code pushing*. This design paradigm usually relies on the object migration mechanism or on the weak mobility mechanism.

- *Code on Demand (COD) or Code Pulling.* In COD a client downloads or *pulls* required code from a code repository (or code server) and links it dynamically in order to perform a task. Hence, the client owns the resources needed to perform a service but lacks part of the code (or logic) required to perform it. The COD principles have led to the more recent approach usually referred to as *code pulling*. Similarly to REV, COD usually relies on the object migration mechanism or on the weak mobility mechanism.

- *Mobile Agent (MA).* An MA is essentially an autonomous EU containing the logic to perform a given task and migrate under its own control from machine to machine in a heterogeneous network. While in execution at a given node, the MA is able to suspend its execution at an arbitrary point, transport itself to another machine, seamlessly resume execution from the point of suspension, and possibly gain local access to the resources of the new node. Hence, the agent owns the code to perform a service but lacks the resources needed to accomplish it. The server owns those resources and provides an environment to execute the code sent by the client. The mobility mechanisms adopted to realise this design paradigm are weak, strong, or higher-order mobility.

Since the work described herein adopts the MA design paradigm (and the weak mobility mechanism), the MA topic deserves further space. This is the subject of the next sections.

2.2.3 Definition of Mobile Agent

The term ‘agent’ or ‘software agent’ is often abused and rarely defined precisely. The problem of coming up with an agreed definition of agent boils down to identifying the features that distinguish an agent from a common computational entity. This has raised controversial arguments for nearly a decade and only recently those features have been identified. Theoretical studies on agents and artificial intelligence have come to the conclusion that a computational entity can be regarded as an agent if it exhibits some of the following properties: social ability, autonomy, reactivity, pro-activity, adaptability, persistency, and ability to learn, communicate, co-operate, and move [Wooldridge 95].

Some of those features actually characterise intelligent software agents. Mobility is an orthogonal property with respect to the others, that is not all agents are mobile. So an MA has the mandatory property of being mobile and exhibits a sub-set of the above agent properties. Typically, MAs are computational entities that act on behalf of some other software entity, exhibit some degree of autonomy, and are particularly featured with migration capability. In the context of this thesis MAs exhibit also re-activeness, pro-activeness, adaptability, and cloning capability, whereby *agent cloning* is the ability of agents to create and dispatch copies, or ‘clones’, of themselves.

In the proposed framework, MAs (or simply ‘agents’ for brevity) support weak mobility and are used to realise strongly distributed hierarchical management. Agents are not constrained to particular migration patterns and are not limited in the number of links they can traverse. For this reason we say that these agents realise an *unconstrained weak mobility* mechanism. Conversely, with *constrained mobility* an agent, upon its creation in the client site (by the client application) is only allowed to migrate to a remote server where its execution will be confined [Bohoris 00b] and [Bohoris 00c]. This means that only one-hop mobility is allowed since, upon the first migration, the agent is not any more permitted to migrate to other nodes.

Finally, it should be mentioned that cloning is often used to realise agent migration [Shehory 98] whereas it is used herein to realise delegation of management responsibility. In particular, agent migration and cloning are used to realise vertical, dynamic delegation, aiming at spatial and temporal distribution.

2.2.4 Example Mobile Agent Platforms

The thesis work is not tied up with any particular mobile agent platform because the hypothesis is evaluated by simulations and theoretical analysis rather than by experimental work on a prototype implementation. However, the continuation of the work described herein may involve measurements on real systems and, in turn, the choice of a suitable agent platform. Moreover, the feasibility study of the concepts developed herein has involved the consideration of the features of existing platforms to make sure that the requirements of the proposed agent system can be met by real systems. Hence, a short review of the some of the main agent platforms is reported hereafter. This review is not meant to be comprehensive because such information is already available in the literature [Cugola 96, Cugola 97, Fuggetta 97, Ghezzi 97, Gray 97, Halls 97, Pham 98, Baumann 99, Papaioannou 00a]. The focus is on the identification of platforms that might be suitable to further the thesis work. The main requirement is the support of weak mobility.

Examples of platforms supporting weak mobility are: Obliq [Cardelli 95], Aglets [IBM 99], Tacoma [Johansen 95a] and [Johansen 95b].

Examples of platforms supporting strong mobility are: Agent-Tcl [Gray 95], Mole [Straβer 96] and [Straβer 97], Tycoon [Matthes 95], Telescript [White 94] [White 95a] and [White 95b], Messenger [Di Marzo 95], Ara [Peine 97].

Examples of platforms supporting higher-order mobility are: Tube [Halls 97], Kali Scheme [Cejtin 95], Sumatra [Ranganathan 96] and [Ranganathan 97].

2.2.5 Advantages Claimed for Mobile Agents

One of the first papers that tried to identify the potential benefits of MAs is the seminal white paper by Harrison *et al* [Harrison 95], later published in [Chess 97]. The authors elaborate on the possible individual advantages of MAs to reach the conclusion that the real benefits of MAs are related to ‘aggregate’ rather than ‘individual’ advantages.

Examples of individual advantages identified by Harrison at el. are:

- *MAs can provide better support for mobile clients.* Mobile devices such as laptop computers are only intermittently connected to a network, hence have only intermittent access to a server. Clients may develop an agent request while disconnected, launch the agent during a brief connection session, and then immediately disconnect. Another advantage lies in the ability of an agent to perform information retrieval and filtering at a server and return to the client only the relevant information. This is particularly beneficial even for relatively low-bandwidth connections and addresses the issues of mobile devices characterised by limited storage and processing capacity. Therefore, an agent can play a significant role in reducing network traffic, can support asynchronous interaction, and realise remote searching and filtering.

- *MAs facilitate real-time interaction with server.* An agent executing locally at server side has relatively low and certainly bounded latency and can provide more opportunities for error recovery.
- *MA-based queries and transactions can be more robust.* Remote Procedure Call (RPC) client-server computing, though reasonably robust in LAN-based systems, tends to be less reliable over wide-area networks. MAs offer two advantages: 1) they can provide reliable transfer between client and server without requiring reliable communication (*e.g.* they can handle disconnected computing); and 2) they can deal with server unavailability by means of recovery mechanisms.
- *MAs may facilitate electronic commerce.* Although MAs do not offer any technical advantage here, they do offer advantages to vendors, service providers, and users. For instance, MAs are a plausible method for vendors to distribute the client end of a transaction protocol in a device-independent way.

However, these individual advantages can also be addressed in an *ad hoc* manner with alternative approaches, such as Remote Procedure Call (RPC) [Birrell 84] or asynchronous messaging [Cypser 91], or by redesigning communication protocols. MAs are particularly well suited to addressing various issues, providing ‘aggregate’ rather than ‘individual’ advantages – *i.e.*, the situation in which more than one individual advantage is achieved through the use of MAs.

The claims introduced by Harrison *et al* have sparked wide interest and seem to be still valid. After half a decade of intensive research work the list of individual advantages claimed to MAs has grown. However, nobody has yet found the killer application for MAs, though the conditions which make the MA design paradigm a preferable choice have been identified more clearly. In the rest of this section we summarise the latest individual advantages claimed for MAs, highlighting some of the strongest points which favour them. Some of these advantages are complementary to the ones suggested by Harrison *et al*, whereas others are included in their list.

The individual advantages of MAs, as claimed almost unanimously by [Gray 97, Rothermel 97, Bieszczad 98c, Baumann 99, Lange 98, Lange 99, Papaioannou 00a, Papaioannou 00b], are the following:

- *They reduce network load.* MAs allow users to package a conversation and dispatch it to a destination host where interactions take place locally. MAs are also useful when reducing the flow of raw data in the network by moving the computation to the data rather than the data to the computation.
- *They overcome network latency.* For critical real-time systems, network latency may be not acceptable. MAs offer a solution, because they can be dispatched from a central controller to act locally and execute the controller’s directions directly.
- *They can offload low-powered devices.* MAs allow a low-powered client such as small mobile devices to offload work to a high-powered proxy or an overloaded server offload work to clients.
- *They encapsulate protocols.* As protocols evolve to accommodate new requirements for efficiency or security, it is cumbersome if not impossible to upgrade protocol code efficiently. As a result, protocols often become a legacy problem. MAs can move to remote hosts to establish ‘channels’ based on proprietary protocols.
- *They can dynamically enhance server capability.* Because MAs can relocate computational logic, servers become much simpler. Effectively, a server becomes merely an executing environment for hosting MAs. The server capability can be dynamically extended by sending new MAs. More generally, MAs can be used to distribute or upgrade software on demand.
- *They provide a natural approach to disconnected computing.* Mobile devices often rely on expensive or fragile network connections. Tasks requiring a continuously open connection between a mobile device and a fixed network can be embedded into MAs which can operate asynchronously and autonomously. In fact, MAs do not necessarily require a permanent connection during their operation.
- *They adapt dynamically.* MAs can sense their execution environment and react autonomously to changes. MAs can distribute themselves among the hosts to maintain the optimal configuration for solving a particular problem.
- *They are naturally heterogeneous.* MAs are generally computer- and transport-layer-independent, hence can provide optimal conditions for seamless system integration.

- *They are robust and fault-tolerant.* MAs' ability to react dynamically to unfavourable situations and events makes it easier to build robust and fault-tolerant distributed systems. For instance, MAs can be dispatched to a different host upon being warned that their host is about to shut down.

- *They provide natural support for distributed computation.* MAs are inherently distributed and, as such, can be a fundamental enabler for distributed computation.

- *They potentially result in more scalable distributed applications.* Because they can be dynamically located and can maintain location optimality through migration, MAs can result in increased scalability.

The work described in this thesis intends to exploit and assess some of the above advantages. In particular, the MA's ability to reduce network load and latency, to adapt dynamically to network environment and to increase the scalability of a distributed monitoring application are considered.

2.2.6 General Mobile Agent Applications

As mentioned in the section above, nobody has yet identified a single killer application for MAs. However, there are plenty of applications that can benefit from using the MA paradigm. Examples of applications that clearly benefit from the MA paradigm have been identified by [Harrison 95, Chess 97, Fuggetta 97, Pham 98, Wooldridge 98, Baumann 99, Lange 99, Milojicic 99]. These are:

- *E-commerce and personal assistance.* MAs may be used for real-time access to remote resources in commercial transactions, and for acting and negotiating on behalf of their creator.

- *Secure brokering.* In collaborations in which not all the collaborators are trusted, the parties could let their MAs meet on a mutually agreed secure host.

- *Distributed information retrieval.* MAs can be dispatched to remote information sources where they locally create search indexes or perform extended searches on behalf of the originator.

- *Telecommunication networks services.* Support and management of advanced telecommunication services are characterised by dynamic network reconfiguration and user customisation. MAs may function as the glue keeping the systems flexible yet effective.

- *Workflow applications.* The flow of information between co-workers may be supported by MAs. In addition to mobility, they provide a degree of autonomy to the workflow item.

- *Monitoring and notification.* An MA can locally monitor a given information source and notify specific events without being dependent on the system from which it originates.

- *Information dissemination.* MAs embody the so-called Internet 'push' model. Agents can disseminate information such as news to a number of customers. They can provide access policies and implement the functionality needed to display the news article in accordance with the capability of the output medium.

- *Software deployment*. MAs can be used to automate the process of software installation and upgrading. The agents can gather information about the target environment and user preferences before transporting the software package.

- *Parallel processing.* MAs may create a cascade of clones and administer parallel processing tasks.

The above list covers general applications. The more specific class of application in the management arena is more closely related to this thesis and is, then, reviewed in greater detail in the next chapter.

2.2.7 Issues Associated to Mobile Agents

Despite their numerous benefits and application scenarios, MAs still pose several problems that require further study. Some of them are discussed in [Harrison 95, Chess 97, Rothermel 97, Oppliger 99], and [Papaioannou 00b], and are highlighted below.

- *Security.* This is one of the most emotive issues raised when discussing MA systems. Lack of security guaranties is one of the major arguments against MAs and a driver for a wealth of research in this subject [Vigna 98]. It should be mentioned that much of the effort in security has been towards host protection, whereas less work has addressed the problem of protecting the MA integrity against malicious hosts and execution environments. Another problem is that security measures often result in performance and functional limitations. Finally, the real success of MAs is conditional to overcoming security concerns in order to achieve trust on the part of third-party server providers along with their willingness to allow users to customise server behaviour.

- *Safety.* A safe environment will make sure that MAs can be executed only 'where' and 'if' a sufficient amount of resources such as processing power and local storage capacity is available and that concurrent and consistent access to the resources manipulated by the MAs is guaranteed. This also relates to the willingness of the third-party service providers to sustain the MA's computational load.

- *Secrecy*. MAs can be entrusted with private information such as user profile information, user authentication data, or user negotiation preferences. Protecting an MA is not simple since agents are invariably interpreted within an execution environment. Mechanisms that ensure that agents maintain the privacy of their originator need further attention.

- *Transactional support.* A considerable part of today's commercial applications requires a high degree of robustness. Moreover, transactional support is a pre-requisite for an increasing number of tasks. This will require a tight integration of agent technology and transaction management. There are two challenges to achieve this integration [Rothermel 97]. Firstly, the identification of transaction models which suit the asynchronous nature of agents. Secondly, the development of agent recovery mechanisms.

- *Standardisation and interoperability*. Despite the efforts toward standardising agent systems in order to allow for full interoperability (see [MASIF 97, FIPA, FIPA 98], and [Chiariglione 98]), further work is required before agents can fully interoperate and run across different agent execution environments.

- *Limited Availability of agent execution environments.* Agent execution environments are not widespread and it is difficult to propagate them onto the management site, connectivity provider site, service provider site, and optionally also at the terminal side. This is envisioned to happen in the near future but as of today it represents a problem.

- *Complexity*. MA-based management systems are likely to be quite complex to design and debug because it will be difficult to determine their behaviour in a real, dynamic network environment. Whereas their strength (in comparison to the CS design paradigm) resides in easier implementation, deployment, and maintenance. Debugging a distributed system is difficult. An MA distributed system is particularly difficult because it involves mobile autonomous software entities whose behaviour is often determined by the environment in which they sit and by their perception of it. Although some work has been done on agent design [Aridor 98], on architectural styles for agent distribution [Weir 97], and on decomposition patterns for mobile-code based management [Liotta 98b], the application of these design techniques to the field of management has not been investigated so far. Finally, it is not clear how agent-based, delegated management should be used to pursue a real automation of management itself.

- *Limited availability of quantitative performance evaluation*. The ability of the MA paradigm to result in increased performance, scalability and flexibility in comparison to the CS one has been claimed by many authors. Though acceptable in principle, this claim has not corresponded to a widespread application of the MA paradigm the management arena. One of the reasons is that not enough work has been carried out to assess its strength in a qualitative fashion. Opponents of the MA paradigm suggest that, in fact, an alternative direction could be to enable RPC-based client-server interactions to match the advantages of MAs.

- *Migration overheads.* Agent migration involves overheads that need careful consideration. With today's platforms, migration time between two hosts is in the order seconds (see [Bohoris 00c] and [Knight 99]); migration traffic depends on agent size and state and on the serialisation mechanism; finally, processing overheads are associated with the serialisation and de-serialisation process. Further study is required to reduce migration overheads which limit significantly the MA application domain.

- *Control structures.* Today's MA systems allow the creation and cloning of agents, while efficient mechanisms for controlling agent migration and termination have not been sufficiently investigated. Algorithms for agent location, termination and for orphan detection are discussed in [Baumann 99]. Agent autonomous migration is a potential source of instability if the triggered mechanisms are not well thought out and fine tuned. The stability of MA systems is another interesting subject which requires further investigation. More generally, mechanisms to manage agent mobility in the context of integrated fixed and mobile networks are needed. A mobile networking environment is particularly dynamic as it involves a large number of simple, mobile terminals. Agent control mechanisms are particularly important to prevent instability and require further work.

This thesis tackles more closely the last three issues by providing a quantitative, comparative study of performance and scalability between centralised and agent-based distributed monitoring. The thesis evaluates the impact of migration overheads and presents a simple self-controlled system based on autonomous agents.

2.3 Networks and Discrete Locations

2.3.1 The Agent Location Problem

A major requirement on the active, distributed monitoring system proposed in this thesis is its ability to place MAs efficiently within the monitored system. The system will be partitioned into sub-systems, each of which will be monitored by one or more MAs acting as *local area managers*. In this context, the *agent location problem* is the one of computing the system partitioning and the location of MAs within those partitions. This problem introduces the following constraints:

- The MAs implementing the monitoring operation have to be placed in strategic locations which result in near-minimal overall incurred traffic and/or response time;
- The MA deployment time must be characterised by a predictable upper-bound which must, in turn, be small compared with the overall duration of the monitoring operation itself.

The first problem is analogous to the one of locating multiple emergency facilities in a transport network. In that case, emergency facilities need to be placed in a way that minimises either the total travelling cost (*p-median problem*) or the maximum travelling time (*p-centre problem*).

In the case of distributed monitoring, the total travelling cost is the equivalent of the total *traffic* incurred by the agents in performing the monitoring operation, whereby network traffic is assumed proportional to the total number of links traversed by packets (see Chapter 5 for a specification of the adopted 'traffic' metric). For instance, if agents follow the "polling" technique, the cost associated with *request* and *response* packets will be proportional to the packet size and to the total number of links traversed by those packets.

Similarly, maximum travelling time is the related to the *response time* of the monitoring system (see Chapter 5 for a specification of the adopted 'response time' metric). In the case of polling-based monitoring, this is the span of time elapsed between issuing a request packet by the agent and the arrival of the corresponding response packet. We should add to that the agent-to-monitoring station communication time.

Due to the equivalence between the agent problem and the centre and median problems, the former will be specified more rigorously through the formulation of the latter problems. The formulations described below are extracted from textbooks on graph theory and location theory to which the interested reader is referred for further details ([Handler 79, Tansel 83a, Tansel

83b, Buckley 90, Evans 92, Daskin 95]). Herein we describe the basic concepts necessary to formulate the agent location problem, a focal problem of this thesis. Existing approaches to solving location problems relevant to this thesis are reviewed in Chapter 3.

2.3.2 Classification of Location Problems

Network location problems occur when a number of *facilities* or *suppliers* are to be located on a network. The network of interest may be a road network, an air transport network, a network of shipping lanes, or a communication network. Given a set of suppliers and a set of *demand points*, the goal is to locate the suppliers in a way which minimises a given *objective function* which reflects costs.

A comprehensive classification of location problems is presented in [Tansel 83a]. We report here a simplified classification presented in [Evans 92] which classifies location problems according to three characteristics:

- The potential location of the facilities to be located – either at vertices or anywhere on the network;
- The location of demand points – either at vertices or anywhere on the network;
- Objective function – either to minimise the total cost to all demand points or to minimise the maximum cost to any demand point.

A simple classification scheme is depicted in Figure 2-1, which highlights (with shaded boxes) the particular class of problems within the scope of the thesis. Being interested in communication networks both the facility and the demand nodes will be located at vertices. More specifically, in our context, a facility is an area manager – *i.e.*, an MA – and the demand points are the monitored nodes.

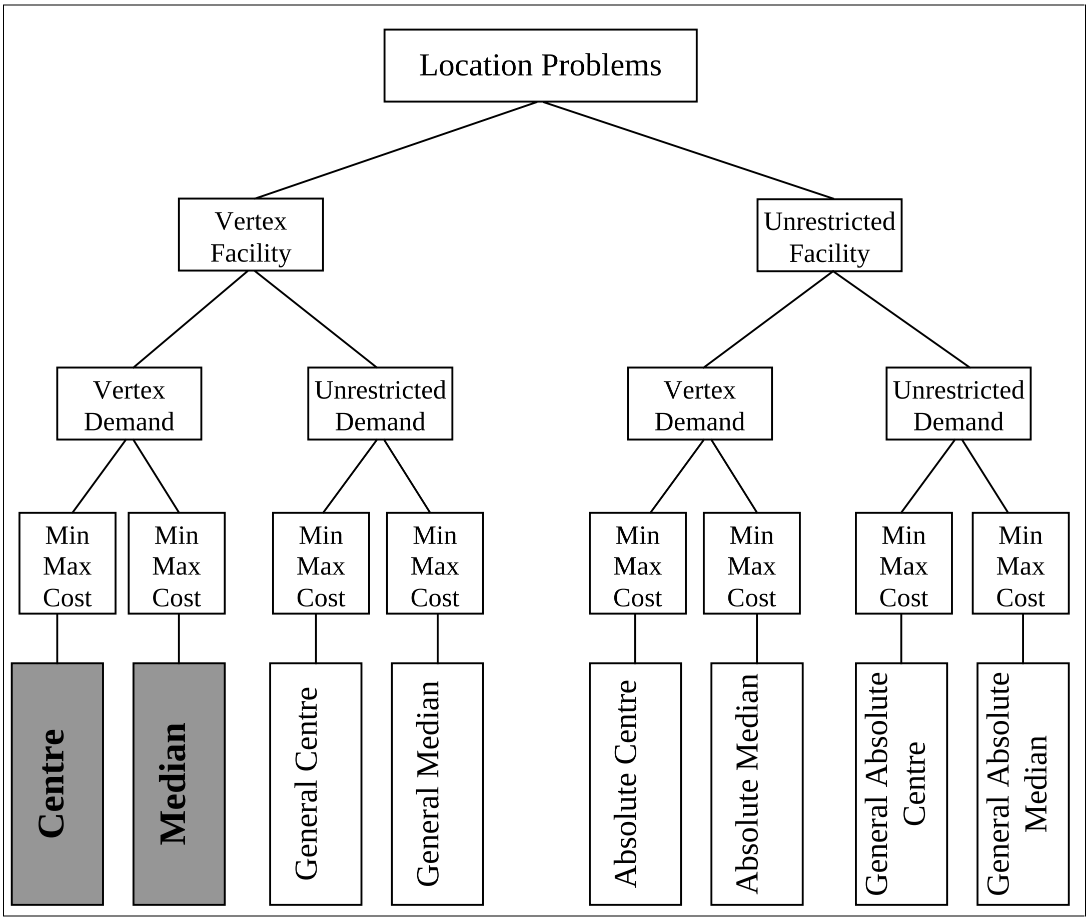


Figure 2-1. Classification of location problems according to [Evans 92].

In the case in which both facilities and demands occur only at vertices, the *centre* of a graph is any vertex whose farthest vertex is as close as possible. Other terms are used in the case of unrestricted locations. A *general centre* is any vertex whose farthest point in the graph is as close as possible. An *absolute centre* is any point whose farthest vertex is as close as possible. Finally, a *general absolute centre* is any point whose farthest point is as close as possible.

By analogy to each of these four types of location problems, we can define the *median*, *general median*, *absolute median*, and *general absolute median* simply by changing the objective function from minimising the maximum distance from the facility to a demand to minimising the sum of the distances from the facility to all demand points. This thesis focuses on centre and median problems.

In general, if the number of facilities to be located is *p*, the location problems are termed *p-centre* and *p-median*, respectively. Finally, problems with more than one objective function are termed *multi-objective location problems*.

Location problems are usually formulated as *integer linear programming* (LP) problems. A formal specification of the problems tackled in the thesis is given below. LP problems are usually formulated in terms of *inputs* (*i.e.* constants that are given in the problem definition and statement), *decision variables* (i.e the quantities that we are trying to find), and *surplus variables* (*i.e.* used to convert inequalities constraints into equality constraints).

2.3.3 Formulation of the *p*-centre Problem

In this section, we specify the *vertex p-centre problem* or *minimax problem* as formulated in [Daskin 95] (pp.160-162). We define the following notation:

Inputs

d_{ij} = distance from demand node *i* to candidate facility *j*

h_i = demand at node *i*

p = number of facilities to locate

Decision Variables

$$X_j = \begin{cases} 1 & \text{if we locate at candidate site } j \\ 0 & \text{if not} \end{cases}$$

Y_{ij} = fraction of demand at node *i* that is served by a facility at node *j*

W = maximum distance between a demand node and the nearest facility

With this notation the *p-centre* problem can be formulated as follows:

MINIMISE W (1*a*)

SUBJECT TO:

$$\sum_j Y_{ij} = 1 \qquad \forall i \qquad (1b)$$

$$\sum_j X_j = p \qquad (1c)$$

$$Y_{ij} \le X_j \qquad \forall i, j \qquad (1d)$$

$$W \ge \sum_j d_{ij} Y_{ij} \qquad \forall i \qquad (1e)$$

$$X_j = 0, 1 \qquad \forall j \qquad (1f)$$

$$Y_{ij} \ge 0 \qquad \forall i, j \qquad (1g)$$

The objective function (1*a*) minimises the maximum distance between a demand node and the closest facility to the node. Constraints (1*b*) requires each demand node *i* to be assigned to exactly one facility *j*. Constraints (1*c*) stipulates that *p* facilities be located. Constraints (1*d*) state that demands at node *i* cannot be assigned to a facility at node *j* unless a facility located at node *j*. Constraints (1*e*) state that the maximum distance between a demand node and the nearest facility to the node *W* must be greater than or equal to the distance between any demand node *i* and the facility *j* to which it is assigned.

For those cases in which we want to consider, the demand-weighted distance, constraint (1*e*) can be replaced by

$$W \geq h_i \sum_j d_{ij} Y_{ij} \qquad \forall i \qquad (1e')$$

For the general graph and for variable values of *p*, the *p-centre* problem is *NP*-complete (see [Kariv 79] and [Garey 79]). Conversely, when the number of facilities, *p* is fixed, the vertex *p-centre* problem may be solved in polynomial time. In fact, for a network with *N* nodes, we need only evaluate each of the $O\binom{N}{p} = O(N^p)$ possible combinations of *p* facility sites.

2.3.4 Formulation of the *p-median* Problem

In this section, we specify the *vertex p-median problem* or *minisum problem* as formulated in [Daskin 95] (pp.200-203). This is to find the location of *p* facilities on a network so that the total cost is minimised. The cost of serving demands at node *i* is given by the product of the demand at node *i* and the distance between demand node *i* and the nearest facility to node *i*. This problem may be formulated using the following notation:

Inputs

d_{ij} = distance from demand node *i* to candidate facility *j*

h_i = demand at node *i*

p = number of facilities to locate

Decision Variables

$$X_j = \begin{cases} 1 & \text{if we locate at candidate site } j \\ 0 & \text{if not} \end{cases}$$

$$Y_{ij} = \begin{cases} 1 & \text{if demands at node } i \text{ are served by a facility at node } j \\ 0 & \text{if not} \end{cases}$$

With this notation the *p-median* problem can be formulated as follows:

MINIMISE $$\sum_i \sum_j h_i d_{ij} Y_{ij} \qquad (2a)$$

SUBJECT TO: $$\sum_j Y_{ij} = 1 \qquad \forall i \qquad (2b)$$

$$\sum_j X_j = p \qquad (2c)$$

$$Y_{ij} - X_j \leq 0 \qquad \forall i, j \qquad (2d)$$

$$X_j = 0, 1 \qquad \forall j \qquad (2e)$$

$$Y_{ij} = 0, 1 \qquad \forall i, j \qquad (2f)$$

The objective function (2*a*) minimises the total demand-weighted distance between each demand node and the nearest facility. Constraint (2*b*) requires each demand node *i* to be assigned to exactly one facility *j*. Constraint (2*c*) states that exactly *p* facilities are to be located. Constraint (2*d*) link the location variables X_j and the allocation variables Y_{ij}. They state that demands at node *i* can only be assigned to a facility at location *j* (Y_{ij} = 1) if a facility is located at node *j* (X_j = 1). Constraint (2*e*) and (2*f*) are the standard integrality conditions.

For fixed *p*, the *p-median* problem may be solved in polynomial time. Daskin observed that, despite the fact that the problem is O(N^p), the number of possible solutions that must be enumerated tends to become exceptionally large in practice (see [Daskin 95] p. 203). In fact, the time required to solve all *p-median* problems by enumeration for *p*=1 to *p*=*N* for any given value of *N* is

$$\sum_{j=1}^{N} \binom{N}{j} = 2^N - 1 = O\left(2^N\right)$$

which is exponential in *N*. Hence, alternative algorithms, *e.g.* based on heuristics, need to be found in practice to solve problems of realistic size in a reasonable amount of time.

2.3.5 Medi-centre Problems

Medi-centre problems combine the *minisum* objective of the median formulation with the *minimax* objective of the centre formulation. There are various ways of combining median and centre problems to meet the requirement of practical applications. While the concept of mixed median-centre formulations was indicated in [Hakimi 64], relatively little effort has been devoted to computational methods for such models, probably because the ‘pure’ problems were not well solved until quite recently. Handler gives various formulations of medi-centre problems in [Handler 79]. In this thesis we do not tackle medi-centre problems and focus on the solution of the agent location problems formulating it as a *p-median* problem.

2.3.6 Formulation of the Agent Location Problem

In general it would be preferable to formulate the agent location problem as a medi-centre problem. However, because the main requirement is to find a solution which can be computed efficiently, the agent location problem has been studied from the particular angle of a *p-median* problem which aims at minimising total incurred traffic rather than response time. In practice, reduction in traffic tends to have a positive impact on response time too. Hence the approach has been to find a suitable approximate solution to the *p-median* problem and study the effects on both total traffic and response time.

Therefore, the agent location problem specified as a *p-median* problem is formulated as in the above Section 2.3.4. In addition, for the solution to be viable, we also need to be able to give upper-bounds on the total computational time. This is needed because only those monitoring tasks whose duration is reasonably longer than the span of time needed to compute their location are beneficially implemented with MAs.

As such, the agent location problem is an *NP*-complete problem in general. This thesis provides an efficient, provably near-optimal solution to it.

2.4 Conclusions

This chapter introduces the main research areas touched by this interdisciplinary thesis in the form of a brief tutorial. It also provides the glue between those areas and drives the reader through some open issues that arise when concepts from one field are brought into another. The chapter serves not only as a tutorial but also to identify important research gaps and highlight the ones which are tackled in the thesis, providing motivation for the thesis work.

The thesis marries the perspectives of strongly distributed hierarchical management and those of autonomous, mobile software agents. The proposed active, distributed monitoring approach falls under the umbrella of vertical delegation of management responsibility whereby a main management station distributes a given monitoring task over one or more MAs. The use of MAs allows for dynamic, spatial, and temporal distribution. Agent migration and cloning are triggered by network or system events, following the event-driven delegation concept.

An important gap addressed by this thesis is the attempt to build a monitoring system targeted for large-scale, dynamic systems. Code mobility offers a powerful means to pursue that but also poses a number of problems. Focal point of the thesis is the investigation of efficient algorithms to solve the agent location problem. This is an *NP*-complete problem when striving for optimality. Approximate solutions exist but do not suit the requirements of the agent location problem. This aspect is further elaborated in Chapter 3 which complements Chapter 2 with a more detailed break-down on the work related to the thesis.

Chapter 3

Related Work

This chapter complements the previous one by focusing more closely on the review of work related to the thesis. As already stated, this thesis investigates solutions for scalable, distributed, and adaptable monitoring based on MAs in which the real essence of MAs is exploited. To the best of the author's knowledge, in the literature, there is not sufficient work targeting all of those properties. This chapter reviews work that has pursued some of the following targets (not necessarily all of them):

a) *Application of MAs to fundamental functions of management*. This thesis is focused on monitoring, whereas other investigators have studied MAs in contexts such as fault, configuration, or performance management.

b) *Exploitation of the real essence of MAs*. In addition to 'mobility' other important features of MAs include 'cloning', 'autonomy', 'reactivity', and 'pro-activity'.

c) *MAs as a means for dynamic delegation of management functionality*. MAs encapsulate management logic delegated by another management entity. This delegated logic actually travels with the MA itself.

d) *Quantitative performance evaluation of non-functional properties of MA-based management systems*. The study of performance and scalability of the system in comparison with weakly distributed hierarchical paradigms is fundamental. In MA approaches part of the control of the system is encapsulated in the agents themselves. Migration triggered by autonomous decisions may affect the stability of the system, another important non-functional property. Non-functional properties can be assessed by simulation, mathematical modelling or prototype-based experimentation.

e) *Viable solutions to the agent location problem.* We have elaborated on the need for efficient solutions to this problem in Chapter 2.

The above targets are also key requirements for the proposed agent system. Therefore, the reviewing of work that addresses these requirements will lead to with a clearer view of which literature gaps are addressed by the thesis.

3.1 Management by Delegation

A seminal work towards *strongly distributed hierarchical management* is the one introduced by Yemini *et al* with their Management by Delegation (MbD) framework [Yemini 91]. MbD also represents one of the first concrete attempts to make use of Mobile Code in Network Management. Its inventors claim that MbD is not only a technique that allows for dynamic decentralisation and automation of management functions, but it also introduces a paradigm shift in the management arena [Goldszmidt 98].

The basic underlying principle of MbD is that management processing functions can be delegated dynamically to the network elements and executed locally rather than centrally. Thus, instead of moving raw data from the elements to a central management application, the application itself is moved and executed at the elements where data actually resides. An MbD platform is implemented as a set of *elastic servers* residing in the network elements. An elastic server is a multithreaded process whose *program code* and *process state* can be modified, extended and/or contracted during its execution [Goldszmidt 93]. *Server elasticity* contrasts with the traditional *client-server* paradigm, which does not provide support for such a dynamic transfer of functionality between client and server. Thus elastic servers introduce a new paradigm of interaction between components in distributed applications and provide a powerful mechanism to dynamically compose distributed applications by connecting and integrating independently delegated programs.

A manager can dynamically dispatch *delegated agents* to remote elastic servers using a *delegation protocol* and can then control their execution. This protocol includes service primitives to delegate, instantiate, suspend, resume, abort and remove delegated programs. Thus, by using this protocol, a manager application can augment, during execution time, the functionality of a subordinate element, allowing it to perform an open-ended set of management programs. Delegated agents can monitor, analyse, and control devices independently from the manager, except where explicit co-ordination is required.

MbD supports *dynamic delegation* [Mountzia 96] and provides mechanisms for both *spatial* and *temporal distribution* [Goldszmidt 96c]. Several applications have been prototyped on this MbD platform [Goldszmidt 96b], showing the advantages, and some of the fields of applicability for the delegation framework. For instance, a scenario of ATM switch management is reported in [Goldszmidt 96c]. Another example is an application for real-time monitoring of the *health* of large-scale networks [Goldszmidt 95b] and for the management of *stressed* networks [Meyer 95]. Further, some considerations describing how MbD can reduce the *management control loop* and thus make networks more *reliable*, are reported in [Meyer 95, Goldszmidt 96b]. More generally, a class of applications which might benefit from MbD is described in [Meyer 95, Goldszmidt 96b]. Therefore, in this MbD framework, delegated scripts can operate independently from the manager, and on behalf of it, whilst managers are concerned with the management of delegated scripts. Thus, this platform is a feasible base-framework for autonomous, hierarchical, and delegated management.

3.1.1 Evolution of the Original MbD Work

MbD has been used and further elaborated by many researchers. An application framework for application management and system administration has been developed at INRS-Telecommunications (Canada) [Gagnon 93, Gregoire 93a, Gregoire 93b, Gregoire 95]. Delegation is used to reduce the impact of management on applications and communications. Particular emphasis is put on management applications rather than on network elements. A manager can develop, deploy and co-ordinate delegated programs to different network elements; and can, then, monitor their data logs. A thread management module provides an environment to develop management programs through a formal specification language and then to transform them into a corresponding implementation. The system supports not only mechanisms to allow access of resources – *e.g.*, reading, retrieving and configuring – and delegation of functionality but also *recursive delegation*, called *worms*. Worms can be useful for diagnostics, configuration and accounting [Gregoire 95], for instance, tracing the symptoms of a fault across different machines using recursive delegation in search of the root of a problem.

Another interesting aspect of this delegation environment is the fact that it is built around the CAML language and the CAML-light Virtual Machine [Gagnon 93]. This language has an important programming construct, known as continuation, which allows the execution of a function within a previously saved context. Moreover, the authors have enhanced the virtual machine to support multiple contexts of execution – *i.e.* multiple threads. Thus, new threads can be developed independently and dynamically added to the set of running threads. This platform was prototyped to remotely manage a UNIX, distributed and heterogeneous system. SNMP access mechanisms were integrated. The worm functionality – *i.e.*, recursive delegation – was used to track users, implement load balancing and experiment with several distributed algorithms [Gregoire 95].

However, despite the valuable work, many of its aspects would need further research. No quantitative results showing the performance benefits of this platform over traditional management platforms have been reported. Furthermore, no conclusions can be inferred about the scalability of the platform because this system was only applied to the administration of UNIX systems over relatively small-scale networks. Besides, this framework seems to be suitable for fine-grain management functions. In fact, relatively simple management tasks that could be implemented with simple threads were specified. In other words, the system is not meant to be used to delegate complex management application modules but only simple tasks. Finally, although some of the operations for dynamic administration have been implemented in order to show the flexibility and the power of delegation, many aspects of administration have not been covered yet [Gregoire 93a] – *e.g.*, the application to lower-level management should be investigated. A limitation of this platform is that it is not language-independent. Also, threads have to be compiled in the manager before being sent to their respective virtual machine for execution; this could represent a processing bottleneck in the manager side.

Despite the similarities with the work developed at Columbia University, the INRS framework shows some differences. The former is more focused on the infrastructure allowing delegation and extension of server functionality. The latter is more concerned with the formal specification and verification of delegated tasks at the application level. In the former case, scripts are delegated to the server in order to extend its capabilities and to provide new services to be performed locally. In the latter only simple management functions are compiled in the manager and dispatched as threads to their respective virtual machines for execution. Thus, although the delegation concepts are very similar, the actual implementation approaches and level of management granularity are quite different. Therefore, these works can be considered as complementary.

Many other researchers have used and elaborated MbD. Steenekamp and Roos have proposed a framework that extends the delegated agent model to provide agents that implement system management policies in terms of rules [Steenekamp 96]. Suzuki *et al* propose improving the efficiency of delegated management operations by using a script binding mechanism that divides an operation script into a *script skeleton* and *operational objects* [Suzuki 96]. Schonwalder presents some co-ordination primitives for co-operation among management agents based on the MbD paradigm [Schonwalder 96]. Keller analyses the suitability of

Common Object Request Broker Architecture (CORBA) for implementing MbD [Keller 96]. Mounzia has proposed a distributed MbD approach based on flexible agents [Mountzia 98].

A scenario showing the utility of delegation to increase security in distributed systems is described in [Meyer 95] where the management of a distributed intrusion detection system is presented. In this case a centralised management approach would require the collection of a relatively large amount of audit data – typically in the order of 10 megabytes per hour and per each audited machine and increasing linearly with the number of users – requiring a large amount of network capacity and processing power. Management decentralisation leads to faster reactivity to intrusion, which in turn may result in better security.

An application of MbD to an ISDN-based remote-access environment is described in [Payer 97]. Finally, a classification scheme including delegation types, phases, functional areas, trigger modes and lifetimes of delegated tasks is reported in [Mountzia 96]. Requirements and technologies for MbD are described in [Mountzia 97b, Mountzia 96]; and a comparative review of decentralised network management techniques is reported in [Kahani 97, Martin-Flatin 97a, Martin-Flatin 97b].

3.1.2 Agent Constrained Mobility

The author of this thesis has carried out some collaborative work to study MbD from the particular perspective of MAs. This is reported here rather than in one of the experimental chapters because this work does not meet requirement b) (page 42) by not considering the application of *multi-hop* MAs to the field of management.

In [Bohoris 00b] we have looked at various models of agent mobility in the particular context of network performance monitoring. We have introduced the concept of *constrained mobility* and discussed its practical use for dynamically programming network elements. In essence, agent constrained mobility is achieved when MA technologies are used in a constrained fashion; that is by limiting the multi-hop migration ability of the agent according to a *single-hop* migration scheme. The agent migrates from the management station (where it is created) to a remote machine, where it executes a task and terminates upon completion. This concept is derived from MbD with the difference that, in our case, what is shipped from source to destination is an autonomous mobile software agent; whereas an MbD *delegated agent* may be seen as a component aimed at dynamically enhancing the *elastic server* capability.

In our performance monitoring system, MAs are created at the network management level according to the user requests and then migrate to network elements to perform monitoring functions in a local manner. The behaviour of the monitoring algorithms can be customised, enabling dynamic programmable functionality to be provided directly in the managed network elements.

The following is extracted from [Bohoris 00c] and summarises the outcomes of our investigation on constrained mobility. Details on the method used to carry out the experimental work are not repeated herein. The performance monitoring case study described in [Bohoris 00a] has been implemented over four different infrastructures to carry out a performance comparative analysis. These were Java-RMI, CORBA, Code-Shell and Grasshopper. The case study consisted essentially in providing traffic rates with thresholds, quality of service alarms and periodic summarisation reports by simply observing raw information such as traffic counters in network elements. This is a functionality similar to the OSI-SM metric monitoring and summarisation facilities [X739 93, X738 93] but when it is provided through code mobility, users of the service are also able to customize it according to the semantics of a particular application as explained in [Bohoris 00a].

Grasshopper is not one of the most efficient MA platforms. It has been chosen because, due to its functionality, it can be considered a general-purpose MA platform. Moreover, Grasshopper follows the current standardization directions, since it is compliant with both the Mobile Agent System Interoperation Facility Specification [MASIF 97] and the Foundation for Intelligent Physical Agents [FIPA].

Java-RMI and CORBA have been chosen as representative of the most popular ‘static’ distributed object technologies. In addition, CORBA is emerging as a significant technology for network and service management.

Finally, Code-Shell is a code mobility platform optimised for constrained mobility developed at the University of Surrey, UK. This platform was prototyped with the objective of achieving performance comparable to the one obtained with static distributed object technologies.

When constrained code mobility is deployed using either Code-Shell or Grasshopper, a performance monitor object is created by a “master” object somewhere and is sent to execute within a target node. When Java-RMI and CORBA are used, the performance monitor object is created at the target node through an object factory and the relevant intelligence needs to pre-exist at that node. The performance monitor gathers information locally, applies thresholds and sends QoS alarms or periodic summarisation reports to the master object.

The agent creation and migration overheads, the cost of remote invocation that models the reporting of results, both in terms of response time and packet sizes, and the computing requirements at the target node were measured. In the cases of Grasshopper and Code-Shell, measurements have been taken at steady state, that is after the code had been shipped to the remote elements. In this way, it was possible to perform a direct comparison with the implementations based on Java-RMI and CORBA, respectively. The additional overheads incurred by code mobility – namely code deployment time and network traffic incurred during the transmission of the code from manager to network elements – were measured separately.

3.1.2.1 Response Time Measurements

The response time of management operations for each of the four cases of performance monitoring systems are reported in Figure 3-1 which, for an easier comparison, combines in a single chart the mean values and best linear fit of the results.

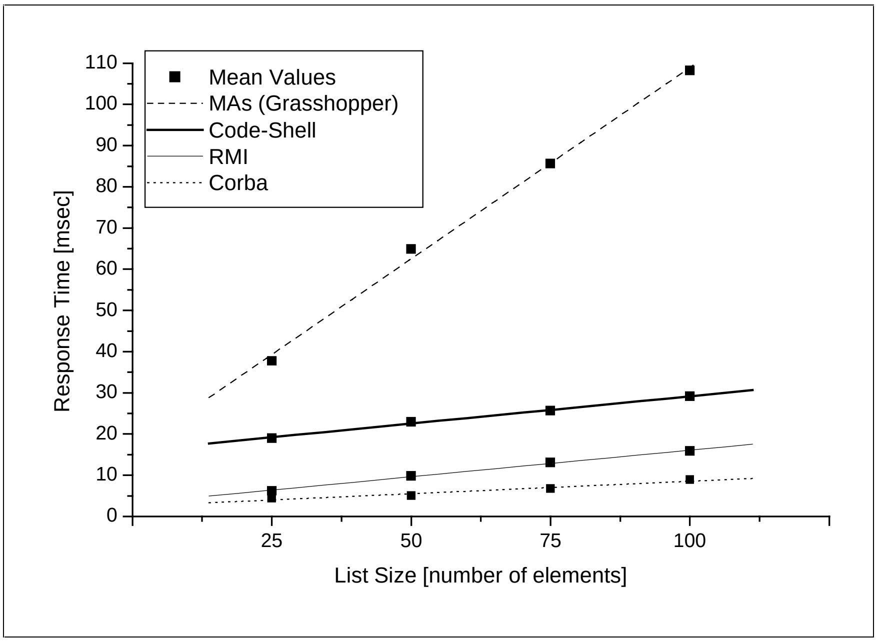


Figure 3-1. Mean values and best linear fit of response times.

The first conclusion that can be drawn by observing the plots is that the system based on Code-Shell exhibits the same degree of scalability as the one of the systems based on Java-RMI and CORBA. In fact, the slopes of the curves of those three cases have a comparable value. On the contrary, the Grasshopper system exhibited a much bigger slope showing its intrinsic inability to perform well under more demanding conditions.

From the performance point of view the Code-Shell system gave a response time in the order of 2-3 times larger than the one of the Java-RMI system and in the order of 4 times larger than the one of the CORBA system.

3.1.2.2 Traffic Measurements

The packet sizes in all four cases were also measured. An array of objects containing 25, 50, 75 and 100 "*Double*" numbers respectively was remotely sent using remote invocations in the

Mobile Agent, Code-Shell, RMI and CORBA systems. Each time, the payload of the TCP packets was measured. A chart of the results gathered can be seen in Figure 3-2.

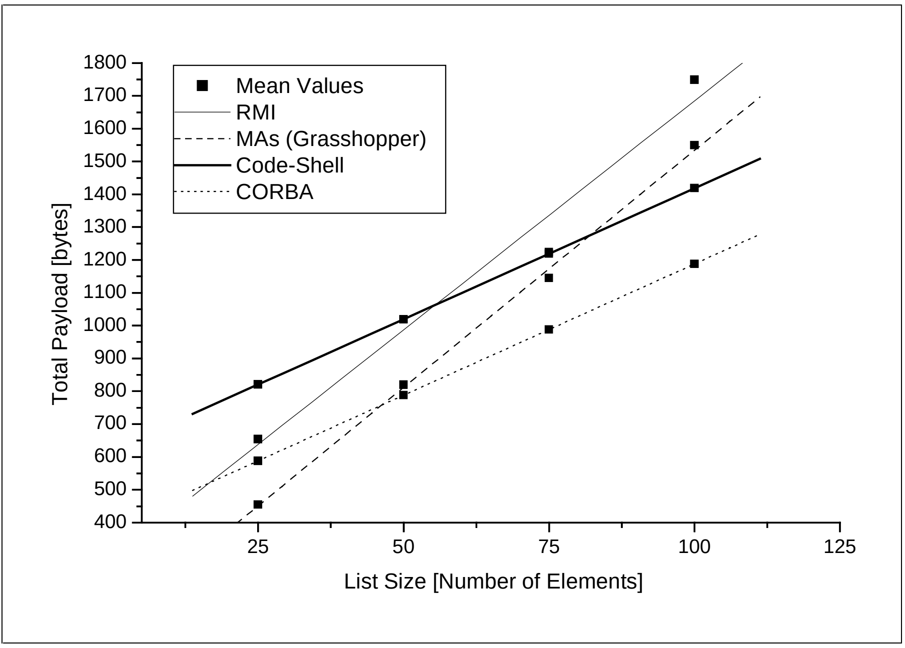


Figure 3-2. Mean and best linear fit of total incurred TCP payloads, measured as the sum of all the bytes incurred in the network to complete the given network performance monitoring task.

It is interesting to observe that, within the measured range of values, the four solutions incurred a comparable level of traffic. The Grasshopper and RMI systems performed better for small scales whilst the Code-Shell and CORBA systems exhibited better performance for larger scales. The Code-Shell platform transparently optimises the transfer procedure and this is the reason why, for a large number of elements, it incurred less traffic in the network compared with the standard RMI system.

3.1.2.3 Memory Measurements

The memory requirements for the monitoring systems based on Grasshopper, Java-RMI, and Code-Shell, are compared in Figure 3-3.

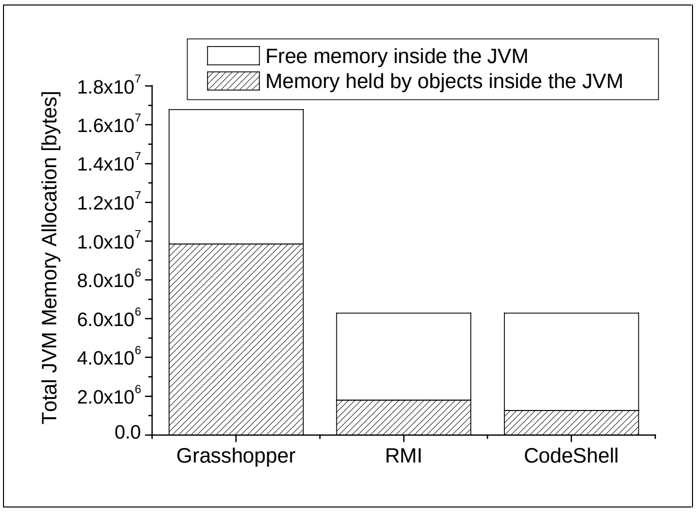


Figure 3-3. Memory requirements for the Java-based network performance monitoring systems.

It can be observed that Code-Shell performs as well as Java-RMI and significantly better than Grasshopper. The latter, resulted in a fivefold occupation of memory which is yet another dramatic drawback of relying on a general-purpose MA platform.

3.1.2.4 Code Migration Overheads

The delay and traffic involved during code migration have been measured for the two programmable network performance monitoring systems based on Grasshopper and Code-Shell, respectively. By substituting the generic code migration mechanism of Grasshopper with a simpler code deployment protocol the time required to program a network element was reduced by more than 4 times (from 1500 msec to 350 msec).

The additional traffic incurred by code migration was also measured. The transmitted data for the Code-Shell system was 2,236 bytes; for the Grasshopper system it was 2,854 bytes.

3.1.2.5 Remarks on Constrained Mobility

Constrained mobility can be the vehicle to realise network programmability and MbD functionality with current technologies. The implementation of Code-Shell has demonstrated that it is possible to realise management systems based on the constrained mobility concept achieving at the same time performance levels comparable to the ones of systems based on the most popular static distributed object technologies. We also aimed at quantifying the performance gain achievable by optimising general-purpose MA platforms, retaining only the most basic form of code mobility exemplified by constrained agent mobility. The results presented more extensively in [Bohoris 00b, Bohoris 00c] suggest that constrained mobility is easily integrated in network management systems. Moreover, using constrained mobility it is still possible to achieve performance and scalability typical of static distributed object technologies with the additional advantage of dynamic programmability. In addition to performance monitoring it is believed that other functions such as configuration and fault management can similarly benefit from constrained mobility but this is subject of future investigation.

Finally, constrained mobility is a particularly suitable model for tasks requiring a relatively long period of time to complete and in those cases where information intended for off-line analysis is collected by the agent in the remote machine.

3.1.3 MbD in the Context of Internet Management

Another example of decentralised management through delegation is described in [Kooijman 95]. In this system a central manager can delegate monitoring tasks to *area agents*. Tasks are not pre-defined and can be instantiated and uploaded to agents, at any time, as scripts. The important aspect of this work is the application of event-driven management to delegation. Scripts are only executed if certain events are generated. Otherwise the scripts are set in sleeping mode, *i.e.* no processing resources are used. Events can be triggered either by external requests – *e.g.* remote controlling network management systems – or by the core system of the area agent. The operative environment for downloadable scripts consists of an extension of the RMON MIB [Waldbusser 91]. One of the limitations of this delegation framework is that this is targeted mainly for monitoring operations and no other management functions are considered. Also it is not clear whether area agents can be added dynamically and whether a hierarchy of agents can be defined in a flexible fashion – *i.e.* allowing more than two levels of managers. Finally, no quantitative performance analysis and no information about the scalability of the platform have been reported.

An extension of SNMPv2 providing elementary delegation facility is presented in [Siegl 95]. A hierarchy of SubManagers, acting as intermediary between the Manager and the Devices can be defined here. A Network Management Procedure (NMP) is represented in three tables stored in the SubManager. One table contains general information about the procedure - *e.g.* the polling interval. The second one is used to store the program (each statement is presented in a separate row). Results and timestamps are stored in the third table. Programs are written in low-level network-oriented script language. A manager can load an NMP and ship it to a SubManager. The procedure is then executed by forking a worker process inside the SubManager. Results are stored in the appropriate table and can be interpreted by any manager capable of SNMPv2.

The advantage of this platform is that it supports dynamic configuration of SubManagers – *i.e.* run-time delegation of functionality – and hierarchical management. Also it can be easily integrated into existing network management systems as it presents interfaces to both managers and agents conforming to SNMPv1 and SNMPv2. However, a performance analysis of the script execution environment has not been reported. The efficiency of the adopted language was not compared with any other interpreted approach. Also the number of NPMs that can run simultaneously is limited as the execution of each script involves the forking of a new process. Therefore, this approach is more resource consuming that others which implement a management task as a thread instead of a UNIX process. Another limitation is that no asynchronous report mechanisms were implemented, polling being the only means to retrieve data.

More recently MbD has been looked at by standards bodies, such as IETF and ISO/ITU (the latter will be discussed in the next section). Levi and Schonwalder have been working on the integration of the MbD model in the IETF management framework since 1996 [Levi 96]. The Distributed Management (DISMAN) working group of the IETF was chartered to define an architecture where a main manager can delegate control to several distributed management stations. The DISMAN framework provides mechanisms for distributing scripts, which perform arbitrary management tasks to remote devices. To the DISMAN group, distributed management does not mean management functionality distributed in a statically set way, but something that is ‘movable’. The objective is to allow the ‘distributed management’ application to keep pace (scalability) with the changing needs of large distributed system.

Proposals for several related Management Information Bases (MIBs) already exist. Among these are DISMAN-SERVICES-MIB, TARGET-MIB (to express targets for traps and script transfer), EVENT-MIB (based on the RMON alarm and event groups), Notification LOG-MIB, Expression MIB, Schedule MIB (definition of MOs for scheduling management operations), Remote Operations MIB, and Script-MIB. In particular, Script-MIB defines a standard interface for the delegation of management functions based on the Internet management framework [Schonwalder 97, Schonwalder 99, Levi 99, Schonwalder 00]. This comprises capabilities to transfer management scripts; for initiating, suspending, resuming, and terminating scripts; to transfer arguments and results; and to monitor and control running scripts.

These proposals are not yet standardised but signal the first steps the Internet has taken towards implementing management as a dynamically distributed application. However, it is not yet clear whether implementation costs will have an effect on the acceptance of the powerful DISMAN concept. Currently, there is only a limited number of Script MIB implementations. Initial evaluations have been provided based on the Jasmin platform [Schonwalder 00] but this technology has not yet been thoroughly tested and evaluated. Another limitation of this approach is that it is not generic and framework-independent, being specific to the Internet Management framework.

Finally, an enhancement of DISMAN based on MAs, called Mobile DISMAN, has been proposed by Oliveira and Lopes [Oliveira 99, Lopes 00]. More recently, Quittek and Brunner have realised the MbD concept over three different frameworks, Script MIB, Voyager, and ABone [Quittek 01]. Voyager is a commercial MA platform, whereas ABone stands for Active Networks Backbone and is a network of hosts running the same active networks technology. The authors elaborate on the differences among the three approaches in terms of performance, functionality, manageability, security, robustness, resource consumption, and extensibility. As expected, Script MIB offers all the required MbD functionality whilst resulting in best performance.

3.1.4 MbD in the Context of OSI Management

MbD has attracted interest in the OSI management community, though significantly less than in the Internet community. Perhaps the reasons for that originate in the different philosophy of OSI management, which is more complex and relies on a high degree of standardisation.

An early study towards the realisation of MbD in the context of OSI management has been carried out by Vassila *et al* [Vassila 95]. Upon discussing the need for programmable management facilities within the TMN environment, the authors propose the concept of Active Managed Objects (AMOs) as opposed to the standardised, static Managed Objects (MOs). AMOs offer the means to specify and express arbitrary management functions along with a mechanism to dispatch and control management scripts to the Network Element (NE) using existing TMN mechanisms. AMOs may be delegated to a TMN application in the agent role and function close to other MOs they access.

Two years later, the authors describe a pilot implementation of AMOs [Vassila 97] within the OSIMIS TMN platform [Pavlou 95]. They present two APIs which need to be available to program scripts; one provides access to the local environment in which the script is embedded; the other provides access to MOs in local and remote systems. The control of the script execution is effected by representing the script and its execution as an MO.

Since AMOs use the normal TMN mechanisms for information modelling and access (GDMO and CMIS/P), they might have been standardised. This never happened; however, a similar approach was pursued by the ISO/ITU that has designed the CMIP command sequencer [X753 97, ISO-10164 98]. Again, management functionality are delegated as scripts which are transferred to the location where they are to be executed. These scripts can be started remotely, arguments can be passed to them, and results can be returned to the initiator. The scripts can communicate in agent role with the delegating manager.

3.1.5 Remarks on MbD

The powerful concepts introduced by the MbD work are the germs which have led to the MA work of this thesis. In other words, MbD is extremely relevant for historical reasons, having paved the way of dynamic, distributed management (strongly distributed management paradigm) as opposed to more traditional centralised or weakly distributed management approaches.

The MA work that has followed MbD has been built on the ideas laid out by MbD by adding an extra degree of freedom to distributed management. With MAs the distribution mechanism is not bound to a one-hop delegation mechanism. The MA paradigm is more powerful than that, allowing for multiple-hop migration not only at delegation time but also during the execution of the delegated task.

In addition, the agent concepts bring a new set of instruments and provide a more flexible framework for the design of even more powerful strongly distributed management systems. Other researchers have tried to go beyond MbD exploiting essential features of MAs such as cloning, autonomy, reactivity, and pro-activity. Their work is reviewed in the next section.

3.2 Management based on the Mobile Agent Paradigm

Grasping the state-of-the-art of MA-based management is not an easy task because, since the mid nineties, several researchers have attempted to study and exploit the MA paradigm aiming at scalability and flexibility. However, if we take a closer look at the work produced, most MA studies degenerated into MbD work, whilst agent mobility was mostly used as a general mechanism for remote programming. The multi-hop capability offered by MA platforms was not exploited in most cases. Another class of work degenerated even to a larger extent by making use of ‘stationary’ agents. This is the case of most multi-agent approaches in which inter-agent co-operation is key to increase the intelligence and adaptability of the management system.

Both MbD and multi-agent approaches are possible ways of pursuing increased scalability and flexibility in management. On the other hand, in this section we concentrate on approaches in which agent mobility is the key feature. Our interest is to survey previous work that attempts to exploit the multi-hop capability of MAs, similarly to the approach proposed in this thesis work.

Bieszczad and Pagurek have led extensive work on MA-based management during the last few years. In a recent article they describe representative applications of MAs to network management [Bieszczad 98c]. They identify possible applications belonging to each of the five OSI functional areas – *i.e.*, fault, accounting, configuration, performance, and security management. However, they describe only example applications to fault, configuration, and performance management.

In the following sub-section we take a similar approach but describe a larger range of representative applications which is not bound to network management. We do not survey MA-based accounting and security management because very little work has been described in the literature. In addition, these areas are not central to the thesis.

3.2.1 MA-based Fault Management

Network fault management refers to efforts made to identify, trace, and solve network problems. This typically involves the following activities: 1) collecting messages from network components; 2) generating alarms by filtering messages; 3) diagnosing faults that caused the messages by correlating the alarms; 4) correcting faults; and 5) verifying that the network problem is eliminated. El-Darieby and Bieszczad have presented a number of possible ways for exploiting agent mobility to automate most of the above activities [El-Darieby 98]. They have implemented proof-of-concept applications in which agents roam the network, accessing information from each node, processing this information at each node locally, and carrying the results of this processing during the migration. Their agents can filter alarms, eliminate multiple occurrences of the same alarms, correlate the remaining alarms, and trigger auto-migration depending on the processing results. They can also perform auto-diagnosis by attempting to determine the causes of the generated alarms and whether those alarms are false.

The experimentation carried out by El-Darieby and Bieszczad identifies a number of conditions which result in increased efficiency in comparison to centralised polling-based fault detection. Agent migration is triggered primarily by the status of the network. Hence agent migration overheads increase with fault rate. On the other hand, centralised polling-based fault management incurs the same overheads regardless of the network status.

A PhD thesis has highlighted some of the issues related to MA-based fault management [Grimes 96]. The author presents various scenarios in which agent mobility is particularly advantageous, though these are described as future work rather than being the subject of experimentation. A potential application is the tackling of routing problems in IP networks such as permanent or intermittent link failures and routing loops. For instance, routing loop detection may be a periodic check or a part of the testing repertoire of the MA as it migrates along a problematic route. Route testing in a route with multiple paths is another example. In this case, in order to check each possible path between source and destination, an MA which forks itself at branches may be employed to reduce test duration.

A more recent work addresses the additional problem of modelling managed resources for the purpose of diagnostic testing and performance monitoring [Guiagoussou 01]. The authors propose additions to the Java Management Extension API (JMX) specific to fault management termed Java Fault Management Extension (JFMX). They present a diagnostic test client/server scenario and discuss various implementations in order to highlight the performance advantages introduced by agent mobility. They introduce roaming agents with enough skills to be able to autonomously decide ‘when’, ‘where’, and ‘how’ to conduct test results, alarm filtering and correlation.

3.2.2 MA-based Configuration Management

According to ISO, configuration management controls the configuration state of a system or network and the relationships between components. It also initialises, configures, and shuts down network equipment.

In the context of configuration management, MAs may be used for two different functions: service provisioning and component provisioning [El-Darieby 98]. In the former case, MAs may be used to streamline complex configuration processes involving several parties in a multi-vendor, heterogeneous environment.

Pagurek *et al* have illustrated some of the opportunities for using MAs in the context of Permanent Virtual Circuit provisioning in ATM networks [Pagurek 98a, Pagurek 98b]. In their scenarios, MAs are used to negotiate all aspects of a PVC that need to be established between two ATM switches manufactured by different vendors. If a PVC includes a path through different operating companies, offering similar services with different prices, an agent can be exploited to perform service negotiation, a task typically carried out manually by a human operator. Agents can also hide network heterogeneity, allowing for automatic PVC setting up.

In the context of component provisioning, configuring a device requires that a large number of attributes in the network and on the device be set up as well as certain software components be installed. MAs can be used to automate the necessary configuration by deploying plug-and-play network components. Raza has illustrated this opportunity in the context of network printer installation [Raza 98]. Agents are used to discover the network devices that will need printer drivers, find those drivers in the Web and co-ordinate their installation.

3.2.3 MA-based QoS Management

The concept of Quality of Service (QoS) has gained significant importance with the introduction of an increasing number of multimedia services in data telecommunication networks. QoS requirements are usually expressed by *QoS parameters* such as data error rate, packet loss rate, throughput, end-to-end delay and delay jitter. *QoS negotiation* is an important functionality of the connection set-up procedure, very important for effective *QoS admission control*. *QoS contracts* (or service level agreements) are stipulated between users and providers. The provider needs to employ suitable management policies in order to comply with those contracts. *QoS monitoring* is another important function. Finally, *QoS adaptation* is an important feature in view of the dynamic behaviour of current networked systems.

QoS functionalities are inherently distributed since they involve allocation, monitoring, and control of distributed network resources. Some researchers have investigated the employment of the MA paradigm in the context of QoS management, due to the dynamic nature of this operation.

De Meer *et al* have proposed an architecture supporting MA-based QoS management, along with simple application scenarios [deMeer 98a]. QoS is pursued by monitoring fix-rate traffic and distributing equally the rest of the capacity for best-effort traffic. MAs are used to control the bit-rate of the applications relying on best-effort traffic, and to decentralise this control logic. The value of this initial work consists in the introduction of the MA paradigm to tackle a problem that is becoming increasingly critical in the context of IP-based multimedia networking.

The same authors take a more pragmatic approach in a second paper in which they propose a method to provide guaranteed real-time services with the Resource reSerVation Protocol (RSVP) in the presence of non-RSVP compliant routers [deMeer 98b]. MAs are dynamically sent at the edges of non-RSVP clouds to create QoS tunnels across those clouds. Agents closely monitor the tunnel and generate alarms in the case of QoS violations, allowing for compensating measures. Agent mobility is exploited to re-tunnel traffic exceeding a given threshold.

Some experimental results are presented in a more recent work by De Meer *et al* [deMeer 00]. The authors focus on QoS monitoring and realise a QoS management strategy with MAs that can respond to variations of user requirements and network state. They elaborate on the benefits of using IntServ/RSVP to realise QoS schemes *a-la* ATM. Agents are used to discover about resources available inside the network and claim those resources on behalf of the user. Agent can also trigger adaptation of applications inside the network on behalf of the user. This is performed based on sharing of excess resources. RSVP signalling is used by the agents for resource reservation, admission control, and QoS adaptation control. Initial experimental results demonstrate the potentiality of QoS management complemented by MAs.

Anastasi *et al* stress the use of MAs to a larger extent, for QoS provisioning in the context of integrated fixed and mobile IP networks [Anastasi 00]. They complement De Meer's work with support for user and terminal mobility with both stationary and mobile agents. QoS provisioning in a mobile environment poses additional problems. Dynamic QoS adaptation is a critical problem because wireless networks are characterised by lower bandwidth and larger packet loss ratio, which may vary abruptly over time due geographic impairments, meteorological conditions, etc. The authors describe various possible uses for MAs but do not report any experimental results.

A completely different approach if proposed by Guedes *et al* [Guedes 98]. The authors start from the assumption that QoS negotiation is only simple if the resources are managed by a single entity and by a set of entities supporting common negotiation protocols. Instead, in distributed multimedia systems, negotiation and management of resources are complex tasks since resources are diversified, distributed, and managed by autonomous entities. The proposed approach does not impose any restriction on the distributed multimedia system or to the underlying network. The agents responsible for QoS negotiation provide end-to-end control for each flow. Hence only flow sources and sinks are directly involved in QoS negotiation and management. In other words, the proposed agent architecture does not rely on any resource reservation or traffic engineering mechanism and does not demand any change in sensitive components such as routers. Unfortunately, the authors do not report any experimental or simulation-based evaluation of this approach. Finally, another architecture for QoS-driven connection management based on MAs is discussed in [Yucel 99].

3.2.4 MA-based Routing

Distributed, dynamic routing is very important in dynamic, large-scale networks. Mobile networking makes the problem of finding efficient routing algorithm even more difficult to tackle. Various authors have studied this problem from the MA perspective.

Schuringa and Remsak use genetic programming to build MAs that monitor the network status and set the routing tables, aiming at maximising network throughput and minimising overall packet delay [Schuringa 00]. Their work originates in biological-driven ant systems, which compute shortest path by resembling the behaviour of social ant colonies [Appleby 94, Steward 94, Schoonderwoerd 96].

Ants are simple MAs whose behaviour is modelled on the trail-laying abilities of real ants. The ants move across the network between randomly chosen pairs of nodes; as they move, they deposit simulated pheromones as a function of their distance from their source node and the congestion encountered on their journey. They select their path at each intermediate node according to the distribution of simulated pheromone at each node.

Schuringa and Remsak significantly improve the original AntN*et al*gorithm [Di Caro 98] and use genetic programming techniques to build MAs. Their agents take the same route as normal packets, according to the routing table information and accumulate information on local queues at each node they visit. When the agent arrives at its destination, it takes the same route backwards and updates the routing tables, according to the collected information. This then is a good example of the use of agent weak mobility for distributed routing.

A multi-agent co-operative approach is taken by Kramer *et al* [Kramer 99]. They use two kind of agents in their simulated scenarios. *Routing agents* are MAs that roam the network to gather topological information and modify routing tables. *Message agents* are smart data packets that use those tables to route themselves. They make their decision about where they need to go based on information that routing agents accumulate and cache.

The work found in the literature on MA-based routing finds its roots in artificial intelligence since the logic embedded in the MAs is inspired to either biological or social behaviour. Nevertheless, the approach aims at simplicity and consequently MAs tends to be relatively simple and small in size. The principle is that by imitating natural behaviour at an individual MA level it is possible to achieve a good collective behaviour. This approach has fostered significant research across the world but has not yet led to concrete deployment. One argument is that such collective behaviour may be difficult to control and cause unexpected problems.

3.3 Mobile Agent-based Monitoring

Monitoring is an essential part of network and system management. Monitoring the health and stability of networks has become crucial to the management of distributed systems relying on those networks. Tools and probes to measure the performance of networks for the purpose of management, fault diagnosis, or performance evaluation have been studied by several research groups. The agent approach has been envisioned as one possibility to deal with large-scale, dynamic systems. In fact, monitoring has been identified as one of the application areas that have the greatest potential of benefiting from agents.

However, current prototypes tend not to exploit important features such as agent mobility and cloning. Interesting work has tackled the problem using a stationary, multi-agent approach For instance, agent co-operation can be exploited to support network-aware distributed applications by capturing the state of the network and using this information to provide adaptability to the changing condition of the network [Wijata 00]. Wijata *et al* propose an architecture to show how static, distributed agents can maintain a view of the network state that is generally dynamic, transient and sometimes tightly coupled with the semantics of the application. Network state includes knowledge of the topology, availability of resources, and the available QoS. Network agents provide integrated control of monitoring network elements and collected performance information.

Static agents are also proposed by Dini *et al* [Dini 97]. The authors propose a model for realising a monitoring agent system that adapts its polling frequency automatically according to predefined criteria. Agents embed also generic functionality for filtering, and event creation. The adaptation capability reduces the performance impact of the agents while increasing the accuracy of management-relevant information. Different adaptation strategies allow a flexible configuration of the agent system according to the individual requirements of the managed components.

Static agents can only provide limited adaptability in the context of highly mobile, dynamic networking environments. Agent mobility adds one degree of freedom to the adaptation capability of the monitoring system. Gavalas has recently dedicated his PhD thesis to MA-based performance management [Gavalas 01b]. His work has led to several publications [Gavalas 99a-d, Gavalas 00a-d, Gavalas 01a-b] to which the interested reader may refer for a survey of the field.

Gavalas's work has several similarities with the work described herein, despite having been developed independently. The author of this thesis was not even aware of Gavalas's work until recently. Gavalas starts from similar assumptions on the limitations of non-MA-based approaches and describes mobility models which suit distributed monitoring. He illustrates practical scenarios which show the potential benefits of MAs in the context of performance monitoring. He has realised a prototype MA platform targeted for management purposes and has experimented with practical performance management applications, carrying out a comparative work between SNMP-based performance monitoring and MA-based alternatives.

Gavalas has experimented with intelligent filtering applications for decentralised performance management. He has implemented simple applications to aggregate several MIB values providing high-level performance indicators, efficiently acquire atomic snapshots of SNMP tables, and filter tables' contents through generic filters.

Similarly to the author of this thesis, Gavalas aimed at assessing performance advantages as well as overheads of MA-based monitoring. However, he followed a completely different methodology based on experimentation and practical realisation of simple scenarios. For this reason he was constrained to implementation difficulties and focused most of his work on the realisation of a management-oriented agent platform, illustrating then its capability through case studies.

From this point of view the work presented herein can be considered as complementary to the one presented by Gavalas. In fact, the type of problems discussed here are, to a large extent, orthogonal to his work. We approach the problem of placing agents optimally in graph-theoretical terms, proposing a scalable algorithmic solution to it, and carrying out our comparative work by a mixed mathematical-modelling and simulation-based methodology. On the other hand, Gavalas's work may be used to support the viability of our solution.

Agent optimal placement and adaptable monitoring are also central themes of the work by Abdu *et al* [Abdu 99]. The authors present an adaptive model in which an initial configuration of management agents is determined according to a set of user and system requirements. These agents can later be dynamically re-configured – *i.e.*, re-located – to adapt to changes in resource availability and user or system constraints. Hence, the requirements and problems addressed by Abdu *et al* are extremely similar to the ones of this thesis. The problem of computing number and location of agents is formulated as an integer linear programming problem. The proposed solution is based on modifications of an existing enumeration algorithm and aims at optimality. This is the major difference with the thesis work. By aiming at location optimality the authors come up with an algorithm characterised by exponential complexity, which is only viable for relatively small-scale systems. On the contrary we study an approximate but scalable solution.

Another work on agent-based network monitoring is reported by Bivens *et al* [Bivens 99]. Their work goes along similar lines of others as far as assumptions and objectives are concerned. However, the work seems to be at its early stage because it does not demonstrate the exploitation of agent mobility and reports only sketchy, partly contradictory experimental results. What is actually exploited is one-hop agent mobility which may be seen as a particular instance of MbD. Multi-hop mobility is envisioned for increasing robustness and fault-tolerance; but this aspect is not demonstrated. What is new in this work is the idea of using the agent for pro-active caching.

Very relevant to this thesis work is the study of agent dissemination models by Theilmann *et al* [Theilmann 99]. The authors study precisely the same problem as we do herein, that is the agent location problem. They present and in-depth formulation and analysis of the problem, elaborating on the reasons why multicast routing algorithms are not suitable to it. Basically, multicast groups are persistent for a large number of messages. Their participants may change over time, but the group as a whole (and its address) remains. That is why multicast routing can adapt its paths over time according to the network conditions and to the group members. In contrast to this, the group including all the agent hosts involved into a specific dissemination problem is only valid for single message – *i.e.*, the one bearing the actual agent being deployed. Setting up a new multicast group for every single agent deployment process would impose an extreme network load since group members are widely distributed. In other words, the cost of building a whole multicast tree does not pay off if this is only used for agent deployment.

Like the author of this thesis, Theilmann *et al* find that one of the major obstacles for effective agent dissemination is the computation of the number and location of agents, for a given network and a given task. They also find a polynomial, approximate solution to this problem. They carry out a thorough assessment of the proposed algorithm, which results in significant performance improvements. The problem with their approach is that, similarly to all other *p-centre* and *p-median* algorithms available in the literature, they rely on the knowledge of the network distance map. They achieve that by using the tool *traceroute*, which determines the routing path between two Internet hosts. This is clearly a major limitation when network scale and dynamics is high, since in this case the computation of the network map involves excessive overheads. In particular, network map computation will involve timescales which are far larger than network dynamics.

3.4 Evaluation of Management Applications based on Mobile Agents

The evaluation of MA-based management involves the comparison of different distributed management paradigms on the one hand and the assessment of the underlying MA platform on the other. The assessment of management systems and applications is not a simple task because, to the best of the author's knowledge, there are no agreed or standardised benchmarking applications. Another problem is the difficulty in getting access to realistic traffic profiles typically generated by management applications. For instance, we need to enter the realm of telecom operators to measure and characterise traffic generated by network management platforms. The lack of benchmarks and traffic patterns is, therefore, a major obstacle towards the development of methodologies for comparing different management systems. This is not the case in fields such as computer architecture, for which such methodologies do exist and are well established.

A similar problem is encountered by those who face the task of assessing and comparing different MA platforms. Consequently, there are no methodologies to carry out the evaluation of MA-based management, which is generally performed following *ad hoc* procedures.

It should be noted that the evaluation of MA-based management involves both *functional* and *non-functional* properties. Most commonly investigations address the functional properties of MA-based systems. Non-functional properties which include issues of *performance*, *scalability*, *adaptability,* and *stability* are more difficult to assess (for the reasons mentioned above) and have so far received only scant attention. In the remaining part of this section we review the approach followed by other researchers to evaluate non-functional properties of MA systems and MA-based management systems.

Three different methods have been followed: 1) by mathematical modelling; 2) by simulation; and 3) by experimentation. These are ‘complementary’ rather than ‘alternative’ methods. For instance, experimental measurements are necessary to fine-tune models and simulations; modelling and simulations are often used to validate each other; and simulations are often used to study the behaviour of the system under particular conditions.

Lee *et al* suggest an interesting method to assess the non-functional properties of multi-agent systems, which unfortunately does not include agent mobility [Lee 98]. Performance models for MA systems are proposed by Straβer and Schwehm [Straβer 97]. Mathematical models for remote procedure calls and agent migration are used to carry out a comparative analysis between those approaches. It is shown that optimality can be achieved by a mixed sequence of the two. The models are validated by measurements of interactions among real agents in the Mole MA system. This work follows a mathematical approach which is far simpler than the one presented in this thesis and captures only higher-level behaviour of the system.

An interesting analytical comparison among the MA paradigm, the Client-Server paradigm, and the Remote Evaluation approach is proposed by Puliafito *et al* [Puliafito 99]. The authors present several models of Petri nets and draw similar conclusions to those of this thesis. They identify conditions in which MAs are not the convenient paradigm due to their overheads.

An experimental evaluation of the MA paradigm is presented by Ismail *et al* [Ismail 99]. The authors compare the performance of two different applications realised, first, over Java RMI (Client-Server approach) and, then, over two different MA platforms. They demonstrate they were able to implement an MA platform characterised by a more efficient migration mechanism in comparison with the Aglets platform. They report agent migration times in the order of hundreds of milliseconds. We have followed a similar path in [Bohoris 00c], implementing a light-weight platform of constrained agent mobility and achieving migration times of the same order of magnitude.

Optimisation of agent migration is a key factor for MA platform efficiency. Soares and Silva have implemented in the JAMES platform (a Java-based MA platform oriented for telecommunications an network management) various optimisation techniques [Soares 99]. These include *caching schemes*, *code prefetching*, *thread* and *socket pooling*, as well as protocol optimisations based on dynamic modification of buffer sizes. All those techniques are shown to significantly impact the overall migration time. The open issue is how applicable those techniques are in general, rather than for specific applications.

On the issue of code migration, Hohl *et al* propose new mechanisms to transport code [Hohol 97]. They consider MA systems that use modular code structure according to an object-oriented approach. In this case, only an MA 'skeleton' is transported between nodes, whereas the necessary classes are transported only when needed. The MA fetches its classes directly on the local host or from neighbouring 'code servers'. An additional way to make the code transport faster is to prefetch classes that may be used in the future. This approach has a great potential of reducing migration times, but no performance analysis has been provided yet, though the author claimed their intention of experimenting on those mechanisms over the Mole MA platform.

Efficient code deployment is also tackled by Lipperts [Lipperts 00]. His work shares with this thesis the ideas of exploiting agent autonomy to provide adaptation to changes through agent migration. However, the migration logic embedded in the agent is different. In our case the agent senses the network routing tables and computes simple estimates on the bases of which it decides whether or not to trigger migration. Instead, Lipperts studies a solution based on utility theory. Agents' preferences between different locations are expressed as utilities. Combining utility states with probabilities of these states provides a means for solving the agent location problem. This 'utility logic' originates in artificial intelligence and is rather complex to embed within the agent if compared with the proposed thesis approach. That is why overheads associated with Lipperts's approach are larger. In fact, the performance results show that relatively small improvements are associated with agents incorporating 'utility logic' in comparison with agent solutions in which migration is triggered without this logic.

A methodological comparison among several existing MA platforms is carried out by Silva *et al* [Silva 00]. The authors create a simple benchmarking application, run it over each of the platforms under examination and measure computational time and incurred traffic. This work if far from identifying benchmarking applications that may be valid in general but has the value of starting off in the right direction. The interesting result is that agent migration time varies between few hundred milliseconds and nearly two seconds. This is an important figure to be considered when relying on general-purpose MA platforms.

Dikaiakos and Samaras propose a novel performance analysis approach to gauge quantitatively the performance characteristics of MA platforms [Dikaiakos 00]. They propose a hierarchical framework of benchmarking designed to isolate performance properties at different level of detail. They examine the behaviour of those benchmarks over various commercial, Java-based, MA platforms.

The works reviewed above show that a lot of investigation has been addressing the evaluation of MA platforms in general. Not so much literature is available on the assessment of the more specific field of MA-based management systems.

Sahai *et al* have experimented with Network Management enabled with MA capabilities [Sahai 97b-c, Sahai 98a-c]. They demonstrate the functionality of the MAGENTA MA platform through a network management application. They have implemented and evaluated a mobile network manager to manage a network from a portable computer through a wireless link. MAs are utilised to study the performance of network components, install software, audit network component usage and for network discovery. Timescales to create and launch MAs were in the order of half a second, while the time for an agent to return back to its manager with results, after performing its task, was in the order of the second. The work by Sahai *et al* illuminates on timescales and performance trade-offs but presents an *ad hoc* performance evaluation of a specific application rather than having a general validity.

A more general approach has first been proposed by Baldi and Picco, who present a quantitative model for traditional (Client-Server) and mobile code design of network management functionality [Baldi 98]. Their aim is to provide guidelines on how to determine the optimal design paradigm according to the model parameters. To do that they follow a methodology based on mathematical modelling, similarly to the work of this thesis. However, their models are extremely simplistic and are not accompanied by any simulation work. They also take a different perspective, that is the one of software engineering. We take a more pragmatic approach by trying to solve some of the hurdles towards realisation of automated MA-based management. Baldi’s work has the value of having motivated more quantitative studies, including the one presented herein.

On the contrary, Rubinstein *et al* follow a simulation-based approach with additional experimentation over MA platforms but do not employ mathematical modelling [Rubinstein 98, Rubinstein 99a-c, Rubinstein 00a-c, Rubinstein 01]. It is interesting that their choice is to adopt NS, the same basic network simulator as the one of this thesis. However, their simulations are conducted in a different way and are more limited in scope. They initially assume very simple LAN topologies rather than conducting a methodological simulation study for general IP network topologies. In more recent work, they extend topologies and include general IP networks generated by GT-ITM, the same topology generation tool adopted herein. However, they only generate a few networks and do not seem to repeat the experiments to ensure statistical significance.

In addition, a number of assumptions are questionable. For instance, they neglect migration overheads by treating MAs as common UDP packets. This assumption makes the results questionable because migration overheads are found in this thesis to be the main limiting factors of MA-based management. Other simulation parameters are set up in a similar fashion as herein, though at the time of designing the simulations the author of this thesis was not aware of Rubinstein's work. This is an encouraging finding which somehow strengthens our simulation work.

Another significant difference with our work is that the itinerary of Rubinstein's MAs is predetermined by the management stations at MA creation time. In this way he does not exploit agent reactivity and autonomy. An important contribution of this thesis is to devise a novel algorithm to target those features which result in increased adaptability of the MA system and, consequently, in increased scalability.

Valuable in Rubistein's work is the comparison between the MA paradigm and SNMP-based management.

MA-based Network Performance Management is evaluated by Gavalas in his recent PhD work [Gavalas 01b]. His models are either inspired to or derived from work carried out by the author of this thesis [Liotta 98a, Liotta 99a, Liotta 99b, Liotta 01a]. His main contribution consists of the practical experimentation with his management-oriented MA platform. Gavalas work has numerous similarities with this thesis and draws similar conclusions. Having been carried out independently and having followed a different methodology, it can be seen as complementary to this thesis.

We can conclude that, despite the fact that a lot of work has addressed the problem of evaluating MA-based management applications, methodological studies of this approach exist but do not provide in-depth evaluation. Existing work based on mathematical modelling provides high-level models which capture only superficial aspects of the system. Simulation-based approaches are rare and are not conducted at the necessary level of detail. Experimental work tends to focus more on the performance evaluation of MA platforms rather than on the management application.

This thesis is inspired by existing work on mathematical modelling but provides more detailed models; provides extensive simulation results; re-uses the experimental measurements of overheads involved in the agent migration process; and brings a novel algorithm towards automation and efficiency in agent dissemination.

3.5 Possible Approaches to the Agent Location Problem

While in the previous sections we have elaborated on works that have used MAs in various ways to solve typical management problems, here we review existing location algorithms that may, in principle, be applied to solve the agent location problem. This has been specified in Chapter 2 as a *p-median*, or minisum problem in which, for a given monitoring task, we want to find the number of agents, *p* and their location which results in minimal incurred traffic by the monitoring system as a whole. We have also seen that this is an NP-complete problem; hence, the agent location problem seeks, in practice, a near-optimal solution computable in polynomial time. For this solution to be actually viable, we also need to be able to give upper-bounds on the total computational time. This is needed because only those monitoring tasks whose duration is reasonably longer than the span of time needed to compute their location are beneficially implemented with MAs.

An important contribution of this thesis is the proposal of a novel solution to the agent location problem. This work has been motivated by the fact that existing *p-median* algorithms do not meet all the requirements of our agent system – *i.e.* they do not satisfy at least one of the properties enumerated below. In the following subsections we shall review those algorithms and show why they are not viable solutions to the agent location problem.

The main requirements on (or properties of) the agent location algorithm are summarised below:

1. *The algorithm can be computed in polynomial time with relatively low polynomial degree*. The polynomial degree should be relatively low (ideally of first degree or at most of second degree) with maximum number of agents, number of monitored objects, and network diameter. Some of the algorithms proposed in the literature are not accompanied with any computational analysis or do not satisfy this requirement.

2. *The algorithm solves the problem for general network topologies.* Some algorithms proposed in the literature solve the problem only for particular classes of topologies, such as tree networks.

3. *The algorithm solves the p-median problem for any p ≥ 1 and smaller than the number of network nodes*. Some algorithms proposed in the literature can only tackle particular cases such as *p*=1 or *p*=2.

4. *The algorithm admits upper-bounds on computational time*. As already mentioned above, only those monitoring tasks whose duration is reasonably longer than the span of time needed to compute their location are beneficially implemented with MAs. Hence it is important to be able to foresee the agent computational time which represents an unwanted overheads for the agent system.

5. *The algorithm provides an indication of the 'goodness' of the computed locations.* By having an indication of the distance from optimality of a given facility system it is possible to take the important decision of whether or not it is worth refining the location procedure. A near-optimality threshold can be adopted to stop the search for better locations. Moreover, in the case of applications such as the agent location problem, one might want to employ a provably near-optimal algorithm to justify the cost of building the system in the first place.

6. *The algorithm does not use the network matrix as input parameter*. Many of the algorithms proposed in the literature are based on the principle that a complete map of the network (or network topology) – *e.g.* the network distance matrix – is available at the node where the location algorithm is executed. This is a major limitation towards scalability. As network scale grows the assumption that an up-to-date topological information is available at a central node becomes more and more unreasonable.

7. *The algorithm can be computed in a distributed fashion*. Again, for scalability reasons it is unreasonable to have to try to solve a highly distributed problem (the agent location algorithm) using a centralised approach. This will result in heavy load at the monitoring station which will be the sole responsible entity for collecting and processing topological information. What is envisioned is an algorithm which can be computed in a distributed fashion to limit the collection overheads and 'parallelise' computational load.

Location theory tackles location problems according to four different approaches [Handler 79]: a) enumeration; b) graph theoretic; c) heuristic; and d) mathematical programming. These will be discussed separately in the following sub-sections. An extensive survey of location algorithms is presented in [Handler 79, Tansel 83a, Tansel 83b, Buckley 90, Evans 92], and [Daskin 95]. Here we only report a selection of the most relevant approaches and comment on their advantages and limitations in tackling the agent location problem.

3.5.1 Enumeration Approach

Enumerating all possible solutions to determine the optimal solution is a naïve approach, and for large networks the required computational effort is unwieldy. As already mentioned in Chapter 2, the *p-median* problem for fixed *p* is O(N^p) in a network with *N* nodes, whereas the time required to solve all *p-median* problems for *p*=1 to *p*=*N* for any given value of *N* is ([Daskin 95] p. 203)

$$\sum_{j=1}^{N} \binom{N}{j} = 2^N - 1 = O\left(2^N\right)$$

which is exponential in *N*. This breaches requirement (1) necessary for the *p-median* algorithm to be a viable agent location algorithm.

A number of polynomial algorithms characterised by low degree has been proposed to solve the 1-median and the 2-median problems. These meet requirement (1) but breach either requirement (2) or (3). For instance, Hakimi presented a simple enumeration procedure for determining an absolute median in a non-oriented network [Hakimi 65]. This approach is actually based on the distance matrix; hence this algorithm breaches also requirement (6).

Enumeration algorithms have also been proposed to solve the *p-median* problem optimally and in polynomial time on a tree network [Kariv 79]. However, these do not meet requirement (2).

3.5.2 Graph Theoretic Approach

Graph-theoretic approaches take advantage of the underlying network structure to determine the *p-medians*. These approaches have been successful only when the underlying network has a non-oriented tree structure [Handler 79]. In particular, Goldman has presented an algorithm that solves the 1-median problem on a tree in just O(*N*) steps [Goldman 71]. Matula and Kolde proposed an algorithm which solves the *p-median* problem on a tree (where *p*>1) in O(N^3p^2) steps [Matula 76]; while an O(N^2p^2) is presented in [Kariv 79].

Therefore, graph-theoretic approaches proposed in the literature breach either requirements (2) and (3).

3.5.3 Heuristic Approach

Heuristic procedures rely on intuitive trial-and-error methods and cannot guarantee an optimal solution, but they can be applied to any general network structure – *i.e.,* they meet requirement

(2). Heuristic procedures are especially useful when ‘good’ rather than ‘optimal’ solution is required. Some of them are based on intuitive notions about the optimal solution of the *p-median* problem, while others attempt to generate common-sense approximate solutions to exact mathematical formulations, mostly integer programming formulations. In fact, some of these heuristics are used to generate better initial solutions and/or branching decision rules to obtain faster convergence in the mathematical programming algorithms.

A comprehensive revision of heuristic algorithms is reported in [Handler 79, Tansel 83a, Tansel 83b, Buckley 90, Evans 92], and [Daskin 95]. Those algorithms belong to the following categories: node partitioning, myopic approach, node substitution, heuristic branch-and-bound, and improvement algorithms. The interested reader may refer to the above references for the details.

The revision of the algorithms proposed in the literature led to some important conclusions. Several algorithms proposed in the literature meet requirement (1), (2), and (3). The major problem of any heuristic approaches is that they do not provide a measure of ‘goodness’ of the solution – *i.e.*, requirement (5) is not met. (See [Daskin 95] p.221.). Most of them do not satisfy requirement (6) either. Requirement (4) is met in some algorithms. Finally, requirement (7) depends on the implementation. However it should be observed that none of the reviewed algorithm is oriented towards distributed computation, being mostly based on an input network distance matrix.

In conclusion it should be mentioned that the agent-based solution proposed in this thesis falls under the umbrella of heuristic procedures.

3.5.4 Mathematical Programming Approaches

The mathematical programming approaches are based on an integer programming formulation of the *p-median* problem. A comprehensive review of mathematical programming algorithms is reported in [Handler 79] and [Buckley 90]. Because of the availability of several integer programming computer routines and the large theoretical base in mathematical programming, these approaches have attracted wide attention and have proved rather successful for general networks. The major impediment to using this approach for the agent location problem is that it relies on the network distance matrix – *i.e.* does not meet requirement (6). Moreover, despite the number of authors reporting computational times (see [Khumawala 72, Jarvinen 72, El-Shaieb 73, Garfinkel 74, Schrage 75]), there is not much literature on theoretical computational complexity of those algorithms.

Particularly relevant to this thesis are algorithms based on *Lagrangian Relaxation* (this technique is discussed in [Geoffrion 74, Held 74, Fisher 75, Cornuejols 77, Narula 77, Daskin 95]). These are optimisation-based approaches based on relaxations of the integer-programming formulation of the *p-median* problem. When coupled with one or more heuristic algorithms, the lagrangian approach often gives results that are provably optimal or very close to optimal [Daskin 95] – *i.e.*, it meets requirement (5). This is an extremely useful metric for establishing the distance from optimality of other heuristic algorithms. For instance, in this thesis we prove that the proposed algorithm leads to a near-optimal solution of the *p-median* problem by showing that this solution is upper-bounded by the one achievable with a provably near-optimal lagrangian algorithm presented in [Daskin 95].

Nevertheless, lagrangian algorithms are not viable to the agent location problem because they do not meet requirements (1) and (6). Computational times are reported in [Cornuejols 77] and [Daskin 95], which show that this approach is only suitable for moderate-sized networks.

3.6 Conclusions

In this Chapter we have surveyed work that is strongly related to the thesis in order to identify gaps in the literature and provide motivation for addressing some of them. We have also highlighted work which is complementary to the thesis as well as issues that are orthogonal to the ones addresses herein. Clearly, this thesis does not address all the unsolved problems, but it is useful to set up the whole picture before concentrating on particular issues.

An important target is to advance towards the realisation of automated MA-based management. Many hurdles delay this process. We tackle one of them, that is the study of algorithms to automate the agent dissemination process in a way which results in near-optimal distribution of monitoring agents.

Another problem is carrying out the evaluation of MA-based management in a methodological manner, aiming at results which are statistically significant and of practical use. We have discussed literature gaps on methods, benchmarks, traffic profiles, and models. This thesis proposes models (both mathematical and for simulation) and follows a mixed mathematical and simulation-based methodology. The thesis, though, does not take a software engineering perspective. As such, it does not have the aim of providing general models or proposing a general methodology. Nevertheless, the approach followed here may serve as a starting point for other researchers.

We can see from the literature survey that MAs are associated with a variety of properties. However, when it comes to MA-based management, MAs tend to lose most of them and to degenerate into a mechanism for dynamic programming remote elements. The gap here is the exploitation of the real essence of ‘agents’ and MAs. The position of this thesis is to try and exploit to a larger extent agent *mobility*, *autonomy*, *cloning*, *reactivity*, and *pro-activity*. The hypothesis is that those properties may lead to increased performance and scalability, and may facilitate adaptation.

This Chapter concludes the first part of this thesis which has introduced the thesis work, the research hypothesis, the main objectives and motivations, and has surveyed background material and related work.

PART II

THESIS CONTRIBUTIONS

Chapter 4

Mobile Agent based Distributed Monitoring

This Chapter initiates the second part of the thesis which includes the various research contributions. Chapter 4 introduces the proposed *active distributed monitoring* approach in an incremental fashion. A simpler solution, which is computed in a centralised fashion, allows focusing on the basic algorithmic ideas behind the proposal. The same principles apply to the distributed version of that algorithm which is described afterwards. We can say that the thesis proposes two flavours of a single algorithm that solves the agent location problem.

Given two systems, the *monitored system* and the *monitoring system*, the former is exemplified by a set of interconnected nodes. When they are monitored, nodes are seen by the agents as Monitored Objects (MO). The monitoring system includes a set of algorithms implementing the actual monitoring operations. We recall from the previous chapters that the difference between *centralised* and *distributed* monitoring systems is that in the former case the monitoring algorithms are executed in a single node by a single manager; whereas with distributed monitoring the monitored system is partitioned into subsystems each of which is monitored by a separate *area manager*.

In a *static distributed monitoring system* both the location and operations of the area managers are predetermined at design time by the main manager. In contrast, in an *active distributed monitoring system* the area manager locations and operations are computed dynamically and can be changed even during the execution of the monitoring algorithms, in order to provide adaptation to changes in the network status.

The application of code mobility to distributed monitoring is conditional on efficiently solving the *agent location problem* introduced in Chapters 2 and 3.

In the vision of this thesis, area managers are implemented with mobile agents following the weak mobility paradigm. Given a monitoring task specified by a manager, the agent system needs to determine the following, as depicted in Figure 4-1: 1) the number of agents required to implement the task in a distributed fashion; 2) the set of managed objects monitored by each agent; 3) the location of each agent; and 4) the operations implemented by each agent.

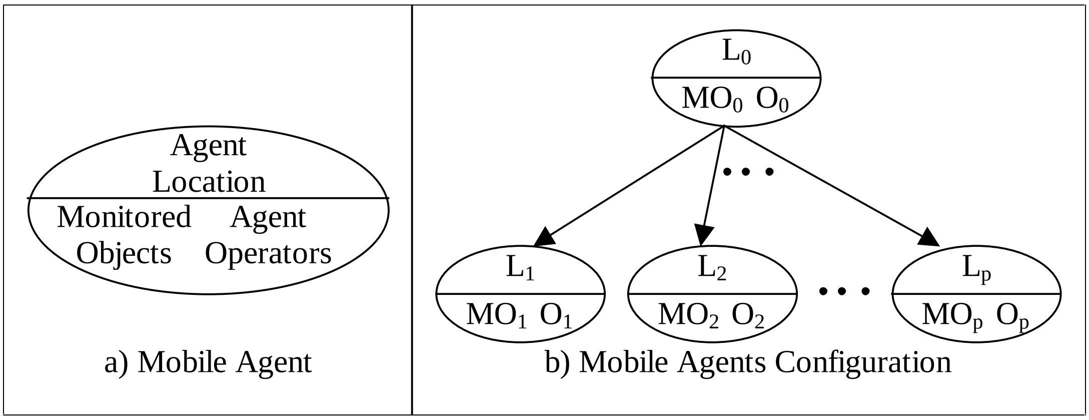


Figure 4-1: a) Main components of a mobile agent; b) Agents configuration.

The aim is to efficiently place the agents in the network in a way that optimises the traffic incurred by the monitoring system as well as its overall response time.

In accordance with the focus of the thesis, this chapter is mainly dedicated to the description of two different flavours of the proposed agent location algorithm which finds an approximate solution to problems 1-3. This algorithm aims at placing *p* monitoring agents appropriately in the network. The centralised algorithm uses the *constrained mobility* concept during the agent deployment phase (Section 4.1). The distributed one, uses the *weak mobility* approach and assumes agents capable of *cloning* (Section 4.2).

The main difference between the two flavours of the proposed algorithm regards their algorithmic computational complexity, as described in Chapter 6. The performance of a distributed monitoring system based on the proposed agent location algorithm is assessed by simulation in Chapter 8.

The agent location algorithm computes, for a given monitoring task, a suitable number of area managers (or agents) and their location, and partitions the network in order to create associations between portions of the network and area managers. Problem 4 identified above – *i.e.*, the automatic agent task decomposition for the determination of the actual operations realised by each agent – has assumed a more marginal role in this thesis. This relates to the classic problem of specifying routines which can be executed in a parallel or distributed fashion. Research in the area of parallel computing suggests that this problem is very difficult, if not impossible, to solve in a general fashion. In Section 4.3 we elaborate a bit on the possibility of solving it for a particular class of monitoring tasks. However, we shall only provide some reasoning without actually solving the problem in a general fashion.

Section 4.4 captures the main features of the proposed agent-based distributed monitoring system. Finally, Section 4.5 elaborates on the possibility of further exploiting agent weak mobility to build *active* distributed monitoring systems capable of adapting to changes in the network state even after the agent deployment. This is another topic which, despite its interest, has been investigated only marginally in this thesis.

4.1 Centralised Location Algorithm for Agents Incapable of Cloning

We describe here a centralised algorithm that, given a monitoring task specified by the manager, finds an approximate solution to the agent location problem. The algorithm is computed at the monitoring station and assumes the availability of a simple code mobility infrastructure supporting constrained mobility. It makes use of simple routing information – *i.e.* next-hop address and costs such as hop-counts – obtainable from network routers through standard network management interfaces. This could be obtained, for example, through the use of SNMP.

It should be emphasised here that this usage of information that is already available in the network is significant. It removes the need for the agent system to perform additional network performance tasks and, consequently, improves efficiency. Routing tables are used but not manipulated by the agent system. They are maintained by routing algorithms, which operate independently from the agent system. In this way the resulting network partitioning reflects indirectly the status of the network.

The algorithm is initially illustrated by showing its basic steps for the simple example network topology of Figure 4-2 (nodes are routers or hosts connected by point-to-point links). After that, a more formal specification is given.

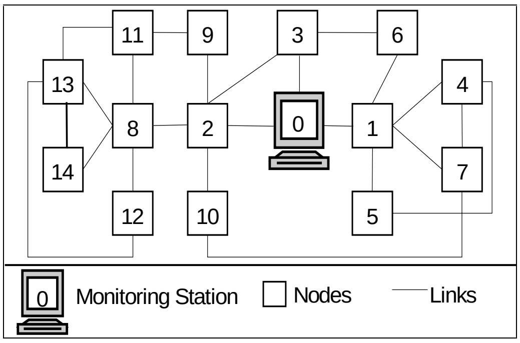


Figure 4-2: Example Network Topology.

4.1.1 Centralised Algorithm by Example

In the case of the centralised algorithm, agent location and deployment are performed sequentially. The agent location is computed at the monitoring station. Agents are, then, created at the monitoring station and dispatched to their target destinations.

The core of the algorithm is represented by a recursive network partitioning process which relies on a heuristic procedure. This procedure computes the partitioning starting from the monitoring station node by appropriately processing the *'next_hop' addresses* and *routing costs* between the current node and every other node to be monitored. This information is typically stored in local routing tables.

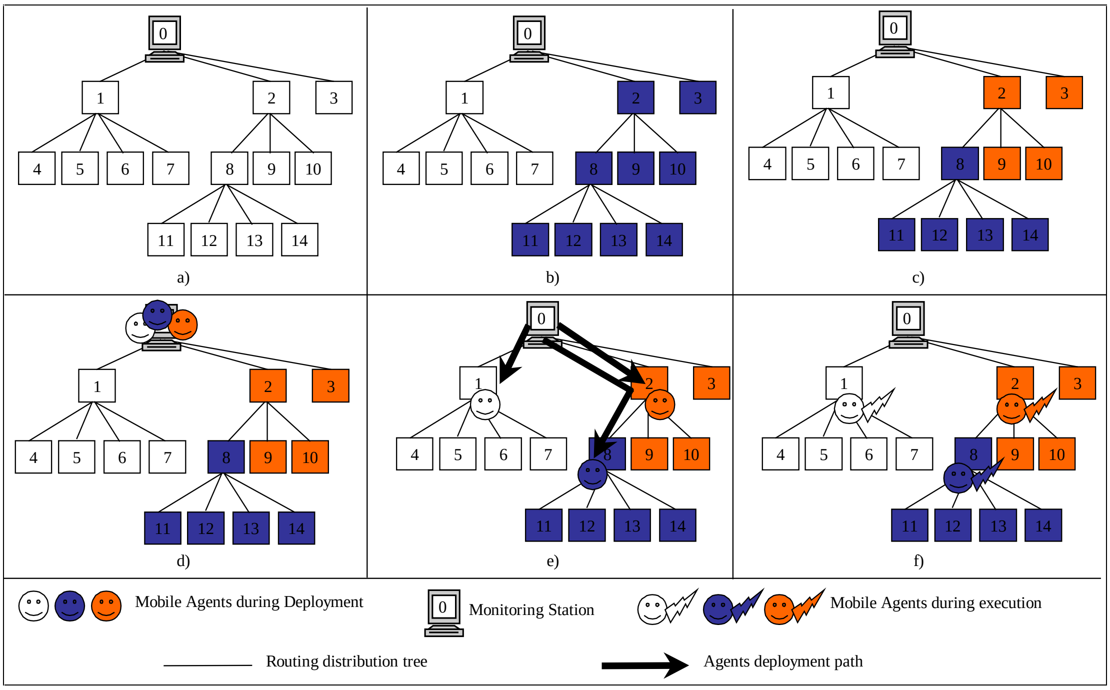

a)
b)
c)
d)
e)
f)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Mobile Agents during Deployment
Monitoring Station
Mobile Agents during execution
Routing distribution tree
Agents deployment path

Figure 4-3: Agent configuration steps for the centralised algorithm.

Figure 4-3 illustrates the basic steps of the partitioning process for the example network of Figure 4-2. It is assumed that the monitoring station monitors nodes 1 to 14; hence, the routing tree rooted at node 0 is depicted (Figure 4-3a). One of the heuristic components of the procedure consists in estimating the number of agents required to implement the given monitoring task in a distributed fashion. This is done by considering parameters such as total number of monitored nodes and network radius, and constraints such as the maximum allowed deployment time.

In the example, approximately two thirds of the nodes are reached through node 2. Most of the remaining nodes are reached through node 1. There are not enough nodes through node 3 to justify a third partition, since separate partitions are meant to be monitored by independent agents. Thus, the network is partitioned into two sub-partitions (Figure 4-3b). One agent is associated with the smaller partition and other two are associated with the larger one. The larger partition needs now to be further divided into two. Node 2 becomes the current node from which the partitioning process continues (Figure 4-3c). Eventually three agents are created, each one associated with a different partition (Figure 4-3d), and the deployment process is initiated (Figure 4-3e). Finally, the agents start monitoring the nodes (Figure 4-3f). Each agent concentrates the information collected from its partition before returning it to the central monitoring station in the form of periodic notifications or reports and events or alarms.

Figure 4-4 depicts the situation upon deployment completion. The number of agents has been established along with their location and the system has been partitioned. We can observe that the monitoring path – *i.e.*, the agent-to-monitored nodes communication path – does not necessarily coincide with the agent deployment path.

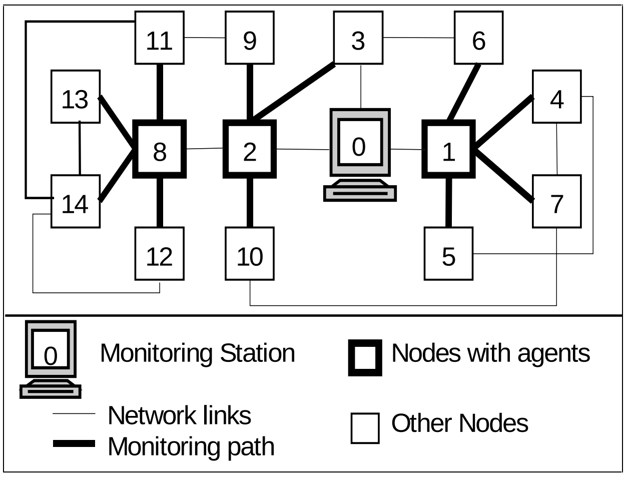


Figure 4-4. Final agent location and monitoring path.

4.1.2 Centralised Algorithm by Flow-chart Diagram

The centralised version of the proposed agent location algorithm is depicted in Figure 4-5 in the form of a flow-chart diagram. This is a recursive algorithm which visits the network distribution tree following a depth-first approach. The algorithm is computed on the network matrix which has to be provided as one of the input parameters. Another parameter is the percentage of agents (*P*) with respect to the number of monitored nodes (*P=p/N* whereby *p* is the actual number of MAs as well as the number of network partitions, since one agent per partition is assumed).

Other input parameters include the routing tables (one per network node); the monitoring station node (*u*); and the heuristic function used to subsequently create the network partitioning at each iteration. The heuristic utilised for the simulations of the centralised algorithm is very simple. It utilises as input parameters, a network node (*v*) acting in the role of root node; the number of agents candidate for that node; the number of monitored nodes reached through that node; and the sum of routing costs extended to all the monitored nodes reached through the given root node. Routing costs are extracted from the routing table of the root node.

The heuristic function provides as output an estimate of the number of agents that should be forwarded to each of the root's neighbour nodes. MAs are associated to neighbour nodes proportionally to the number of monitored nodes (reached through the neighbour node) and to their associated routing costs. Hence, network partitions including a larger number of monitored nodes or/and whose associated routing cost is higher are allocated with a proportionally large number of agents.

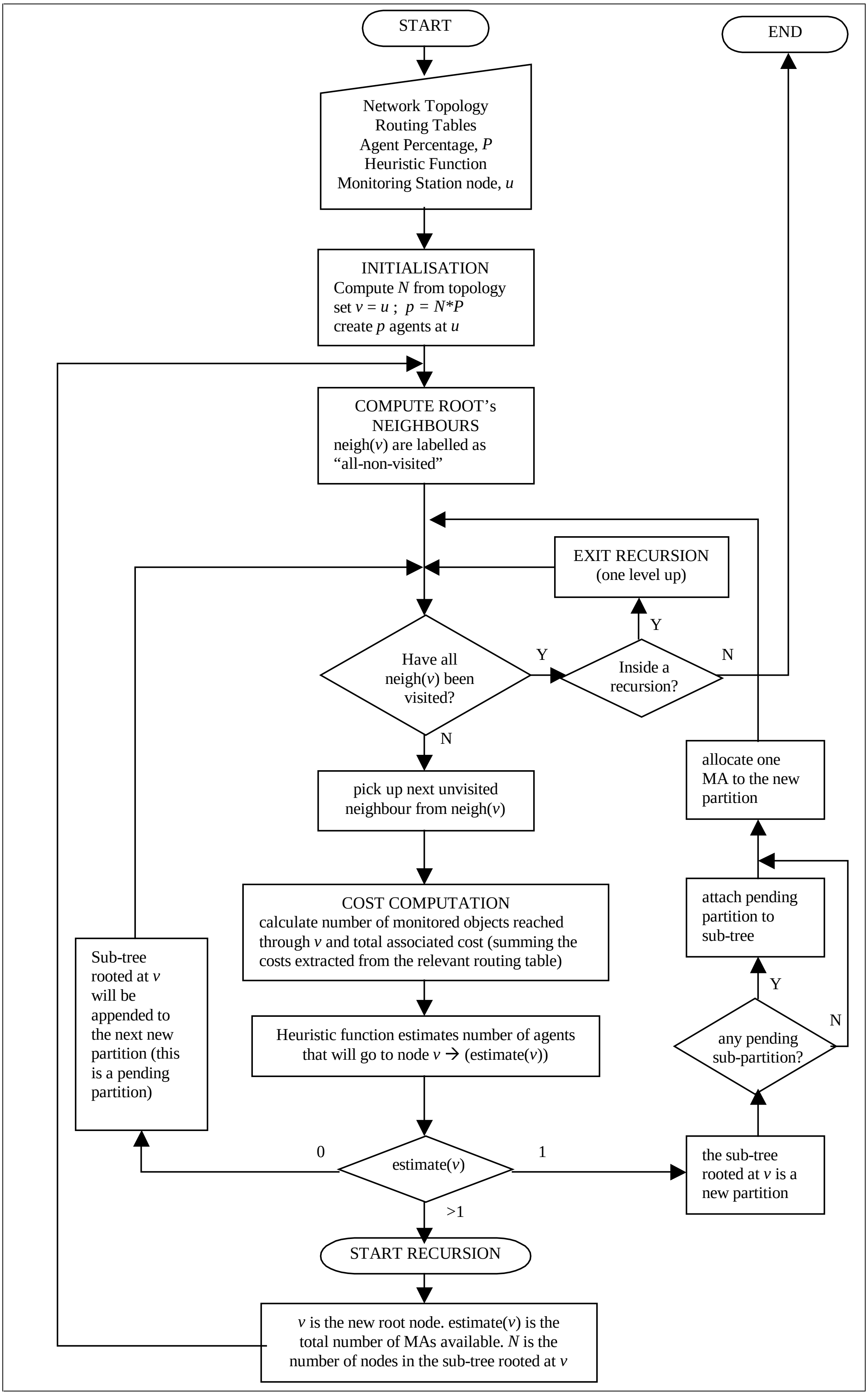

START
END
Network Topology
Routing Tables
Agent Percentage, P
Heuristic Function
Monitoring Station node, u
INITIALISATION
Compute N from topology
set v = u ; p = N*P
create p agents at u
COMPUTE ROOT's NEIGHBOURS
neigh(v) are labelled as "all-non-visited"
EXIT RECURSION (one level up)
Y
Have all neigh(v) been visited?
Y
Inside a recursion?
N
N
pick up next unvisited neighbour from neigh(v)
allocate one MA to the new partition
COST COMPUTATION
calculate number of monitored objects reached through v and total associated cost (summing the costs extracted from the relevant routing table)
attach pending partition to sub-tree
Sub-tree rooted at v will be appended to the next new partition (this is a pending partition)
Y
N
Heuristic function estimates number of agents that will go to node v → (estimate(v))
any pending sub-partition?
0
estimate(v)
1
the sub-tree rooted at v is a new partition
>1
START RECURSION
v is the new root node. estimate(v) is the total number of MAs available. N is the number of nodes in the sub-tree rooted at v

Figure 4-5. Flow-chart of centralised version of the agent location algorithm.

4.1.3 Centralised Algorithm Formally

A more formal description of the proposed algorithm is given in Listing 4-1. The key symbols, procedures, and variables are specified in Table 4-1 and a commentary on the MA algorithm is given below.

```
p ← heur_1(|V|; min_deploy_time; R(u);...)
FOR x = 1 to p
   ma_ID ← new_ma(x)
   MAs ← MAs ∪ ma_ID
   node(ma_ID) ← u
   monitored(ma_ID) ← {}
curr_nd_ID ← u
MA(curr_nd_ID) ← MAs
MO(curr_nd_ID) ← V \ u

neigh(curr_nd_ID) ← {v | v ∈ V & dist(curr_nd_ID; v)= 1}
FOR EACH v ∈ neigh(curr_nd_ID)
   MO(v)← {x | x ∈ MO(curr_nd_ID) &
            next_hop(curr_nd_ID; x) = v }
   cost(v)← $\sum_{(y|y\in MO(v))}$[routing_cost(curr_nd_ID, y)]

FOR EACH v ∈ neigh(curr_nd_ID)
   estimate_MA(v)← heur_2{ MA(curr_nd_ID);
      $\bigcup_{(x\ |\ x\in neigh(curr\_nd\_ID)}$[MO(x)]; $\sum_{(x\ |\ x\in neigh(curr\_nd\_ID)}$[cost(x)] }

FOR EACH v ∈ neigh(curr_nd_ID)
   IF (estimate_MA(v) > 0) DO
      FOR y = 1 to estimate_MA(v)
         MA(v) ← MA(v) ∪ {ANY z ∈ MA(curr_nd_ID)}
         MA(curr_nd_ID) ← MA(curr_nd_ID) \ z
         node(z) ← v
      IF MA(curr_nd_ID) ≠ {}
         MO(curr_nd_ID)← MO(curr_nd_ID) \ MO(v)
      ELSE
         MO(v) ← MO(v) ∪ MO(curr_nd_ID)
         MO(curr_nd_ID) = {}
      curr_nd_ID ← v
      Start Recursion
# The remaining monitored objects in MO(curr_nd_ID)
# are equally distributed among the remaining agents
# MA(curr_nd_ID).

FOR EACH ma_ID ∈ MAs
   send ma_ID to node(ma_ID)

# Each agent autonomously starts executing its
# monitoring task upon arriving to its target node.
```

Listing 4-1: Centralised version of the agent location algorithm based on constrained mobility.

The network is modelled as a graph $G=(V,E)$ consisting of a set of vertices (or nodes), V and edges, E. Each edge, $e\in E$ has two end points, v_1 and $v_2\in V$. Given two generic nodes v_x and $v_y\in V$, let us denote by $d(v_x)$ the degree of v_x in G, by $dist(v_x,v_y)$ the distance between v_x and v_y. Let $R(v_x) = \max_{\{v|v\in V\}} \{dist(v_x,v)\}$ be the radius of the network centred in v_x; $neigh(v_x) = \{v \mid v\in V$ & $dist(v_x,v) = 1\}$ is the set of neighbours of v_x in G; and T_r is the routing tree rooted at the monitoring station.

KEY TO SIMBOLS '←' assignement; '\' difference between sets; '{.}' set or list; '{}' empty set or list; '∈' is element of; '|' conditional clause	
KEY TO PROCEDURES	
heur_1	heuristic function which computes the required number of agents on the basis of the monitoring task features and of some network topological information
new_ma(.)	procedure generating a new agent
dist(nd_ID; v)	distance between node nd_ID and node v
routing_cost(nd_a, nd_b)	procedure which extracts from the local routing table the estimated cost to reach node nd_b from node nd_a.
heur_2	heuristic function estimating how many of the agents residing in curr_nd_ID should be migrated to node v. It works on the basis of the total number of agents in curr_nd_ID, the total number of nodes monitored from curr_nd_ID, and the total costs associated to perform the monitor from curr_nd_ID
KEY TO VARIABLES	
p	total number of agents
MAs	set of the ID of all agents
node(ma_ID)	the node ID of the agent identified by ma_ID
monitored(ma_ID)	the set of nodes monitored by the agent identified by ma_ID
curr_nd_ID	the node of the algorithm current iteration
u	node hosting the main monitoring station
MA(nd_ID)	set of agents residing in the node identified by nd_ID
MO(nd_ID)	set of nodes reachable through nd_ID that will be monitored by the agents residing in nd_ID
neigh(nd_ID)	set of neighbour nodes of nd_ID
cost(v)	total cost associated to monitor the object reachable through v from curr_nd_ID
estimate_MA(v)	number of agents that will be migrated from node curr_nd_ID to node v

Table 4-1: Key symbols, procedures, and variables for the algorithm of Listing 4-1

The algorithm is divided into 4 main parts:

1. Initialisation (lines 1-9). A simple heuristic procedure determines the number of required agents p on the basis of the total number of nodes $N=|V|$, the network radius, and other relevant data – *e.g.* constraints on maximum allowed agent deployment time. For example a simple heuristic to estimate the number of required MAs, p is to create a number of agents equal to a certain portion of the total number of monitored nodes. For instance, the simulations reported in Chapter 8 show that if p is too small there is no significant difference in performance between distributed and centralised monitoring. It is also shown that large values of p result in large agent deployment overheads and that a good trade-off is to fix p at 10-20% of N. Once the number of agents has been determined, the agents are created and initialised. Their initial location is the monitoring station node; their set of monitored nodes is empty. Their operations (not shown in the listing) are set up by the manager.

2. Iteration (lines 10-33). This part represents the core of the algorithm. It is an iterative procedure that, by progressively matching the monitoring task parameters with network topology and routing information, ends up computing the location of each agent along with its set of monitored nodes.

3. Agent Deployment (lines 34-35). Once the agent locations and set of MOs have been computed the agent deployment process is initiated.

4. Start Monitoring (lines 36-37). Each agent starts performing its monitoring task over its set of MOs upon arriving to its target location.

The above algorithm can be further illustrated by referring again to the example network schematised in Figure 4-2 and Figure 4-3. The main steps of the agent location algorithm are captured in Table 4-2.

Algorithm Execution State		**Agent MA$_1$**	**Agent MA$_2$**	**Agent MA$_3$**
Initialisation	Location	0	0	0
	MOs	{}	{}	{}
First Iteration	Location	1	2	2
	MOs	{1, 4-7}	{2, 3, 8-14}	{}
Second Iteration	Location	Iteration ends because estimates is 0	2	8
	MOs		{2, 3, 9, 10}	{8, 11-14}
Third Iteration			Iteration ends	Iteration ends
MA Deployment		Migration to node 1	Migration to node 2	Migration to node 8

Table 4-2: Main steps of the algorithm of the centralised algorithm for the example network of Figure 4-2.

The initial estimate is that three agents are needed. Thus, three agents are created at the monitoring station. Their ‘location’ field is set to the monitoring station node ID that is zero; their list of monitored agents is set to an empty list. The first iteration results in estimating one agent for node 1, two agents for node 2 and no agent for node 3. In fact, not enough nodes are reached through node 3 (there is actually only node 3) so the deployment of an agent to node 3 would not be justified. The MOs are associated with the agents accordingly and the second iteration is initiated. Agent MA$_1$ is confirmed to have to be in node 1, whereas the remaining agents are subject to another level of iteration. This iterative process eventually ends when all agents are associated with nodes from which no better locations can be found. At this point the agent migration process is actually initiated from the monitoring station node.

4.2 Distributed Location Algorithm for Agents Capable of Cloning

We describe here a distributed flavour of the agent location algorithm presented in Section 1.1 above. This algorithm assumes the existence of an agent system supporting *weak mobility* and *agent cloning*. Like the centralised algorithm, the distributed one makes use of routing information obtainable from network routers through standard network management interfaces. So it is assumed that this information is available.

We also assume that MA hosts - *i.e.*, locations in which MAs are able to run - are evenly distributed within the network. This, in other terms, means that for each router there is at least one MA host that is located relatively close to it and, for each LAN, the number of MA hosts is proportional to the number of MOs that need to be monitored in that LAN. Under these assumptions, the MA distribution tree —*i.e.*, the set of routes used for MA deployment— does not differ significantly from the routing tree rooted at the monitoring station. Without loss of generality, we envisage a scenario in which routers can act as MA hosts during MA deployment. In such a case, the MA distribution tree would actually coincide with the routing tree.

Similarly to the approach followed in the previous section, the algorithm is initially illustrated by showing its basic steps for the simple example network topology of Figure 4-2. After that, a more formal specification is given.

4.2.1 Distributed Algorithm by Example

Unlike the centralised algorithm, the distributed one computes agent locations during agent deployment. The network partitioning process is initiated at the monitoring station but is then taken over by the agents, which perform it in a distributed fashion. This process is illustrated in Figure 4-6. The manager initially delegates a given task to one agent and starts it at the monitoring station (Figure 4-6a). This agent, as well as all the agents created subsequently, employs a heuristic procedure for partitioning the network, which is analogous to the one used in the centralised algorithm. Thus, based on heuristic information, the first agent (residing at the monitoring station) partitions the network into two parts by processing the *‘next_hop’ addresses* and *routing costs* between the current node and every other node to be monitored (Figure 4-6b). We recall that in the centralised algorithm this information is stored as a data structure in the monitoring station. Instead, with the distributed algorithm routing information is directly extracted from the routing tables of the local router.

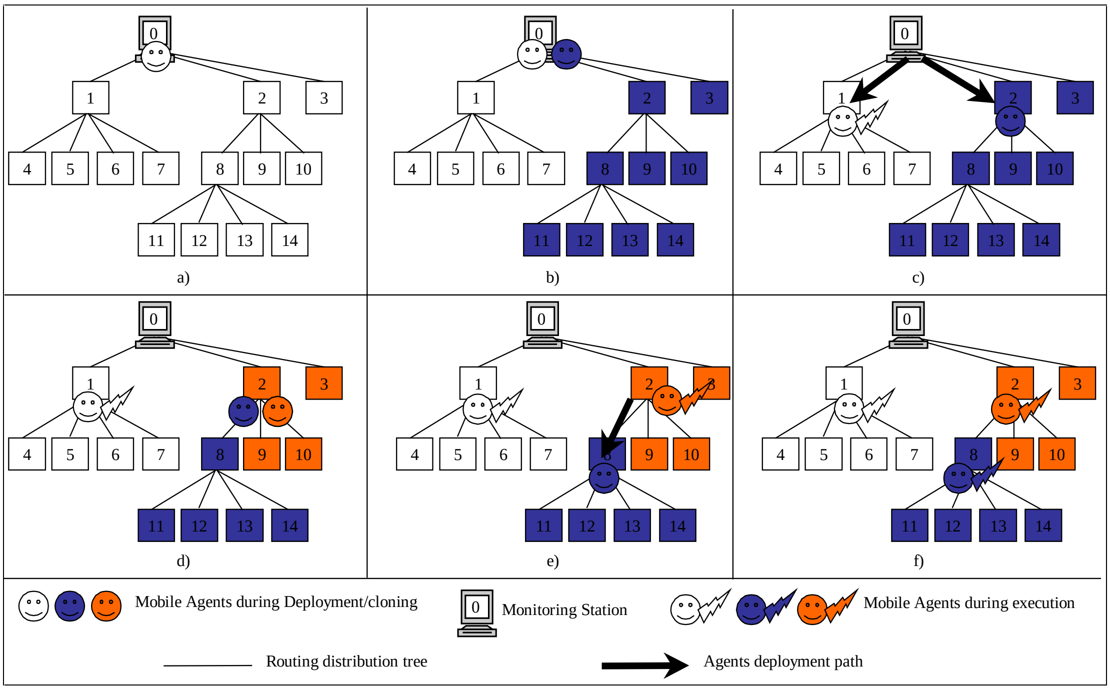

a)
b)
c)
d)
e)
f)
Mobile Agents during Deployment/cloning
Monitoring Station
Mobile Agents during execution
Routing distribution tree
Agents deployment path

Figure 4-6: Agent configuration steps for the distributed algorithm.

Since one agent per partition is required, the agent will clone and set up a second agent. This procedure involves the monitoring task decomposition which includes the specification of the sub-tasks for the two agents and the appropriate assignment of monitored nodes (see Section 4.3).

After that, each agent autonomously searches its best location and migrates to it (Figure 4-6c). The agent in location 1 is now ready to start since it has estimated that its current location is the one with minimum cost. Conversely, the agent in location 2 decides to share its task with another agent and clones it (Figure 4-6d). The decomposition/migration process starts again leading to one agent running in node 2 and the other migrating to node 8 (Figure 4-6e). Eventually, the agent in location 8 has found its cheapest location and starts executing (Figure 4-6f).

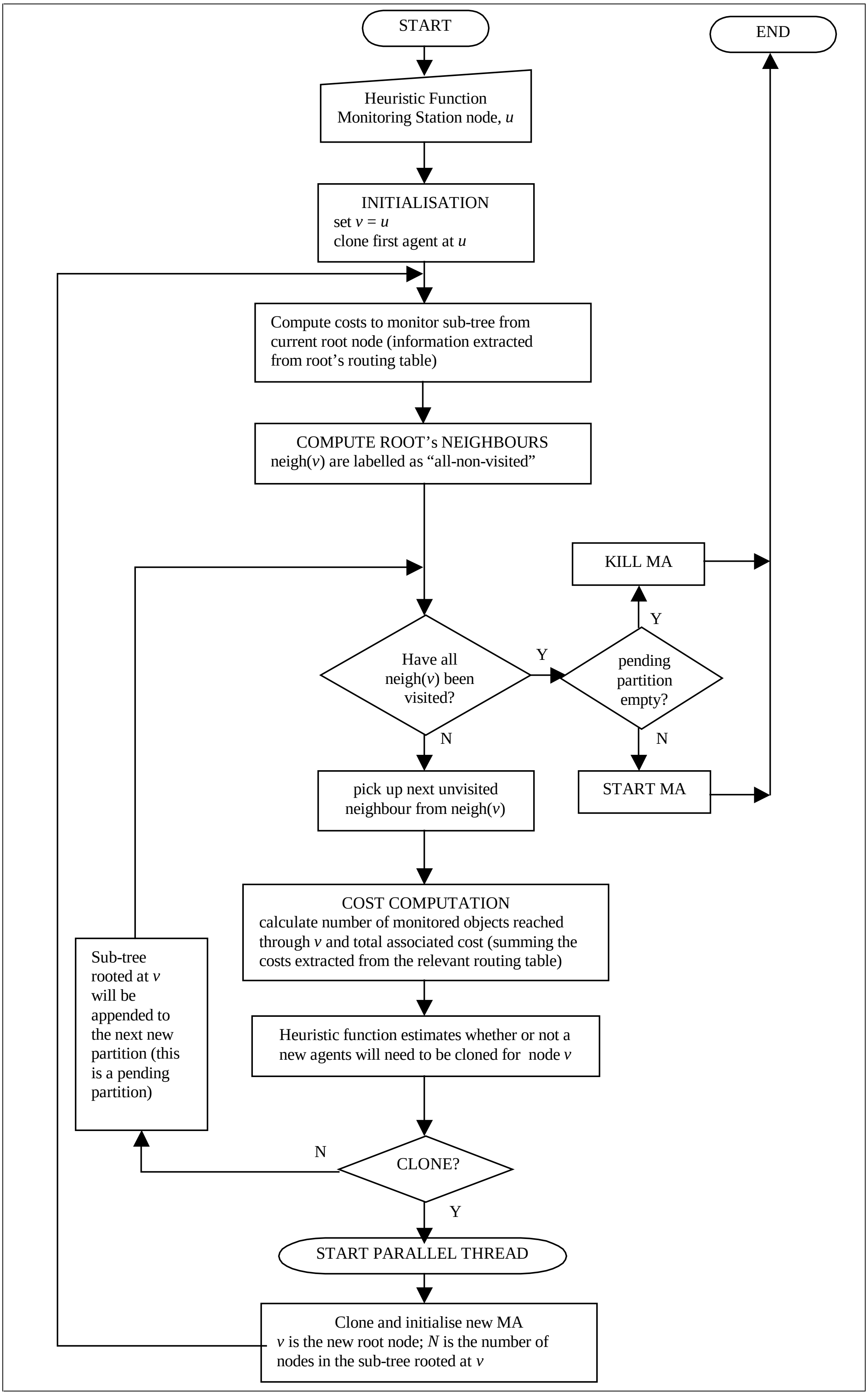
It can be noted that the use of cloning results in minimal agent deployment traffic. In fact, only two agents leave the monitoring station, although the resulting number of agents is three. The cloning algorithm is executed in a distributed fashion (on nodes 1, 2, and 8). Finally, the processing is performed in parallel among nodes at the same level (1 and 2).

4.2.2 Distributed Algorithm by Flow-chart Diagram

The distributed version of the proposed agent location algorithm is depicted in Figure 4-7 in the form of a flow-chart diagram. Although the chart may appear very similar to that of Figure 4-5, there are several differences. First, this algorithm is based on parallel threads of execution rather than on a sequential, recursive procedure. Starting from the root node, an agent may spawn autonomous agents who visit disjoint portions of the network distribution tree in parallel. Spawned agents may in turn spawn other agents, achieving a high level a parallelism.

A second important difference is that the algorithm is not computed on the network matrix (which is in fact not provided as an input parameter); each agent makes decisions on whether or not to partition its portion of the network on the bases of information extracted from the local routing tables.

Another missing input parameter is the agent percentage. This is not required because, unlike the centralised algorithm, the distributed algorithm determines the *location* of agents as well as their *number*. For this reason, the distributed algorithm adopts a slightly more sophisticated heuristic function. In this case the decision of whether or not to clone a new agent and spawn it to the one of its neighbours (*v*) is based on the following parameters: 1) the cost associated to the current agent to monitor all the nodes belonging to the sub-tree rooted at the agent node; 2) the number of the monitored objects reached through *v*; 3) the monitoring cost associated to a candidate agent sitting in *v* and in charge of monitoring the sub-tree rooted at *v*.



START
END
Heuristic Function
Monitoring Station node, *u*
INITIALISATION
set *v* = *u*
clone first agent at *u*
Compute costs to monitor sub-tree from current root node (information extracted from root's routing table)
COMPUTE ROOT's NEIGHBOURS
neigh(*v*) are labelled as "all-non-visited"
KILL MA
Y
Have all neigh(*v*) been visited?
Y
pending partition empty?
N
N
pick up next unvisited neighbour from neigh(*v*)
START MA
COST COMPUTATION
calculate number of monitored objects reached through *v* and total associated cost (summing the costs extracted from the relevant routing table)
Sub-tree rooted at *v* will be appended to the next new partition (this is a pending partition)
Heuristic function estimates whether or not a new agents will need to be cloned for node *v*
N
CLONE?
Y
START PARALLEL THREAD
Clone and initialise new MA
v is the new root node; *N* is the number of nodes in the sub-tree rooted at *v*

Figure 4-7. Flow-chart of distributed version of the agent location algorithm.

Costs are extracted from routing tables. For instance the cost to monitor a node *x* for an agent located in node *v* is equal to the value stored in the *cost* field of the routing table stored at *v* from the row whose *destination* field is equal to *x*. The total cost that an agent has to pay in order to monitor all the sub-tree rooted at the node where the agent is sitting, is equal to the sum of the individual routing costs of each of the involved monitored nodes.

The heuristic function appropriately weights the various input parameters in order to determine whether or not to clone a new agent. An important parameter is the number of nodes in the sub-tree of the neighbour under examination – *e.g.*, *v*. The larger is this number, the higher is the probability to spawn an agent to node *v*. This part of the heuristics works similarly to the centralised algorithm. A good trade-off threshold has been found to be in the order of 10% (i.e., the cloning probability is increased when the number of nodes in the sub-tree is greater than 10). A second driving factor is the difference between the cost associated to the agent and the one achieved when the sub-tree rooted at the candidate neighbour *v* is monitored by a newly spawned agent sitting in *v*. The probability to spawn an agent to node *v* is increased proportionally to this gradient.

4.2.3 Distributed Algorithm Formally

A more formal description of the proposed algorithm is described in Listing 4-2. It starts by creating a first agent at the main monitoring station – *i.e.*, the node where the monitoring information eventually needs to be collected (Step 1-2). This agent is initially configured to monitor all the other network nodes (Step 3) and initiates an iterative procedure which determines whether it is appropriate to clone other agents by progressively matching the monitoring task parameters with network routing information (Steps 4-19). Hence, the core of the algorithm is the heuristic procedure (Step 11) which, for each node adjacent to the agent node, creates estimates based on the following costs extracted from the local routing tables: 1) total cost to monitor all the nodes by the current agent; 2) total cost that would be involved if the subset of nodes reached through a candidate neighbour node were monitored by a new agent located in this node; 3) the sum of the costs computed in 2) extended to all the neighbour nodes.

```
1  ma_ID ← clone_ma
2  ma_nd_ID ← u
3  monitored(ma_ID) ← {V \ u}
4  cost(ma_nd_ID) ← $\sum_{(y|y\in MO(ma\_nd\_ID))}$[routing_cost(ma_nd_ID; y)]
5  neigh(ma_nd_ID) ← {v | v ∈ V | dist(ma_nd_ID; v)= 1}
6  FOR EACH v ∈ neigh(ma_nd_ID)
7     MO(v)← {x | x ∈ monitored(ma_nd_ID) &
8              next_hop(ma_nd_ID; x) = v }
9     cost(v)← $\sum_{(y\ |\ y\in MO(v))}$[routing_cost(ma_nd_ID, y)]
10 FOR EACH v ∈ neigh(ma_nd_ID)
11    estimate_MA(v)← heur(cost(ma_nd_ID); cost(v); |MO(v)|;
12                           $\sum_{(x\ |\ x\in neigh(ma\_nd\_ID))}$[cost(x)])
13 FOR EACH v ∈ neigh(ma_nd_ID)
14   IF (estimate_MA(v) = 1)
15      new_ma_ID ← clone_ma
16      new_ma_nd_ID ← v
17      monitored(new_ma_ID) ← MO(v)
18      monitored(ma_ID)←{monitored(ma_ID) \ monitored(new_ma_ID) }
19      migrate_ma(ma_ID=new_ma_ID; ma_nd_ID=v)
20 IF (monitored(ma_ID) ≠ {})
21    start_ma(ma_ID)
22 ELSE
23    kill_ma(ma_ID)
```

Listing 4-2: Distributed version of the agent location algorithm based on strong mobility.

Whenever a neighbour node is selected by the heuristic procedure, the agent creates a new agent to whom it delegates the task of monitoring all the nodes reached through the neighbour node (Steps 13-18). The new agent is then migrated to this neighbour node where, upon arriving, the agent will re-initiate the iterative process (Step 4-19). The iterative process ends when no new agents need to be cloned, according to what is estimated by the heuristic procedure.

KEY TO SIMBOLS '←' assignement; '\' difference between sets; '{.}' set or list; '{}' empty set or list; '∈' is element of; '|' conditional clause
KEY TO PROCEDURES 'clone_ma' creates a new agent; 'routing_cost(nd_A; nd_B)' extracts from the local routing table the estimated cost to reach node nd_B from node nd_A; 'dist(nd_A; nd_B)' returns distance between node nd_A and node nd_B; 'next_hop(nd_A; nd_B)' extracts from the local routing table the distance between nodes nd_A and nd_B; 'heur(.)' applies heuristics based on previously computed costs to estimate whether a new agent should be cloned for a neighbour node. 'migrate_ma(.)' performes all the necessary operations to migrate an agent to a different node; 'start_ma(ma_ID)' agent ma_ID starts monitoring operation; 'kill_ma(ma_ID)' agent ma_ID is redundant and is then terminated.
KEY TO VARIABLES 'monitored(ma_ID)' set of nodes monitored by the agent ma_ID; 'cost(nd_ID)' total cost payed to monitor the nodes reached through node nd_ID; 'neigh(nd_ID)' set of neighbour nodes of node nd_ID; 'MO(nd_ID)' set of nodes monitored trough node nd_ID; 'estimate_MA(nd_ID)' binary variable: value 1 indicates that a new agent should be sent to node nd_ID; value 0 indicates the opposite.

Table 4-3: Key symbols, procedures, and variables for the algorithm of Listing 4-2.

As a final remark it is important to notice that the centralised agent location algorithm described in Section 4.1 (page 76) leads to the same results – *i.e.*, agent locations and network partition – as the distributed algorithm. This derives from the fact that both algorithms use the same heuristic procedures to partition the network and create associations between network partitions and agents.

However, an important difference is that while in the distributed algorithm the agent location is actually computed during agent deployment, in the centralised algorithm those two steps are performed sequentially. Computational complexity and typical execution times of these algorithms are assessed in Chapter 6.

4.3 Decomposition of Monitoring Tasks

The proposed agent location algorithm computes, for a given monitoring task, a suitable number of area managers (or agents) and their location, and partitions the network in order to create associations between portions of the network and area managers. The actual operations, or sub-tasks, implemented by each agent are determined as part of the agent configuration process by decomposing the given monitoring task into appropriate sub-tasks.

In a simple case, the monitoring task is specified by the manager who also specifies each individual sub-task. This approach, though, requires that the agent configuration problem is solved in a centralised fashion, since task decomposition and agent location are interdependent and will have to be computed at the management station.

On the other hand, if a distributed computation of the agent location is chosen, the monitoring task must be specified in a way that allows for its automatic decomposition. In this case, starting from a given task, sub-tasks are generated automatically during the agent cloning/deployment process and depending on the state of the underlying network.

We have carried out a preliminary study of the specification of 'automatically decomposable' tasks as reported in [Liotta 98b]. However, a formal treatment of this particular aspect is beyond the scope of the thesis. Therefore, the remaining part of this section is limited to the description of task-decomposition by example, according to the procedure adopted in the course of the thesis experimentation.

Let us assume that $T(X)$ represents the given monitoring task operating on the set of nodes X and $\{X_1, X_2, \ldots, X_p\}$ are p disjoint partitions of X with $X \equiv \bigcup_{i|i\in\{1..p\}} X_i$. We say that $T(X)$ is an automatically decomposable task if it is possible to find p generally different tasks ($T_1, T_2, \ldots, T_p$) and some function $f(.)$ for which the following expression holds: $T(X) = f(T_1(X_1), T_2(X_2), \ldots, T_p(X_p))$.

A simple example of directly decomposable monitoring task specification can be observed in Listing 4-3, which is a snippet of the OTCL code implemented for the simulations reported in Chapter 8. In this case, the monitoring station simply polls the Monitored Objects (MOs) twice a second to extract the value of attribute *A* and generates an immediate alarm if any of these attributes is greater than a given threshold. In addition, the sum of these attributes is computed and notified to the monitoring station every minute. Therefore, if we refer back to the parameters of Figure 4-1b on page 75, L_0 is specified in *line 2*, MO_0 is specified in *line 3*, and O_0 is specified in *lines 4-14*.

```
#SNIPPET OF MONITORING TASK SPECIFICATION
1 set manager_node_id 0              #location of monitoring station
2 set agent_location 0               #specify initial agent location
3 set MOs {1-14}           #list of monitored objects with ID 1 to 14
4 set list_of_attributes {A}         #get value of A from each object
5 set poll_rate 2                    #Polling_Rate >= 2 polls/sec
6
7 set task_duration 5                #duration of monitoring operation
8                                    #in seconds
9 set data_processing {sum all}      #sum values of attribute A from
10                                   #all objects
11 set notify 60                     #notify the sum to the manager
12                                   #every minute
13 set alarm {if any > 80}           #send an alarm to the manager if
14                                   #any value > 80
```

Listing 4-3: Specification of a simple directly decomposable monitoring task.

The task can be decomposed into virtually identical sub-tasks, which differ only in the value of two parameters: the list of monitored objects and the location in which each task will be executed (see Listing 4-4). In fact, each sub-task will monitor a disjoint subset of the original list of objects. These parameters are computed by the agent location algorithm described in the above Sections 4.1 and 4.2. The monitoring station will appropriately aggregate the information received from the two agents implementing the sub-tasks.

```
# SNIPPET OF FIRST SUB-TASK
1 set manager_node_id 0             #location of monitoring station
2 set agent_location 1              #agent location
3 set MOs {1,4-7}                   #list of monitored objects
4 set list_of_attributes {A}        #get value of A from each object
5 set poll_rate 2                   #Polling_Rate >= 2 polls/sec
6 set task_duration 5               #duration of monitoring operation
7 set data_processing {sum all}     #sum values of attribute A from
8                                   #all objects
9 set notify 60                     #notify the sum to the manager
10 set alarm {if any > 80}          #send an alarm to the manager if
11                                  #any value > 80

# SNIPPET OF SECOND SUB-TASK
12 set manager_node_id 0            #location of monitoring station
13 set agent_location 2             #agent location
14 set MOs {2,3,8-14}               #list of monitored objects
15 set list_of_attributes {A}       #get value of A from each object
16 set poll_rate 2                  #Polling_Rate >= 2 polls/sec
17 set task_duration 5              #duration of monitoring operation
18 set data_processing {sum all}    #sum values of attribute A from
19                                  #all objects
20 set notify 60                    #notify the sum to the manager
21 set alarm {if any > 80}          #send an alarm to the manager if
22                                  #any value > 80

# SNIPPET OF MONITORING STATION TASK
23 set task_duration 5              #duration of monitoring operation
24 set data_processing {sum all}    #sum values notified by agents
```

Listing 4-4: Specification of the resulting sub-tasks implementing the task of Listing 4-3.

The task of Listing 4-3 is directly decomposable into sub-tasks because the actual operations performed on data – specified in *lines 4-14* – are invariant with respect to the number and location of sub-tasks. In particular, the instructions of line 9 and 13 are invariant to task decomposition.

An example monitoring task that requires a more complex task decomposition procedure is reported in Listing 4-5. This task differs from the one of Listing 4-3 in the actual operations performed on the data collected from the nodes. In particular, the operation specified in *line 9* is invariant to decomposition for the case of Listing 4-3, whereas it is non-invariant in the case of Listing 4-5.

```
#SNIPPET OF MONITORING TASK SPECIFICATION
set manager_node_id 0            #location of monitoring station
set agent_location 0             #specify initial agent location
set MOs {1-14}          #list of monitored objects with ID 1 to 14
set list_of_attributes {A}       #get value of A from each object
set poll_rate 2                  #Polling_Rate >= 2 polls/sec

set task_duration 5              #duration of monitoring operation
                                 #in seconds
set data_processing {avg all}    #computes the average value among
                                 #attribute A of all managed objects
set notify 60                    #notify the avg to the manager
                                 #every minute
set alarm {if any > 80}          #send an alarm to the manager if
                                 #any value > 80
```

Listing 4-5: Specification of a non-directly decomposable monitoring task.

This task can be decomposed into two sub-tasks to be implemented by separate agents as shown in Listing 4-6. It is worth noticing that in order for the manager to be able to compute the correct average, the two area agents will have to periodically notify not only their computed average values but also the number of averaged elements. In fact, the final average value computed at the monitoring station will be:

$$avg = \frac{\sum_{i \in Agent\_list} avg_i * num\_MOs_i}{\sum_{i \in Agent\_list} num\_MOs_i}$$

where avg_i and num_MOs_i are the average values and the number of monitored objects respectively computed by the generic agent *i* and the sums are extended to all the agents implementing the sub-tasks.

```
# SNIPPET OF FIRST SUB-TASK
set manager_node_id 0              #location of monitoring station
set agent_location 1               #agent location
set list_of_managed_objects {1,4-7} #monitored objects
set list_of_attributes {A}         #get value of A from each object
set poll_rate 2                    #Polling_Rate >= 2 polls/sec
set task_duration 5                #duration of monitoring operation
set data_processing {avg all}    #computes the average value among
                                 #attribute A of all managed objects
set num_Mos [llenght $MOs]    #compute number of monitored objects
set data_to_be_notified {$data_processing $num_MOs} #list of data
                                 #to be notified to manager
set notify 60                      #notification period
set alarm {if any > 80}            #send an alarm to the manager if
                                   #any value > 80

# SNIPPET OF SECOND SUB-TASK
set manager_node_id 0              #location of monitoring station
set agent_location 2               #agent location
set list_of_managed_objects {2,3,8-14} #monitored objects
set list_of_attributes {A}         #get value of A from each object
set poll_rate 2                    #Polling_Rate >= 2 polls/sec
set task_duration 5                #duration of monitoring operation
set data_processing {avg all}    #computes the average value among
                                 #attribute A of all managed objects
set num_Mos [llenght $MOs]     #compute number of monitored objects
set data_to_be_notified {$avg $num_MOs} #list of data
                                 #to be notified to manager
set notify 60                      #notification period
set alarm {if any > 80}            #send an alarm to the manager if
                                   #any value > 80

# SNIPPET OF MONITORING STATION TASK
set task_duration 5              #duration of monitoring operation
#repeat for each agent
set tot_sum [exec $tot_sum + $avg * $num_MOs] #update sum of A
                                              #values
set tot_Mos [exec $tot_Mos + $num_MOs] #update total number of MOs
#end of loop
set final_avg [exec $tot_sum / $tot_MOs] #compute averages
                                         #notified by agents
```

Listing 4-6: Specification of the resulting sub-tasks implementing the task of Listing 4-5.

Therefore the task of Listing 4-5 is not invariant to task decomposition but the decomposition process can still be automated provided that the manager who originally specifies the monitoring task also specifies the task decomposition procedure. The bold lines are produced by an appropriate task decomposition procedure that has to be specified along with the original monitoring task, in order to be able to automate the task decomposition process.

In conclusion, tasks such as the ones of Listing 4-3 and Listing 4-5 can be automatically decomposed provided that suitable decomposition algorithms are provided. The former case can be formalised as $T(X) = f(T_a(X_1), T_a(X_2))$. The latter can be formalised as $T(X) = f(T_b(X_1), T_c(X_2))$. More generally, tasks that are invariant to decomposition can be specified as $T(X) = f(T_a(X_1), T_a(X_2), \ldots, T_a(X_p))$, whereas the ones which are variant to decomposition will be specified as $T(X) = f(T_1(X_1), T_2(X_2), \ldots, T_p(X_p))$.

Within the context of this thesis, automatically decomposable tasks are assumed. These are the most common ones and the only ones that can be implemented with the proposed agent approach.

Finally, it is worth mentioning that agents are not bound to monitor only objects. More generally agents can monitor other agents, thus resulting in a hierarchical agent organisation. For example, the task of Listing 4-3 can be decomposed into a set of sub-tasks executed by agents organised in a hierarchy. In this case agents at a low level of hierarchy will monitor the MOs, whilst agents at a higher level will sum the reports from sub-agents and report just the sum to their superior – which implies aggregation and improved efficiency. Clearly, the agent deployment algorithm would require some amendments to cater for hierarchical agents organisations.

4.4 Main Features of Agent-based Distributed Monitoring

By deploying the agents according to either of the above agent location algorithms a *dynamic, distributed* monitoring system is realised. The system is distributed because the actual monitoring operation is performed by autonomous agents placed in different network locations. The system is dynamic because the agent location is not predetermined at design time, yet it depends on the network state. Agent locations reflects network state at deployment time; variations in network state may trigger agent migration to maintain location optimality.

Intuitively, this solution is more efficient than centralised monitoring and more flexible than static decentralised monitoring. Agents can be used to perform a wide variety of processing operation on raw data – such as data gathering, filtering and aggregation – while residing as close as possible to where the data is actually stored. In principle, MAs can be located optimally in order to minimise the monitoring traffic and distribute the processing burden, alleviating the central monitoring station. In turn, agent locality may result in the ability to detect problems locally and react to them promptly.

Agents can implement monitoring operations which are much more sophisticated than the simple probes offered by conventional management systems. In principle, agents can be delegated extremely elaborated monitoring tasks by a central manager. However, in practice this level of complexity may not be always necessary. Furthermore, complex agents tend to be large in size and this results in inefficient migration. It appears, then, that one of the difficulties in designing an effective agent solution for network monitoring is that of finding the balance between the agent complexity and its size. Another crucial factor is the amount of resources consumed by the agent from its hosting entity.

The proposed solution is an example of a monitoring system based on relatively simple, autonomous mobile agents. In fact, the agents' location is decided by performing simple processing of routing information and, once deployed, the agents can process data and report back to a central monitoring system without involving inter-agent communication or collaboration.

Therefore, our distributed agent-based monitoring system combines the simplicity of the location algorithm with the flexibility deriving from the use of mobile agents. Similarly to other approaches to distributed monitoring – such as static hierarchical monitoring – it is inherently more scalable than centralised monitoring, as quantified by the simulations reported in Chapter 7. In addition, the agent approach allows for a dynamic delegation of monitoring intelligence without requiring any modification of the monitoring system, the monitored entities and of the monitoring protocol.

Finally, it is worth mentioning that the distributed agent location algorithm represents a concrete example of a practical application of agent weak mobility. In fact the key properties used to find a near-optimal solution to the *p-median* problem in linear time are agent migration and agent cloning, as shown in Chapters 6 and 7 below.

4.5 Adaptability to Network Changes with Agent-based Monitoring

Our agent approach can be regarded as an evolution of previous research on management decentralisation [Goldszmidt 98] but it differs significantly from other techniques based on static decentralisation of control [M3010 91, Stallings 96]. Agents support dynamic decentralisation in two ways: first, at deployment time they can autonomously choose their target location; and second, during the execution of the monitoring tasks they can adapt to dynamic changes in the network status either by migration or cloning. In the latter case, an *active* distributed monitoring system is realised.

Adaptability is perhaps one of the most interesting features of agents systems and can be exploited in distributed monitoring in order to keep the monitoring agents in near-optimal locations. Adaptability may assume a key role in the context of modern networked systems, which are characterised by rapidly increasing size and which exhibit a very dynamic behaviour.

The conventional approach is to achieve adaptability by dynamically changing the routing tree rooted at the monitoring station. This is indirectly achieved as a side effect of the operation of the network routing algorithms, independently from the monitoring system. As a result of congestion or failures, monitoring packets get re-routed generally through longer paths and, consequently, both traffic and response time tend to deteriorate.

In addition to that, a system based on mobile code can actively adapt to network changes by dynamically moving the agents to more advantageous positions. We can employ three different approaches to adaptability: 1) periodic agent re-deployment; 2) adaptation through agent migration based on autonomous agent strategies; 3) adaptation through agent migration based on co-operative agent strategies.

The first approach involves the periodic re-execution of the agent location algorithm by the main monitoring station. It is the simplest one but periodically involves the full agent deployment overheads, which can be prohibitive for very large systems and is inefficient in cases involving only local network changes, since it is likely that many MAs will end up precisely as before.

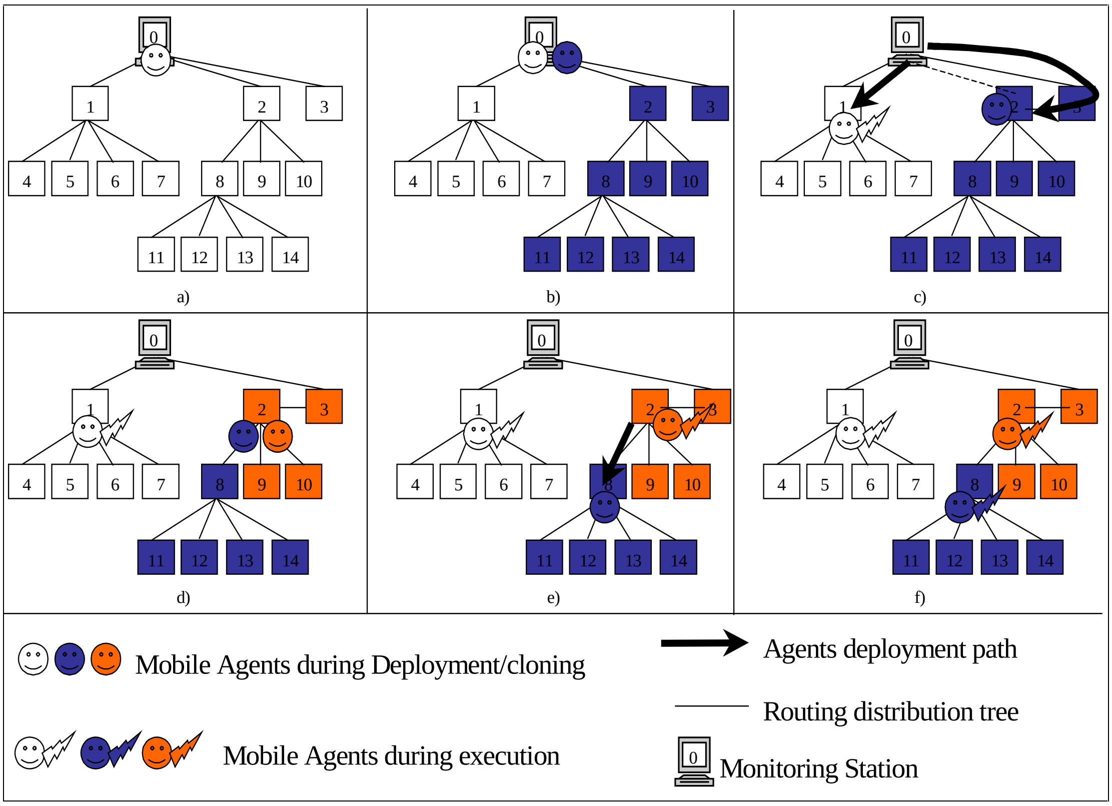

a)
b)
c)
d)
e)
f)
Mobile Agents during Deployment/cloning
Mobile Agents during execution
Agents deployment path
Routing distribution tree
Monitoring Station

Figure 4-8. Example agent full re-deployment, following a link failure.

This is illustrated with a simple example in Figure 4-8. In that case, following a link failure between nodes 0 and 2, no agent migration is triggered because none of the nodes belonging to the agent sitting in node 2 is affected by this fault. However, if for instance we were to follow a policy of periodic re-deployment, the agents will simply follow a different deployment path but will all end up in the same locations where they were sitting before the occurrence of the fault (Figure 4-9).

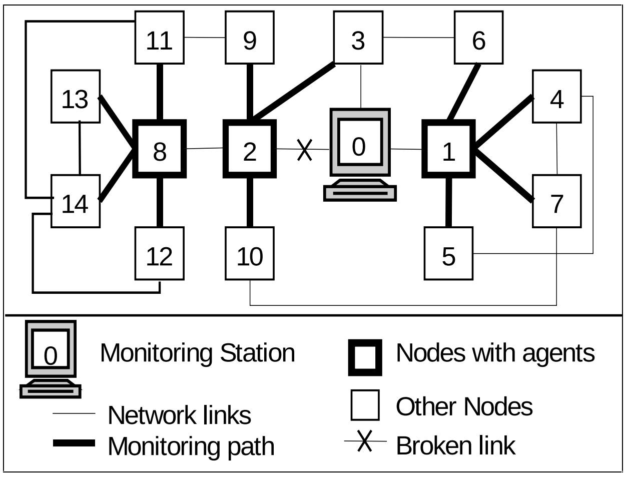


Figure 4-9. Monitoring path for the example of agent full re-deployment.

The second approach realises a simple adaptive system based on autonomous migrating agents. Following the initial agent deployment, each agent is assigned a disjoint sub-set of the MOs. Agents continue to monitor their initial set of objects and periodically estimate the cost of alternative locations by adopting estimation procedures analogous to the ones of Section 4.2. Migration is triggered when the cost reduction is significant. Therefore, the adaptation strategy of this approach is based on local decisions only. Locality results in simplicity but has the drawback of not considering global optimisation strategies.

A simple example illustrating agent self-regulation in response to a link failure is depicted in Figure 4-10. In this case, following a loss of connectivity between node 8 and 13, a new (longer) monitoring path is established between node 8 and node 13. As a result, the central node for the system partition comprising nodes {8, 11, 12, 13, and 14} becomes node 14. Hence, the agent originally located in node 8 will relocate to node 14, bringing the system back to optimality.

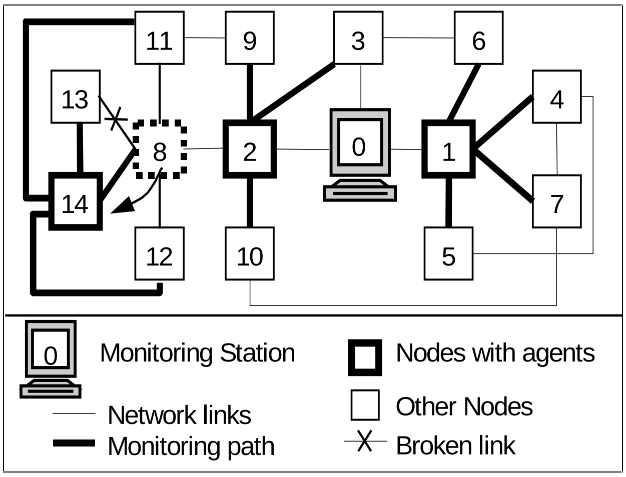


Figure 4-10. Example adaptation through agent migration, following a link failure.

Finally, adaptation based on co-operative agents presents the difficulty of designing a complex system and involves additional inter-agent communication and processing overheads. This form of adaptation is not considered hereafter.

4.6 Summary and Conclusions

This Chapter introduced the concept of active or dynamic distributed monitoring, a possible way of addressing some of the limitations of ‘centralised’ and ‘static distributed’ monitoring. An approach to dynamic monitoring based on mobile agents was proposed. Pre-conditional to this approach is the solution of the agent location problem. Two different flavours of the same algorithm are proposed, a centralised and a distributed one. These algorithms represent a core part of the thesis work since they have been designed to find an approximate solution to the more general problem of placing *p* servers – or service centres – optimally in a network of *N* nodes. This problem is also known as the *p-median* problem which has been studied extensively since the sixties and is known to be NP-complete.

The remaining part of the thesis is dedicated to the assessment of the algorithms presented here, first theoretically (Chapter 6 and 7) and then by simulation (Chapter 8). In particular, the computational complexity of these algorithms is assessed and an estimation of typical computation times is given. Since the algorithms solve the location problem in an approximate way, the distance from optimality is evaluated. Performance and scalability issues are addressed as well. The method used to carry out the assessment is detailed in the following Chapter 5.

Chapter 5

Evaluation Methodology

The hypothesis introduced in Chapter 1 is examined in the next chapters through the evaluation of the proposed active, distributed monitoring system, presented in chapter 4. A hybrid methodology was adopted, based partly on mathematical modelling and partly on simulation. A third option, based on experimentation, was ruled out having been judged less suitable as explained below.

The pure experimental approach would have required the implementation of the proposed agent system either on a real network or on a test-bed. The actual implementation would have not been the problem because open-source MA platforms have become widely available in the last few years and a significant amount of code could have been reused to realise code mobility, agent cloning and so on. What would have been a problem is the actual experimentation phase. In order to carry out the required comparative performance analysis between static and active monitoring a number of network parameters need to be varied, network state needs to be replicated during different experiments (for fair comparisons), and congestion or failure conditions need to be created. Clearly networks that are large enough to be interesting are also expensive and difficult to control; thus, they are rarely available for experimental purposes. On the other hand, small, controlled testbed networks would have been limiting for the assessment of scalability, one of the major parameters and factors under study.

A hybrid methodology was chosen to facilitate the evaluation of the various aspects involved in the agent system. *Mathematical modelling* was used to study the more theoretical aspects of the work, including the following ones:

- the asymptotic complexity of the proposed agent deployment algorithm, *i.e.* its scalability;

- the typical order of magnitude of agent deployment overheads, *i.e.* deployment time and traffic;
- the sufficient conditions on network topology for which the proposed agent location algorithm places the agents near-optimally in the network;
- the study of the agent system under those near-optimal conditions and its comparison with the static monitoring approach

Complementary to the mathematical approach, simulations were carried out to study the agent-based monitoring system under general conditions, for the case of realistic internetworks, including the following aspects:

- the quantitative comparative evaluation of performance and scalability between static monitoring and active, distributed monitoring;
- the assessment of the goodness of the agent locations computed by the proposed algorithm, *i.e.* its distance from optimality;
- the preliminary assessment of the ability of the agent system to adapt to network changes.

Therefore, the research hypothesis was assessed following the path depicted in Figure 5-1. We first studied the transient behaviour of the agent system under general conditions by mathematical analysis (Chapter 6). In this context, the transient behaviour includes the phenomena involved during agent deployment. Then, sufficient conditions for location near-optimality were mathematically obtained (first part of Chapter 7). A study of the system under those near-optimal conditions was conducted both at transient and steady-state time, comparing its performance and scalability with the static monitoring approach (second part of Chapter 7). Finally, a steady-state analysis was conducted by simulation to be able to draw conclusions on the viability of the system and on its degree of optimality for practical network topologies (Chapter 8).

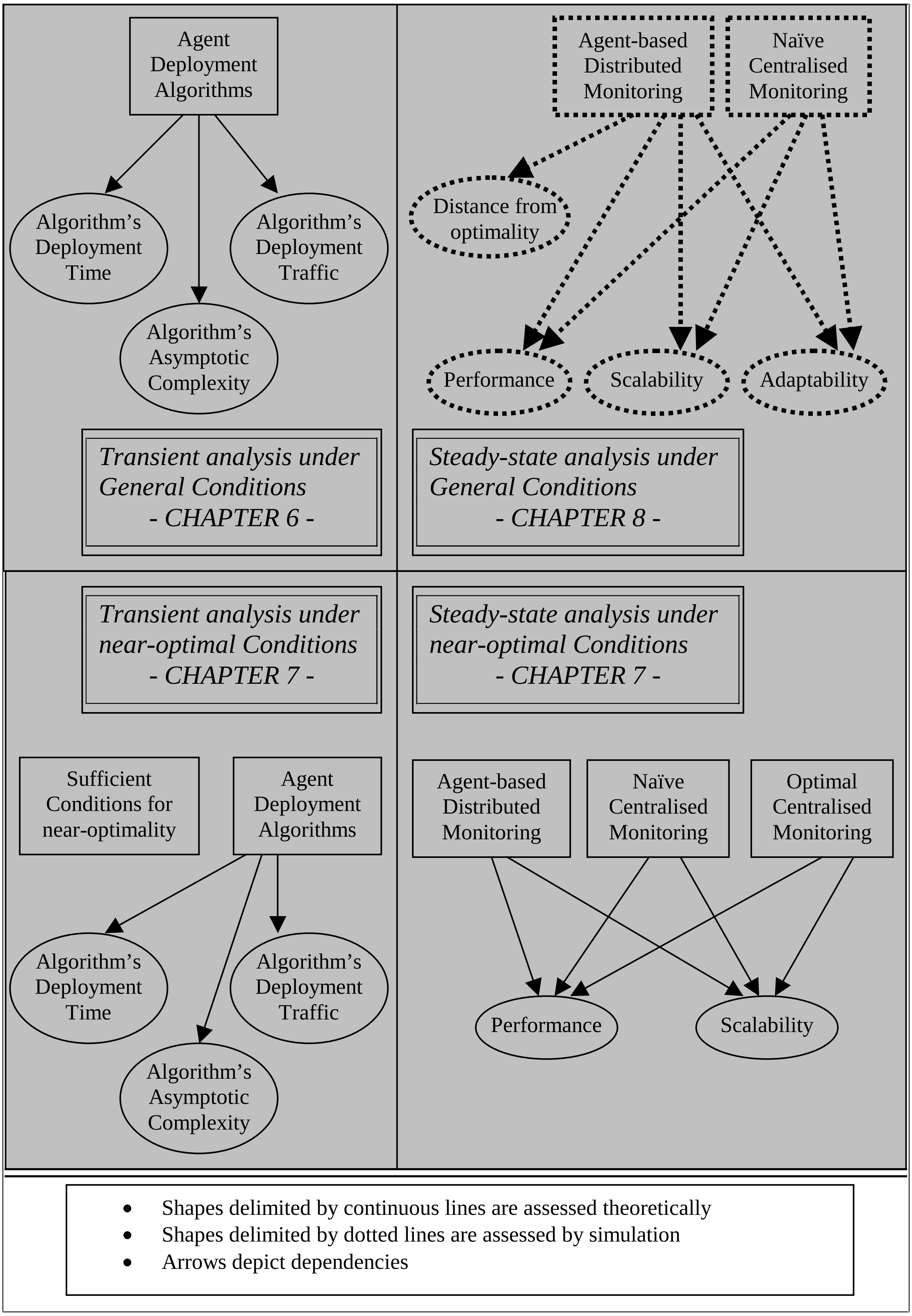

Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Transient analysis under General Conditions - CHAPTER 6 -
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Distance from optimality
Performance
Scalability
Adaptability
Steady-state analysis under General Conditions - CHAPTER 8 -
Transient analysis under near-optimal Conditions - CHAPTER 7 -
Steady-state analysis under near-optimal Conditions - CHAPTER 7 -
Sufficient Conditions for near-optimality
Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Optimal Centralised Monitoring
Performance
Scalability
Shapes delimited by continuous lines are assessed theoretically
Shapes delimited by dotted lines are assessed by simulation
Arrows depict dependencies

Figure 5-1: Schematic representation of the assessment methodology.

The system performance evaluation was carried out following the systematic approach suggested in [Jain 91] (pp.22-28) which consists of 10 basic steps: 1) state goals and define the system; 2) list services and outcomes; 3) select metrics; 4) list parameters; 5) select factors to study; 6) select evaluation technique; 7) select workload; 8) design experiments; 9) analyse and interpret data; 10) present results.

Steps 1 and 2 have already been covered in the previous chapters, in particular in Chapter 4 where the agent system has been described along with its fundamental algorithm. Step 6 has just been described above. The remaining steps are covered in greater detail below. After specifying the metrics used in the work, this chapter discusses the remaining steps of the systematic evaluation separately for the mathematical approach and for the simulation work, respectively.

5.1 Metrics

Metrics represent the criteria used to compare performance. The *performance* of our monitoring system is expressed in terms of two major indicators, the *traffic* incurred by the agents which perform monitoring operations and the overall monitoring *response time*. These are modelled using an approach similar to the one described in [Zegura 97] as discussed below.

Let us assume that the network is modelled by a connected graph, $G = (V, E)$, were V is the set of vertices (corresponding to nodes) and E is the set of edges (corresponding to communication links).

Traffic is modelled as the sum of packet hops incurred by monitoring – *i.e.*, the number of edges traversed by monitoring packets – multiplied by their respective packet size (in bits), b and packet rate, P_r. The traffic $T(v_1,v_2)$ between any two points $v_1, v_2 \in V$ subject to a bit rate B_r is $T(v_1, v_2)= B_r * d(v_1, v_2)$, where $B_r=b*P_r$, b is the size of a poll request or response (assumed comparable for simplicity), and $d(v_1,v_2)$ is the minimum distance between v_1 and v_2.

For simplicity, the *distance* in the network is measured using the hop-metric, in which each edge has unit weight. *Delay*, $D(v_1,v_2)$ between any two points, v_1 and v_2 is assumed to be proportional to their distance $d(v_1,v_2)$. The *response time* of a centralised monitoring system is the one involved in a complete ‘request-response’ operation by the monitoring station, involving all the monitored nodes. So it is the span of time elapsed for all the monitored entities to receive a request packet and for the monitoring station to receive all the responses.

In a decentralised monitoring system, there are more than one monitoring entities which are decentralised. Since each monitoring entity will be involved in monitoring a disjoint sub-partition of the whole monitored system, individual ‘request-response’ operations (for each monitoring entity) tend to be smaller than in the centralised case. However, in order to obtain the total response time, we need to add up the time for those monitoring entities to report to the central station.

Scalability is defined in [Casavant 94] as the ability to increase the size of the problem domain with a small or negligible increase in the solution's time and space complexity. For the purposes of our investigation, scalability is specifically defined as the ability to increase the number of monitored entities *N*, the polling rate P_r, the network diameter, *D(u)* (*i.e.*, the maximum distance between any two nodes in the network), or the number of MAs, *p* with a small or negligible decrease in performance. Hence, the *scale* of a given monitoring problem is measured in terms of *N*, P_r, *D(u)*, and *p*.

Adaptability is more difficult to quantify and hence to measure. In this thesis we have not defined a precise metric for adaptability since only preliminary experiments were carried out. We have expressed it in terms of the gradients observed in the two performance indicators (traffic and response time). Those gradients were calculated by measuring performance at steady state, then forcing a new state (*e.g.*, by causing congestion or link failures), waiting for the system to adapt to the new state and get to steady-state again, and finally recalculating performance.

Larger performance degradation corresponded to smaller adaptability, whereas more adaptable systems were associated with smaller performance degradation.

5.2 Analysis by Mathematical Modelling

The mathematical based work is organised in two parts. We first studied aspects related to the complexity of the proposed agent location algorithm to prove its viability for the proposed agent system (Chapter 6). We were not able to model the precise algorithm for general network topologies, but we could analytically find *upper bounds* on the performance indicators. This allowed the estimation of the time-scales of the agent deployment process and to draw conclusions on its scalability with respect to network size and number of agents.

In the second part of our analysis we started looking into the goodness of the proposed algorithm (Chapter 7). We look for particular conditions which lead to optimality or near-optimality, hoping that these are realistic enough to justify the algorithm. We, then, carry out a theoretical assessment of the agent system both at transient time and at steady state. In these cases, we manage to model the system mathematically. We model two different flavours of a centralised polling approach and carry out a comparative performance analysis between that approach and the agent solution.

The conclusions which we then draw about performance and scalability of our solution are valid under the given restraining conditions on the network topology. We discuss these conditions with respect to real network topologies.

A further step is to study the algorithm at steady state for general network topologies but this was difficult to carry out using this approach. The work was then continued switching to a simulation-based approach, as described later in this chapter (page 107).

5.2.1 Network Model

We have adopted the *network model* described in [Rescigno 97]. The network is modelled by a connected graph *G=(V,E)* with the vertices, *V* corresponding to nodes (processors) and the edges, *E* corresponding to communication links, which are modelled by the *All Ports-Full Duplex* communication model. This network model has been widely used because it generally reflects the hardware characteristics of networks (see references quoted in [Rescigno 97]).

Packet forwarding is modelled using an ‘on/off’ model for link transmission. Only one packet can traverse a single link at any time. A link is modelled as an atomic resource that is switched off during the transmission of a packet over that link and is switched on at the end of each packet transmission. Links are characterised by a fixed latency.

Packets arriving at a node are immediately forwarded through an outgoing interface to the relevant link only if that link is on. Packets are, otherwise, queued at the node. Infinite queues are assumed and, consequently, no packet dropping mechanisms are taken in consideration.

5.2.2 Fixed Parameters

Fixed *parameters* are the ones that affect the system performance and do not vary during the analysis [Jain 91]. The main parameters of the mathematical based analysis are reported in Table 5-1.

Fixed Parameter	Symbol	Value
Node degree (generic node)	δ	generic
Node degree of the monitoring station node	$d(u)$	generic
Size of polling request packet	b_b	generic
Size of polling response packet	b_r	generic
Size of agent notification packet	b_{not}	generic
MA size	MA_{size}	generic
MA transmission time	$TRANSM_{time}$	generic
MA forwarding time	$FORW_{time}$	generic
MA serialisation time	$SERIAL_{time}$	generic
MA de-serialisation time	$DESERIAL_{time}$	generic
MA cloning time	$CLON_{time}$	generic

Table 5-1. Fixed parameters of mathematical modelling.

5.2.3 Factors

Those parameters that are varied during the evaluation are named *factors* and their values are called *levels* [Jain 91]. In general, with both experimental and simulation-based techniques the list of factors, and their possible levels, is larger than the available resources will allow. It is, thus, necessary to limit the levels during the evaluation in order to study the system in its normal range of operation. For example, it will be difficult, if not impossible, to measure the impact of a particular agent migration policy on the overall monitoring traffic if we saturate the network with background traffic.

The mathematical approach usually allows a larger degree of freedom in constraining the factor levels because the system is modelled rather than physical. The main factors of the mathematical-based analysis are reported in Table 5-2.

Factor	Symbol	Level
Number of monitored nodes	N	unconstrained
Number of Mobile Agents	p	$0 \leq p < N$
Levels at which MAs reside in the spanning tree	L	$0 \leq L < R(u)$
Polling rate	P_r	unconstrained
Notification rate	N_r	unconstrained
Network radius	$R(u)$	$R(u) > 1$

Table 5-2. Factors of mathematical modelling.

5.2.4 Workloads

The workload consists of a list of service requests to the system [Jain 91]. When possible reference traffic models, or *benchmarks*, are used to compare the performance of a computer system. This was not possible in our case because, to the best of the author's knowledge, no benchmarking procedures have been specified yet for management systems.

In our case the workload is the traffic injected by the monitoring system to carry out monitoring tasks. We adopted the basic polling management model and, hence, monitoring tasks inject traffic which is proportional to the polling rate, P_r. For a given task and a given polling rate, the overall traffic and response time will generally vary for different agent configurations. Thus, our simple reference workload allows measuring the system performance under a range of input values specified by the levels of the various factors of Table 5-2.

5.2.5 Data Analysis and Presentation

Data analysis is useful to compare different alternatives through experimentation or simulation. In the case of mathematical modelling there is no need for that because the produced models are analytical functions that can be directly interpreted.

Results are directly presented in graphical format. The algorithmic complexity is obtained by studying the behaviour of the models when the various factors tend to infinity. Similarly, the system behaviour under near-optimal conditions is carried out by direct interpretation of the models.

Steady-state performance indicators are plotted against the number of MAs to assess the effects of varying agent configurations. Contour plots are used to capture performance as a function of two parameters instead of a single one.

5.3 Simulations Design

Mathematical modelling provided us with a first batch of information on the proposed agent system. We have discussed the limitations of this approach which have led us to complement that study with simulation work.

By simulations we were able to study the agent system under more realistic conditions, that is for realistic network topologies and over realistic network and transport layers. The aim was to assess the goodness of our algorithm, that is its ability to place MA near-optimally. We describe below the approach followed in the simulation work.

5.3.1 Simulation Environment

In order to simulate IP network and protocol behaviour we have adopted the NS simulator (version 2) from U.C. Berkeley/LBNL [NS, Fall 99] and extended it with Mobile Agent capabilities. Agent migration and cloning have been implemented along with the actual agent location algorithm, which is incorporated in each agent. This algorithm has been optimised to minimise the total incurred monitoring traffic, according to what has been presented in Chapter 4.

In NS arbitrary network topologies, composed of routers, links and shared media can be defined. Several protocols, such as TCP and UDP, are available and various types of applications can be simulated. Among them are FTP, Telnet, and HTTP, which use TCP as the underlying transport protocol, and applications requiring a constant bit rate (CBR) traffic pattern, which use the UDP transport protocol. The CBR traffic generator injects traffic according to a deterministic rate, and a fixed packet size. Optionally, some randomising dither can be enabled on the inter-packet departure intervals.

Other traffic generators provide an exponential on/off distribution and a Pareto on/off distribution, respectively. The former sends packets at a fixed rate during ‘on’ periods and no packets during ‘off’ periods. Both ‘on’ and ‘off’ periods are taken from an exponential distributions. In the latter case, ‘on’ and ‘off’ periods are taken from a Pareto distribution and can be used to generate aggregate traffic that exhibit long range dependency. In NS it is also possible to inject traffic according to a trace file. This might have been created by monitoring traffic traces in a real network.

Point-to-point bi-directional links are characterised by bandwidth, delay, and queue type. Queues represent locations where packets may be held or dropped. Packet scheduling refers to the decision process used to choose which packet should be serviced or dropped. Buffer management refers to any particular discipline used to regulate the occupancy of a particular queue. NS includes support for several algorithms such as drop-tail (FIFO) queuing, RED buffer management, and different variants of fair queuing.

The routing model can be static or dynamic. The distance vector algorithm is used for unicast routing. A prototype implementation of multicast routing is also available.

The simulator is event-driven and runs in non-real-time fashion. Packet losses are simulated by buffer overflows in routers. Buffer overflows are in fact normally the main cause of packet loss in the Internet. There is also support for error models other than losses through buffer overflow, but these are not in our simulations.

In NS there are two ways of collecting output on a simulation. The first is to create traces which can record each individual packet as it arrives, departs, or is dropped at a link or queue. To prevent large trace files it is possible to specify what should be traced in terms of links, queues, type of packets, etc. The second approach is based on *monitors*, which can count various interesting quantities such as packet arrivals, departures, etc. *Flow monitors* can count on a per-flow bases rather than on a packet bases.

A network animator tool, called NAM, can be used for viewing network simulation traces, providing a graphical means of studying the system [NAM]. More elaborate statistics can be achieved through specialised scripts which directly process trace files.

The adopted network simulator is implemented in C++ and Object TCL and is easily extensible with extra capabilities such as new protocols, routing algorithms, queuing disciplines etc. We have realised a new layer which provides the basic MA capability and have implemented the proposed agent-based monitoring system on top of this layer. A new MA class was introduced, including methods to start/stop MA execution; migrate MA; clone MA; destroy MA. MAs encapsulate the proposed agent location algorithm in both its flavours, *i.e.* with or without cloning capabilities. This algorithm has been optimised to minimise the total incurred monitoring traffic.

5.3.2 Description of Simulations

5.3.2.1 Comparative Performance Analysis

We compared performance and scalability of the agent system with static, centralised monitoring. The monitoring model adopted for centralised monitoring is polling-based. The agent system is partly polling-based (each agent collects monitoring information through polling), partly notification based (agents can periodically send reports to the central monitoring station), and partly event-driven (agents may send alarms to the monitoring station).

The adopted scalability indicators are: polling rate; number of monitored nodes; network diameter; and number of MAs. Performance indicators are: total incurred traffic, traffic incurred around monitoring station, and average and maximum response time.

Simulations involved the study of each of the performance indicators against all the scalability indicators. This was achieved by keeping each time 3 of the 4 scalability indicators constant and varying the other progressively. For instance, performance versus polling rate was studied by keeping number of monitored nodes, network diameter, and number of MAs constant.

At least four different points were computed in the X axis (*i.e.* the scalability indicator axis). Each point was initially computed by repeating the simulations 20 times under identical conditions except for the network topology. The topology was generated with an appropriate topology generator tool and procedure as detailed below. The 20 measurements were then treated statistically as detailed below. This procedure required an enormous amount of computational time. Initial statistical treatment of the results showed that 10 repetitions were sufficient for statistical significance; hence subsequent simulations were only repeated 10 times instead of 20.

For each new network, the MA deployment process was re-initiated in order to compute the initial agent location. The parameter responsible for the final number of agents was increased to compute new configurations and study the scalability of the algorithm versus the number of agents. 6 agent configurations were computed for each network and the process was repeated 10 times for statistical purposes.

Agents were actually deployed in the simulated network and monitoring tasks executed for a span of time sufficient to reach steady state. It was observed that 5 real-time simulation seconds were sufficient for that purpose. Simulation traces for the whole duration of the simulation were stored in files. Scripts to automate post-processing of those trace files were created in TCL. Scripts for producing simple statistics were also used. The statistical tool ORIGIN was finally utilised to carry out the statistical analysis of the results as detailed below. Scripts to import the simulation data into origin were also created to automate to a large extent this very time-consuming process.

5.3.2.2 Distance from Optimality

A second important part of the simulations was aimed at studying the distance from optimality of the agent locations computed by the proposed algorithm. This was done by comparing the overall performance achieved by the agent system against the hypothetical case in which those locations were computed optimally.

Provably near-optimal locations were computed using the software package SITATION [Daskin 95] configured to run the *lagrangian* location algorithm. The computed locations were then fed to the network simulator described above to deploy agents and run the same monitoring tasks executed by the agent system in which locations were computed with the proposed algorithm.

The proposed agent system was also compared against the case in which agent locations were generated randomly to provide an estimate of the distance from randomness.

The comparison between the three systems was carried out by adopting *topological* rather than *physical* performance indicators. The first indicator is the network *total hop-distance*, directly related to the total steady-state traffic. The second indicator is the *maximum weighted distance*, directly related to response time.

For statistical significance, comparisons were carried out each time on 10 randomly generated network topologies belonging to the same family (*i.e.* with comparable topological characteristics).

5.3.2.3 Adaptability

An initial study of the adaptability, or self-reconfigurability, of our agent system was carried out by simulating various conditions in which link failures led to increased traffic and response time. We deployed the agent system before the failures and carried out a performance analysis following the procedure already described above. Then, we generated link failures at random locations but in the vicinity of the central monitoring station. We assessed the costs associated with agent migration and, finally, measured traffic and response time after re-configuration.

A comprehensive study of adaptability would have required a large amount of simulations, following a methodological approach similar to the one used above. Due to lack of time a simpler approach was adopted aimed at providing mainly an initial feeling of the adaptability of the system, rather than a complete study.

We have simulated a simple scenario in which 2 links located in the vicinity of the central monitoring station fail. Traffic and response time were measured before the failure. After the failure, the routing protocol readjusted the routing tables and full connectivity was achieved. In addition, the agent system reconfigured itself by relocating some of the agents. Steady-state traffic and response time were measured again. Simulations were subsequently repeated for 10 different randomly generated topologies characterised by comparable topological features. Each time a couple of faults was generated randomly and results were averaged.

5.3.3 Software Parameters

The most significant parameters of the network simulator are reported in Table 5-3.

Fixed Parameter	Value
Network protocol	IP
Transport protocol	UDP
Routing protocol	dynamic Distance Vector
Traffic generator	CBR
Link bandwidth	automatically assigned by topology generator
Link delay	automatically assigned by topology generator
Link queue	drop tail
Node degree (generic node)	automatically assigned by topology generator
Real-time simulation time	5 seconds
Type of topology	Transit-stub

Table 5-3. Network simulator parameters.

It is worth mentioning that Transit-stub topologies have been selected because they are considered to reflect the properties of real inter-networks [Zegura 96, Zegura 97]. This class of topology can be viewed as a collection of interconnected *routing domains*, which are groups of nodes that are under a common administration and share routing information. Each routing domain can be classified as either a *stub* domain or a *transit* domain (Figure 5-2). In a stub domain the path connecting any two nodes *u* and *v* goes through that domain only if either *u* or *v* is in that domain. Transit domains do not have this restriction; their purpose is to interconnect stub domains efficiently. Stub domains can be further classified as single- or multi-homed. Multi-homed stub domains have connections to more than one transit domain. Single-homed stubs connect to only one transit domain. Some stubs have also links to other stubs.

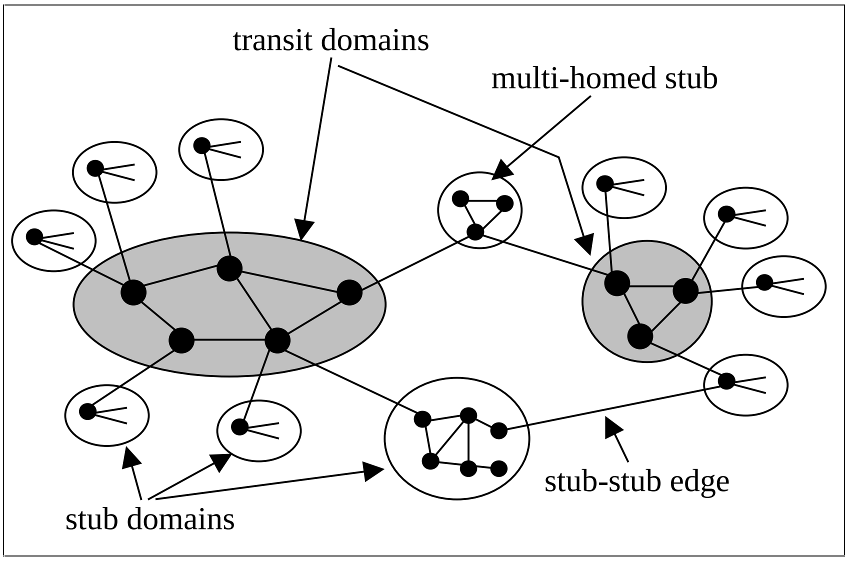


Figure 5-2. Transit-stub topology (from [Zegura 97]).

The most significant parameters of the agent-based monitoring system are reported in Table 5-4.

Fixed Parameter	Value
Size of polling request packet	100 bytes
Size of polling response packet	250 bytes
Size of agent notification packet	50 bytes
MA size	1000 bytes
MA serialisation time	100 msec
MA de-serialisation time	200 msec
MA cloning time	100 msec

Table 5-4. Agent-based monitoring system fixed parameters.

5.3.4 Hardware Parameters

The most significant parameters of hardware and operating system used to run the simulations are reported in Table 5-5.

Fixed Parameter	Value
Processor	Pentium 150 MHz
RAM Memory	48 Mbytes
Available disk space	1 Gbyte
Operating System	Linux 2.0.34
Swap space	100 Mbytes

Table 5-5. Hardware and operating system parameters.

5.3.5 Factors

The most significant simulation factors are reported in Table 5-6. It should be noted that the range of these values was obtained from all the simulations. Particular simulations are characterised by smaller ranges in general. The actual ranges are indicated on a case-by-case bases in Chapter 8.

Simulation Factor	Level
Number of monitored nodes	16 – 64
Average network diameter	7 – 8.5
Percentage of Mobile Agents	0 – 0.7
Polling rate	0.2 – 6 polls per sec
Notification rate	0.1 – 3 packets per sec
Average number of link failures (adaptability only)	2
Average node degree	2.56 – 11.7
Hop-diameter	6.3 – 10
Average hop-depth	5.07 – 7.92
Length diameter	101 – 224
Average length-depth	78.29 – 188
Number of biconnected components	5 – 29.1

Table 5-6. Simulations factors.

5.3.6 Simulation Complexity

The simulation of the agent location process involved relatively long computational times, as can be illustrated by looking at the number of levels and combinations of the relevant simulation factors (Figure 5-3). These are: 1) the number of monitored nodes, N; 2) the percentage of agents, p/N; and 3) network diameter.

```
Randomise 10 times {                                //    10 *
   For each N { 16, 25, 32, 50, 64 }                //     5 *
       For each p/N { 0.1, 0.25, 0.4, 0.55, 0.7 }  //     5 *
           For each D(u) { 7, 8, 8.5 }              //     3 *
               compute agent location               //     2 (hours)
}                                                   // =1500 hours
```

Figure 5-3. Illustration of simulation complexity.

Thus, by considering that the computation of a single agent configuration required an approximate, average time of 2 hours, we can see that the overall computational time for producing the required 750 agent configurations is in the order of 1,500 hours.

Simulation time was actually larger than this figure would suggest, because of the required time to run the actual monitoring tasks and produce simulation traces for increasing values of polling rate (P_r = 0, 0.2, 0.6, 1, 2, 4, 6). Simulation traces including all events of 5 real time seconds were very large and required a total of several days to process and analyse with specialised TCL scripts. Negligible time was required to process the results and convert them in a format that could be fed to the ORIGIN statistical tool. Significant effort was spent instead on the production of diagrams, which required significant manual intervention.

The above estimates should have been doubled if we had replicated the whole simulations for the two flavours of the agent location algorithm. This was not necessary because it was verified that the two algorithms produced the same results in terms of agent configuration and network partitioning (see Section 5.3.8.7).

5.3.7 Workloads

Similarly to what observed in Section 5.2.4 above, no standard or *de facto benchmarks* are available for assessing management systems. We adopted the basic polling management model and hence monitoring tasks inject traffic which is proportional to the polling rate, P_r. For a given task and a given polling rate, the overall traffic and response time will generally vary for different agent configurations.

Polling request packets (originated in MAs and directed to their monitored nodes) and notification packets (originated in MAs and targeting the central monitoring station) were generated using the CBR traffic generator. Response packets were created directly by the nodes which were equipped with a protocol similar to PING. Polling and notification rate ranges and packet sized are indicated in Table 5-6.

5.3.8 Simulation validation

Validation refers to ensuring that the assumptions used in developing the model are reasonable in that, if correctly implemented, the model would produce results close to those observed in real systems [Jain 91]. Jain suggests that model validation consists of validating three key aspects of the model:

1. Assumptions
2. Input parameter values and distributions
3. Output values and conclusions

Each of these three aspects may be validated in three possible ways:

1. Expert intuition
2. Real system measurements
3. Theoretical results

Although this leads to nine possible validation tests, in practice it may not be feasible to use some of these possibilities. Jain argues that, in fact, in most real situations none of the nine possibilities may be feasible. He justifies this statement with the argument that “simulation, being a time-consuming effort, is resorted to only if no other reliable means of finding the same information exists”.

This is the situation of this thesis work in which simulations have been carried out to complement the theoretical work rather than to cross-validate the two approaches. In fact, mathematical modelling and simulations cover orthogonal problems.

Validation through comparison with real systems is the most reliable and preferred way. However, Jain argues again that “this is often unfeasible either because the real system may not exist or because the measurements may be too expensive to carry out”. This was the case of the thesis work.

Since direct comparison with theoretical models or real systems was unfeasible validation was carried out following, again, Jain’s advice. He suggests to validate individual aspects or sub-components of the simulator aiming at increasing the confidence in the simulation model and approach, rather than pursuing the “myth of fully validated model”. The next sub-sections discuss such a validation approach.

5.3.8.1 Network and Transport Layers

Network and transport layers were not subject to any validation since the adopted network simulator was not modified at those levels. The NS network simulator is well known and widely used in the networking community and several works based on this simulator have already been published. An example is [Hanle 98].

5.3.8.2 Network Topology Generation

A crucial point was to assess the proposed agent system for a set of realistic network topologies, composed of routers, links, and hosts. These have been generated using the GT-ITM topology generator [GT-ITM]. The methodology followed to generate the network topologies is the one proposed by Calvert and Zegura, which is widely acknowledged in the research community [Zegura 96, Calvert 97, Zegura 97]. The methodology is, therefore, not subject of validation. However, it was necessary to verify that the produced topologies reflected the requirements for the various individual experiments. For instance, families of topologies characterised by relatively small or negligible variation in average node degree were necessary. More generally, experiments assessing scalability required the gradual increase in network size. This was done by keeping the number of nodes constant when increasing network diameter and *vice versa*. Scripts to automate the verification of all topological conditions were written in TCL. Those scripts were used to help validate the production of network topologies.

Network topologies were also visually inspected using the NAM network animator tool [NAM]. Example 50-node topologies produced randomly but belonging to the same topological family are reported in Figure 5-4. It can be noticed that the actual topologies are significantly different despite other topological features such as average node degree and network diameter being comparable.

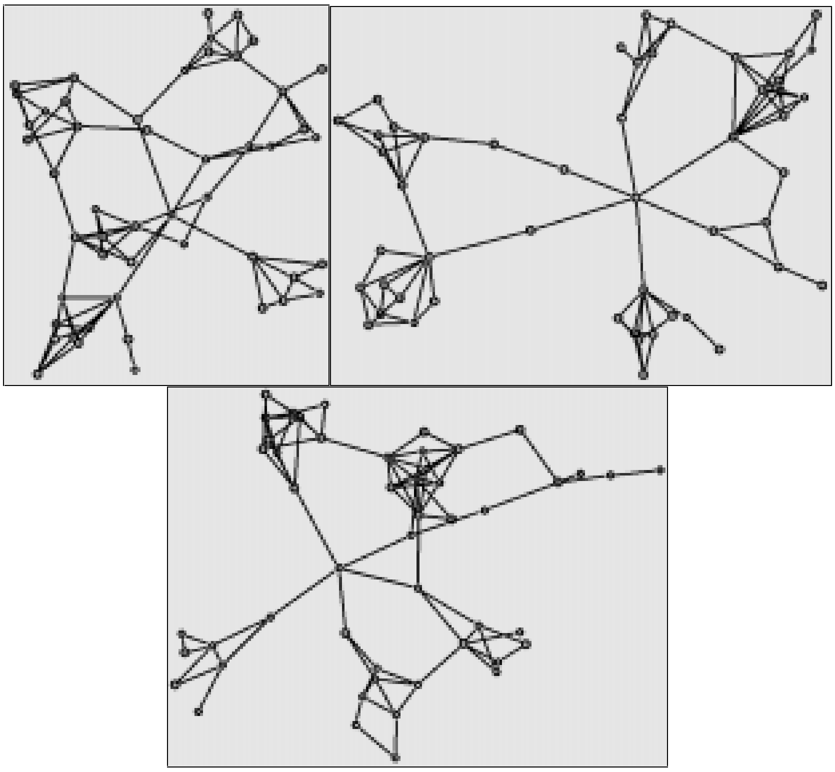


Figure 5-4. Example 50-node randomly generated network topologies.

5.3.8.3 Random Number Generators

Randomisation is widely used in the simulations and it was, therefore, important to produce valid random numbers. Random generators were used, for instance, to generate different topologies characterised by comparable topological features; to randomise the location of the central monitoring station; to randomise the location of link failures, and so on.

The desired properties of a random generator function are as follows [Jain 91]:

- It should be efficiently computable, since simulations typically require several thousand random numbers in each run.
- The period should be large, since a small period may cause the random-number sequence to recycle.
- The successive values should be independent and uniformly distributed. The correlation between successive numbers should be small, since correlation (if significant) indicates dependence.

Apart from the choice of the random generator, the seed selection is crucial as well. The seeds used in random-number generation should not affect the final conclusion. In order to guarantee that the simulator produced similar results for different seed values, we have run the same simulations with different seed values.

The NS simulator provides special support to random numbers. It implements the generator designed by Park and Miller [Park 88]. It also provides libraries to help selecting the seed. A table of 64 known good seeds for that generator is also provided. Therefore, having made use of the library provided with the simulator we could guarantee validity of the randomisation processes.

5.3.8.4 Continuity, Degeneracy, and Consistency Tests

Continuity tests consist of running the simulation several times for slightly different values of input parameters ([Jain 91] p.418). For any one parameter, a slight change in input should generally produce only a slight change in output. Any sudden change in the output may be due to modelling errors and should be investigated. For instance, the performance plots versus scalability should exhibit monotonic or semi-monotonic behaviour.

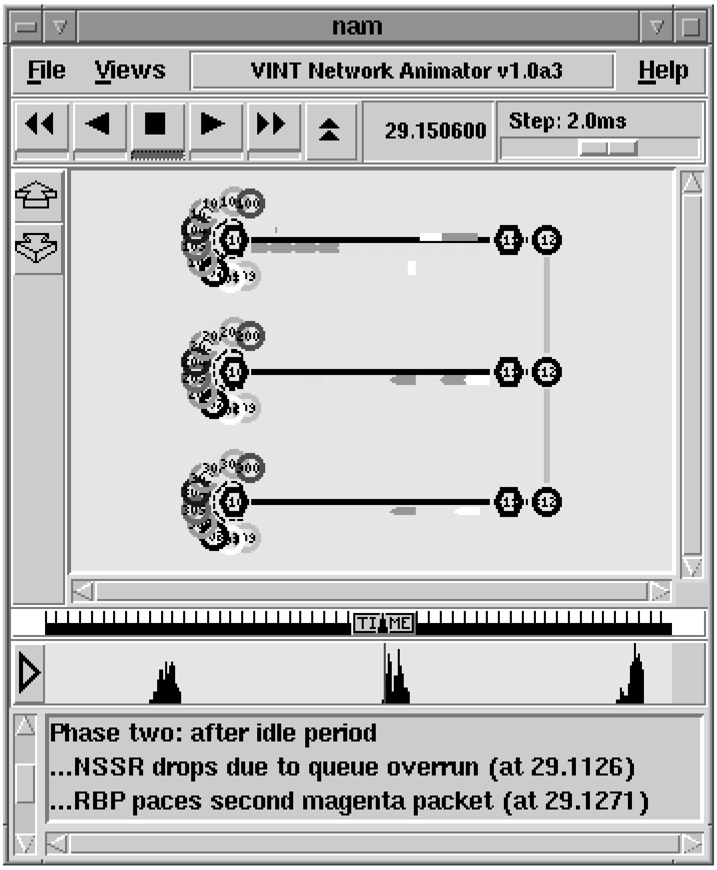
Degeneracy tests consist of checking that the model works for extreme (lowest or highest allowed) values of system, configuration, or workload parameters ([Jain 91] p.419). For instance, the agent monitoring system with zero agents should lead to the same performance as its centralised counterpart. Another example is the situation in which the number of agents is equal to the number of nodes. In this case, agents should be deployed uniformly, with one agent per node.

Consistency tests consist of checking that the model produces similar results for input parameter values that have similar effects ([Jain 91] p.420). For instance, two different agent configurations computed by the location algorithm should lead to comparable levels of traffic and response time.

To increase the confidence in the validity of the simulations, continuity, degeneracy, and consistency tests were carried out. Some of them were done by visual inspection; that is by observing the behaviour of the system using the network animator. Others were carried out through simple scripts written in TCL.

5.3.8.5 Agent Environment

The agent mechanisms constructed on top of the network simulator were tested and verified performing extensive simulations and through routines to automate the analysis of trace files. The aim was to make sure that agent creation, destruction, cloning, and migration were functioning correctly. In addition to the analysis of agent traces, the NAM graphical network animator was used to actually visualise agent behaviour in real time [NAM]. A simple snapshot showing packets destined to different target nodes in different colour is reported in Figure 5-5.



Phase two: after idle period
...NSSR drops due to queue overrun (at 29.1126)
...RBP paces second magenta packet (at 29.1271)

Figure 5-5. Snapshot of the NAM network animator.

5.3.8.6 Agent System Behaviour

Upon validating the basic MA mechanism, the actual agent location algorithm had to be validated as well. This was done again with the help of the NAM network animator. Various network topologies were adopted and for each of them the agent location algorithm was both calculated by hand and simulated in order to verify correctness. It was not possible to find a more general validation mechanism because of the complexity and distributed nature of the agent location algorithm.

To increase the level of confidence, scripts for the analysis of MA traces were written. Various tests were carried out. For instance it was verified that agents were actually monitoring a disjoint subset of monitored nodes and that the node-to-agent allocation mechanism was correct. It was also tested that all nodes were finally allocated to a single agent to verify that the network partitioning algorithm was correct.

The goodness of the agent locations was also verified by comparing the performance of the agent system with the case in which agent locations were randomly assigned and showing significant statistical difference between the two.

5.3.8.7 Congruency between the two Flavours of the Agent Location Algorithm

One of the claims expressed in Chapter 4 is that the two different flavours of the proposed agent location algorithm (with and without cloning) lead to the same agent configuration. This can be expected because both flavours implement the same basic algorithm. The actual computational approach is different (centralised versus distributed) but the steps are the same. Nevertheless, it was necessary to validate this claim with simulation work. This was done by running the two algorithms separately for the same set of network topologies.

20 network topologies were generated with different topological characteristics and size (in terms of nodes and diameter). In each case both algorithms were run to compute the agent location for an increasing percentage of MAs (10%, 25%, 40%, 55%, and 70%). It was verified that agent locations and network partitioning were identical in both cases.

The importance of these tests should be emphasised here. In fact, congruency of the two algorithms allowed us to reduce significantly the overall simulation time for the experiments described in Chapter 8. In fact, it was not necessary to repeat all those experiments twice, once for each location algorithm, since the steady-state configuration and behaviour of the two would have been identical.

5.3.8.8 Agent Migration overheads

Agent migration overheads were accounted for in the simulator in a parametric fashion. The actual values for agent serialisation and de-serialisation time and typical agent size were selected upon evaluating the respective values available in the literature and by performing extra measurement in a real agent system.

The extra measurements were required because little data is available in the literature and, when available, the values refer to general-purpose MA platforms. These are characterised by overheads that are larger than what could be envisioned for network management oriented MA platforms.

Experiments aimed at providing a realistic order of magnitude for the overall agent migration time were carried out on the general-purpose Grasshopper MA platform and on CodeShell, an MA platform optimised for constrained mobility. The results are shown in Figure 5-6.

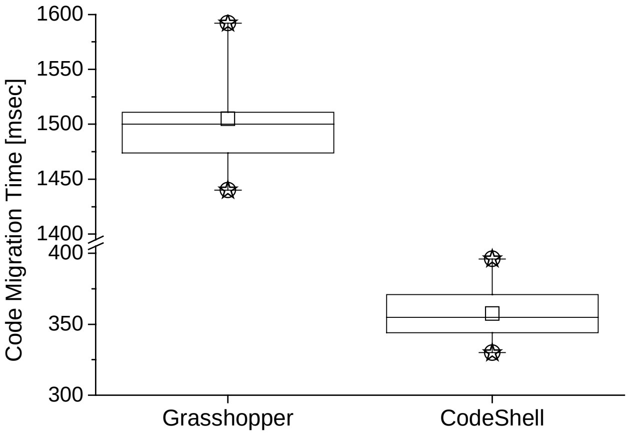


Figure 5-6. MA Migration time measurements on real MA platforms.

Results are reported in the form of statistical box charts (see section below for interpretation). Boxes include mean values (small square) and median values (line).

MA size varies more critically depending on what is implemented in the agent. So it was more difficult to come up with realistic features. Typical agent sizes were inferred from published data [Knight 99].

5.3.9 Statistical Data Analysis and Presentation of Results

As already mentioned above, simulations have been randomised with respect to the various factors and repeated a sufficient number of times to ensure statistical significance of the results. Initially, simulations were repeated 20 times for each randomisation. It was noticed that statistical significance was achieved with 10 repetitions.

Statistical significance was analysed with Microcal ORIGIN, an advanced statistical tool chosen as an alternative to Microsoft Excell. The former was found to provide better support for statistical analysis and data presentation, as well as increased flexibility.

All simulation data were fed to ORIGIN to generate statistical box charts similar to those of Figure 5-6. Statistical box diagrams were used because they are well suited to reporting data generated over different runs of the simulations. They summarise the spread of data in a simple diagram that portrays mean, median, first and third quartiles, and range ([Lewis 99] pp.117-118). Each *Y* column is represented as a separate box. The *Y* axes reports statistical indicators rather than the actual measured values, which correspond to the configuration parameter reported in the *X* axis. Boxes are determined by the 25th and 75th percentiles (first and third quartiles respectively); whiskers are determined by the 5th and 95th percentiles; the little squares represent mean values; horizontal lines represent median values; circles depict lowest and highest values; and stars denote the 1-99% range of the values.

Statistical box charts allow drawing conclusions on statistical significance of results. Partial overlapping between different boxes denote that the indicators of central tendency (mean and median) in the two boxes are statistically comparable. Conversely, non-overlapping boxes denote significantly different results from a statistical point of view.

Two different indices of central tendency were adopted. Mean and median were used to summarise traffic and response time factors, respectively. Data variability was summarised with 25th and 75th percentiles. Best-fit functions on the indices of central tendency were then computed with ORIGIN for a direct presentation of the simulation results.

5.4 Conclusions

In this chapter we have described the method used to examine the hypothesis and assess the proposed agent-based monitoring system presented in Chapter 4. We have contextually tried to provide motivations for some of the choices adopted, discussing issues and difficulties encountered during the experimental work.

The initial idea was to model the whole system, first mathematically and, then, by simulation. This would have been an ideal approach since it would have allowed a full validation of the simulations, with the advantage of being more realistic and, hence, credible than theoretical studies.

On the way we found the hurdle of modelling a complex distributed system in a mathematical fashion and sought alternative ways of examining the hypothesis. Jain’s seminal book on system performance analysis inspired a new path. The evaluation was finally carried out using a hybrid mathematical and simulation-based approach. The two examined orthogonal aspects of the proposed agent system. Validation of the simulations was carried out by validating sub-components and assumptions of the developed simulator.

The rest of the thesis is dedicated to the presentation and discussion of the experimental work according to the methodology presented here.

Chapter 6

Theoretical Evaluation of Agent Deployment Under General Conditions

This chapter initiates the evaluation of the proposed approach to dynamic distributed monitoring based on agents. The focus is on the theoretical evaluation of the two flavours of the agent location algorithm presented in Chapter 4.

The evaluation is carried out for the case of general network topologies. A schematic representation of the focus of this chapter is depicted in Figure 6-1. First, the asymptotic complexity of those algorithms is studied in order to assess their scalability and typical computational time. Then, agent deployment is studied and, again, conclusions on scalability and typical times are drawn. Finally, a similar study covers the evaluation of the traffic incurred by agent deployment.

The study of algorithmic complexity and agent deployment time and traffic reflects the investigation of the *transient* behaviour of the proposed monitoring system. At transient time – *i.e.*, until all agents have been deployed – the monitoring system cannot yet operate. Thus, the identification of scalability and typical traffic and times involved in agent location and deployment is an essential part of the evaluation of the viability of the proposed agent solution. In addition, the overall transient time and incurred traffic are overheads as far as the monitoring system is concerned. In fact, in static monitoring systems the monitoring 'logic' is not deployed dynamically, since it is hard-wired into the system.

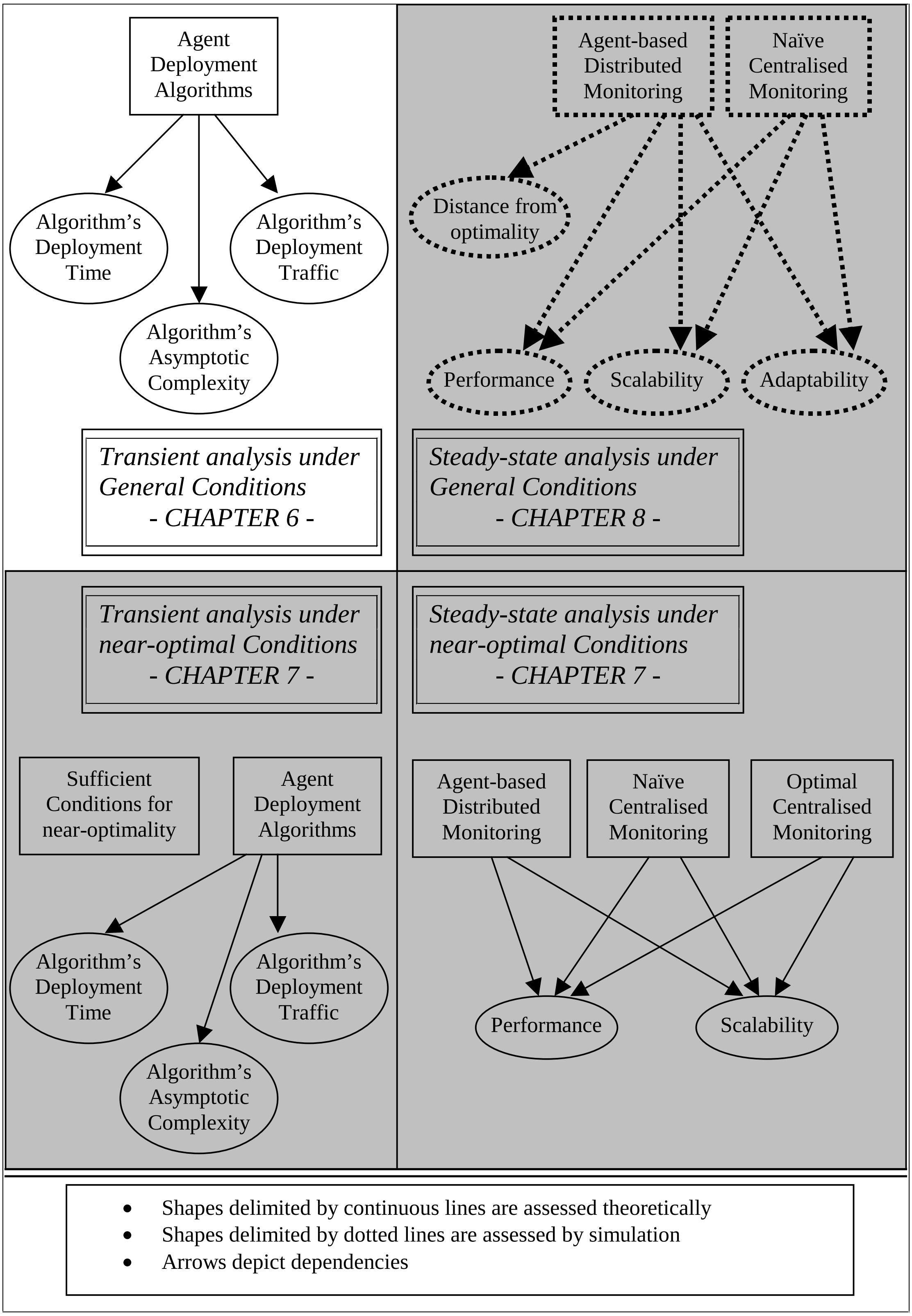

Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Asymptotic Complexity
Algorithm's Deployment Traffic
Transient analysis under General Conditions - CHAPTER 6 -
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Distance from optimality
Performance
Scalability
Adaptability
Steady-state analysis under General Conditions - CHAPTER 8 -
Transient analysis under near-optimal Conditions - CHAPTER 7 -
Steady-state analysis under near-optimal Conditions - CHAPTER 7 -
Sufficient Conditions for near-optimality
Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Optimal Centralised Monitoring
Performance
Scalability
• Shapes delimited by continuous lines are assessed theoretically
• Shapes delimited by dotted lines are assessed by simulation
• Arrows depict dependencies

Figure 6-1: Schematic representation of the focus of Chapter 6.

Therefore, this chapter assesses the complexity and time-scales of the agent deployment process (*transient* behaviour). It does not assess the goodness of the resulting agent locations from the perspective of a distributed agent-based monitoring system (*steady state* behaviour). This is done by simulation and is the subject of Chapter 8. Finally, the study of sufficient conditions for near-optimality is reported in Chapter 7, which also includes a theoretical treatment of both *transient* and *steady-state* behaviour of distributed monitoring under the given near-optimal conditions.

An initial study of the asymptotic complexity of agent location algorithm reported herein has been published in [Liotta 98a].

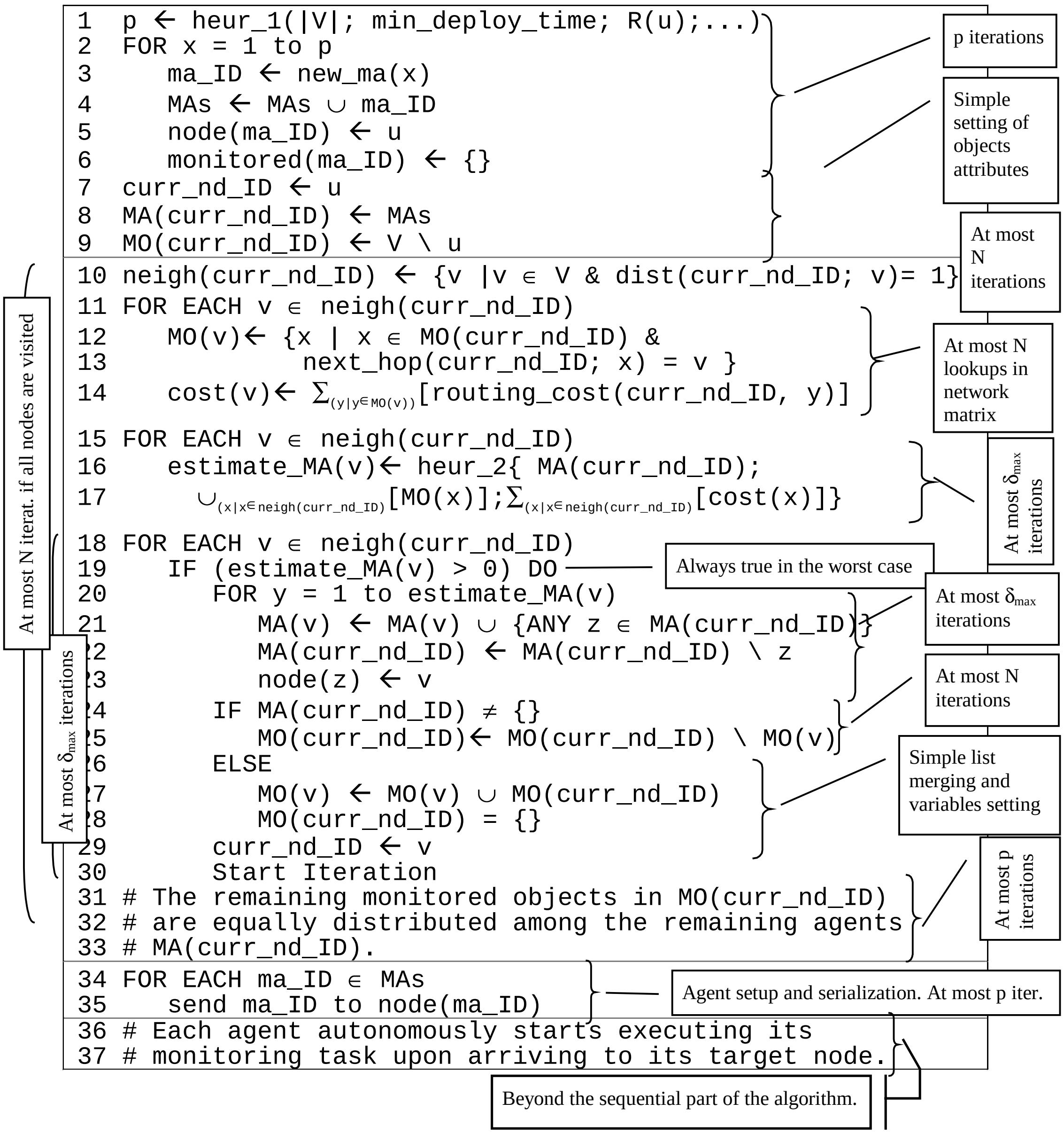
6.1 Asymptotic Complexity of the Agent Location Algorithms

The asymptotic complexity of the two flavours of the agent location algorithm described in Chapter 4 is evaluated here by considering the upper bound on the time taken for the algorithms to complete when the scale of the problem grows. The scale factors considered are the number of nodes, *N* the network radius, *R(u)* and the number of agents, *p*.

6.1.1 Asymptotic Complexity of the Centralised Location Algorithm

The centralised algorithm described in Chapter 4 (Section 4.1 page 76) is reported in Listing 6-1 in which additional boxes highlighting the complexity of the various portion of the algorithm have been introduced. The definitions of the various variables are reported in Table 4-1, page 82.

This is a iterative algorithm executed sequentially at the monitoring station. Lines 1-9 are executed only at initialisation time. In particular, Lines 1-6 involve *p* iterations for the creations and setting up of the required agents.



```
1  p ← heur_1(|V|; min_deploy_time; R(u);...)
2  FOR x = 1 to p
3     ma_ID ← new_ma(x)
4     MAs ← MAs $\cup$ ma_ID
5     node(ma_ID) ← u
6     monitored(ma_ID) ← {}
7  curr_nd_ID ← u
8  MA(curr_nd_ID) ← MAs
9  MO(curr_nd_ID) ← V \ u
10 neigh(curr_nd_ID) ← {v |v $\in$ V & dist(curr_nd_ID; v)= 1}
11 FOR EACH v $\in$ neigh(curr_nd_ID)
12    MO(v)← {x | x $\in$ MO(curr_nd_ID) &
13             next_hop(curr_nd_ID; x) = v }
14    cost(v)← $\Sigma_{(y|y\in MO(v))}$[routing_cost(curr_nd_ID, y)]
15 FOR EACH v $\in$ neigh(curr_nd_ID)
16    estimate_MA(v)← heur_2{ MA(curr_nd_ID);
17      $\cup_{(x|x\in neigh(curr\_nd\_ID))}$[MO(x)];$\Sigma_{(x|x\in neigh(curr\_nd\_ID))}$[cost(x)]}
18 FOR EACH v $\in$ neigh(curr_nd_ID)
19    IF (estimate_MA(v) > 0) DO
20       FOR y = 1 to estimate_MA(v)
21          MA(v) ← MA(v) $\cup$ {ANY z $\in$ MA(curr_nd_ID)}
22          MA(curr_nd_ID) ← MA(curr_nd_ID) \ z
23          node(z) ← v
24       IF MA(curr_nd_ID) $\neq$ {}
25          MO(curr_nd_ID)← MO(curr_nd_ID) \ MO(v)
26       ELSE
27          MO(v) ← MO(v) $\cup$ MO(curr_nd_ID)
28          MO(curr_nd_ID) = {}
29       curr_nd_ID ← v
30       Start Iteration
31 # The remaining monitored objects in MO(curr_nd_ID)
32 # are equally distributed among the remaining agents
33 # MA(curr_nd_ID).
34 FOR EACH ma_ID $\in$ MAs
35    send ma_ID to node(ma_ID)
36 # Each agent autonomously starts executing its
37 # monitoring task upon arriving to its target node.
```

p iterations
Simple setting of objects attributes
At most N iterations
At most N lookups in network matrix
At most δ_{max} iterations
Always true in the worst case
At most δ_{max} iterations
At most N iterations
Simple list merging and variables setting
At most p iterations
Agent setup and serialization. At most p iter.
Beyond the sequential part of the algorithm.
At most N iterat. if all nodes are visited
At most δ_{max} iterations

Listing 6-1: Analysis of the centralised agent location algorithm presented in Chapter 4.

The iterative part of the algorithm is included in Lines 10-30 which are executed at most *N* times, if all nodes are to be visited before the location algorithm converges. Line 10 involves a number of iterations that depends on the way the network is internally represented in the program. For instance if each node is implemented by an object which includes the list of neighbour nodes as an attribute, then Line 10 is a simple retrieval operation. Alternatively, if the network is more simply represented by a matrix, the function that computes the neighbour nodes may require up to *N* iterations.

Lines 11-14 require at most *N* lookup operations into the network data structure. Lines 15-17 require at most δ_{max} iterations, whereby δ_{max} is the maximum node degree in the network. It is worth mentioning that δ_{max} does not usually increase significantly with *N* in typical networks and is usually much smaller than *N*.

Lines 18-29 are iterated at most δ_{max} as well. The condition expressed in Line 19 adds extra computation and thus in order to consider the worst case we assume it is always true. In this case Lines 20-23 involve at most δ_{max} iterations; alternatively Lines 24-25 involve at most *N* iterations and Lines 26-28 require simple operations on lists. Finally, Line 29 is an operation on a simple variable and Line 30 initiates the iteration process.

The remaining instructions are executed only once after the iteration process is finished. Lines 31-33 involve at most *p* iterations that will make sure that all nodes will be monitored by the agents. Lines 34-35 will involve *p* iterations for the individual setting up and serialisation of the agents. After that, the sequential part of the algorithm is terminated since each agent is migrated to its destination where it will operate independently from the other ones.

Lines	**Computational Contribution (iterations)**
1-9	p
10-30	N
10	N
11-14	N
15-17	δ_{max}
18-29	δ_{max}
20-23	δ_{max}
24-25	N
27-29	constant
31-33	p
34-35	p
Total	**$O(p+N^2)$**

Table 6-1: Computational contribution for the algorithm of Listing 6-1.

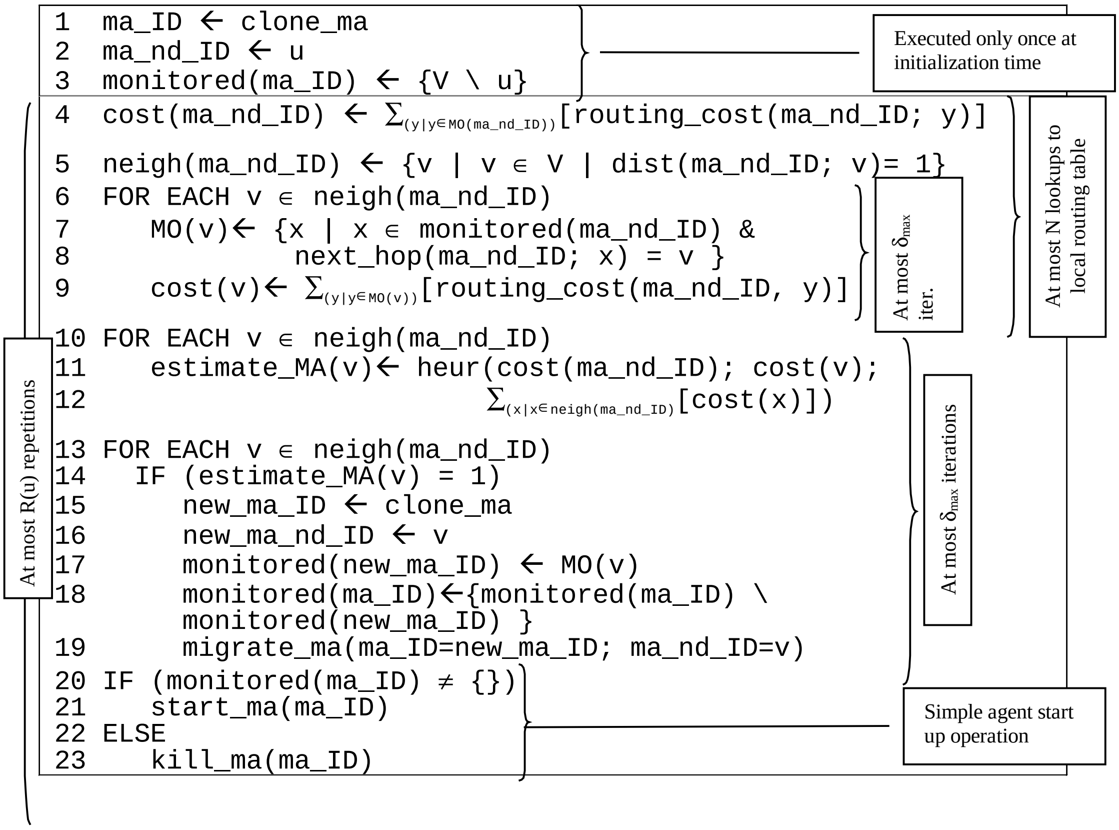
Based upon the above analysis the main contributions to the algorithm computational complexity are summarised in Table 6-1 from which we can conclude that an upper bound on the asymptotic complexity of the algorithm is

$$O\{p + N * [N+N+\delta_{max}+\delta_{max}*(\delta_{max}+ N)] +p+p \} \propto O(p+ N^2).$$

6.1.2 Asymptotic Complexity of the Distributed Location Algorithm

The distributed version of the algorithm described in Chapter 4 (Section 4.2, page 84) is reported in Listing 6-2 in which additional boxes highlighting the complexity of the various portion of the algorithm have been introduced. The definitions of the various variables are reported in Table 4-3, page 90.

For the calculation of the algorithm asymptotic complexity we consider the part of the algorithm that is inherently sequential. The part of the algorithm that can be executed in parallel is less crucial since execution time for this part can be reduced by addition of processors.



```
1  ma_ID ← clone_ma
2  ma_nd_ID ← u
3  monitored(ma_ID) ← {V \ u}
4  cost(ma_nd_ID) ← $\sum_{(y|y\in MO(ma\_nd\_ID))}$[routing_cost(ma_nd_ID; y)]
5  neigh(ma_nd_ID) ← {v | v ∈ V | dist(ma_nd_ID; v)= 1}
6  FOR EACH v ∈ neigh(ma_nd_ID)
7     MO(v)← {x | x ∈ monitored(ma_nd_ID) &
8              next_hop(ma_nd_ID; x) = v }
9     cost(v)← $\sum_{(y|y\in MO(v))}$[routing_cost(ma_nd_ID, y)]
10 FOR EACH v ∈ neigh(ma_nd_ID)
11    estimate_MA(v)← heur(cost(ma_nd_ID); cost(v);
12                         $\sum_{(x|x\in neigh(ma\_nd\_ID))}$[cost(x)])
13 FOR EACH v ∈ neigh(ma_nd_ID)
14   IF (estimate_MA(v) = 1)
15      new_ma_ID ← clone_ma
16      new_ma_nd_ID ← v
17      monitored(new_ma_ID) ← MO(v)
18      monitored(ma_ID)←{monitored(ma_ID) \
        monitored(new_ma_ID) }
19      migrate_ma(ma_ID=new_ma_ID; ma_nd_ID=v)
20 IF (monitored(ma_ID) ≠ {})
21    start_ma(ma_ID)
22 ELSE
23    kill_ma(ma_ID)
```

Executed only once at initialization time
At most N lookups to local routing table
At most δ_{max} iter.
At most δ_{max} iterations
At most R(u) repetitions
Simple agent start up operation

Listing 6-2: Analysis of the distributed agent location algorithm presented in Chapter 4.

Let the network be modelled as a graph *G=(V,E)* consisting of a set of vertices (or nodes), *V* and edges, *E*. Each edge, *e*∈*E* has two different end points, v_1 and $v_2 \in V$. Given two generic nodes v_x and $v_y \in$ V, let us denote by $d(v_x)$ the degree of v_x in *G*, by $dist(v_x,v_y)$ the distance between v_x and v_y. Let $\delta_{\max} = \max_{v|v\in V}\{d(v)\}$; $R(v_x)$=$\max_{v|v\in V}\{dist(v_x,v)\}$ be the radius of the network centred in v_x; $neigh(v_x)=\{v \mid v\in V \,|\, dist(v_x,v)=1\}$ be the set of neighbours of v_x in *G*; and T_r is the routing tree rooted at the monitoring station.

Steps 1-23 are initially performed at the root node, $u\in T_r \subseteq V$ and, depending on the outcome of the test performed in Step 14, may or may not be repeated at the subsequent levels of T_r. In the worst case, Steps 1-23 are repeated *R(u)* times. In any case, agents running at the same level in T_r are executed independently from each other and in separate physical locations within the network. The computational complexity of the agent location algorithm considered as a whole can be determined by considering the part of the algorithm that is inherently sequential. Therefore, the complexity is *R(u)* times the complexity of Steps 4-23.

Upon arriving to a node an agent needs to be de-serialised and instantiated before executing from Step 4. This operation takes a constant time, $DESERIL_{time}$. Step 4-9 require a number of iterations equal, at most, to the total number of monitored nodes. The dominant cost for each iteration is given by the look-up operation to the routing table to extract the *next_hop* and the *cost* values. Thus, at each level of T_r, the total contribution of Steps 4-9 is *c**O(*N*), where *c* accounts for one look-up time. Steps 10-12 involve a number of iterations which, in the worst case is equal to the maximum node degree, δ_{max} that in typical networks is significantly smaller than the number of nodes and, typically, does not significantly increase with N. The iteration on v of Steps 13-19 is actually performed as part of Steps 10-12 and in the worst case involves the process of cloning and configuring δ_{max} new agents. Cloning will take a constant time, $CLON_{time}$; the reassignment of the monitored nodes takes a constant time too because it reuses information initially processed during Steps 4-9. Finally, each new agent will require a serialisation time, $SERIAL_{time}$ before being sent to its destination. The latter will add a forwarding delay, $FORW_{time}$ and a transmission time, $TRANSM_{time}$ (Step 19).

Therefore the agent deployment time, $DEPL_{time}$ that actually coincides with the time to compute the agent location algorithm, can be expressed as:

$DEPL_{time} = \{DESERIL_{time} + c^*\mathrm{O}(N) + \delta_{max}{}^*\ [CLON_{time} + SERIAL_{time}\ ] + TRANSM_{time} + FORW_{time}\}^*\mathrm{O}(R(u)) = \mathrm{c}_1\ {}^*\mathrm{O}(N * R(u)) + \mathrm{c}_2 * \mathrm{O}(R(u)) \propto \mathrm{O}(N^*R(u))$.

In practice, c_1 is of the order of at most 10E-6 seconds since the current router technology allows for a number of look-up operations of at least 10E6 per second. c_2 is in the order of seconds since with current mobile agent platforms [$TRANSM_{time} + FORW_{time}$] is typically in the order of 10E-3 to 10E-1 seconds and [$DESERIL_{time}+ CLON_{time}+ SERIAL_{time}$] is in the order of seconds or fraction of seconds [Knight 99, Bohoris 00c]. Therefore, if N << 10E6 then [$c^*\mathrm{O}(N)$] << $\{DESERIL_{time} + \delta_{max}{}^*[CLON_{time} + SERIAL_{time}] + TRANSM_{time} + FORW_{time}\}$ and, consequently, $DEPL_{time} \approx \mathrm{c}_2 * \mathrm{O}(R(u))$. In this case the deployment term will predominate over the computational one and $DEPL_{time}$ will be in the order of seconds times $\mathrm{O}(R(u))$.

The results of the above analysis prove that the algorithm is $\mathrm{O}(N^*R(u))$ in general while it is, in practice, $\mathrm{O}(R(u))$ if N << 10E6 because, under this condition, $\mathrm{c}_1\ {}^*\mathrm{O}(N * R(u)) << \mathrm{c}_2 * \mathrm{O}(R(u))$. Since $R(u)$ is typically sublinear with N, the proposed algorithm is sublinear with N. The algorithm complexity does not vary with the number of MAs, p since each agent computes its location in parallel and independently from the others. This is conditional to the assumption that the system where the agents are executed has sufficient memory and computational resources to meet the agents requirements.

6.2 Upper bounds on Agent Deployment Time

Agent deployment involves the computation of the agent locations as well as the actual agents' migration. If the centralised algorithm described in Chapter 4 is used, those two phases are performed sequentially – *i.e.*, agent locations are computed centrally prior to initiating agent migration to target destinations. Conversely, in the case of the distributed algorithm agent locations are computed during the actual agent migration. Therefore, in order to compare these two cases, the agent deployment time is assumed as the span of time elapsed from the initiation of the agent location algorithm to the moment in which the last agent has reached its target destination.

6.2.1 Upper Bounds on Deployment Time for the Centralised Algorithm

The time to compute the centralised algorithm is related to its algorithmic complexity. Thus, an upper bound the agent location computational time is $O(p+ N^2)$. An upper bound on the second contribution to the total agent deployment time – *i.e.* the agent migration time – can be accounted for as follows.

Let us assume that the network is modelled by a graph $G=(V, E)$ as specified above. In addition, we assume that at deployment time not more than one MA can simultaneously traverse the same link. This means that if two or more agents concur for the same link, they will be transmitted over the link sequentially. A similar assumption has already been adopted elsewhere and has been shown to reflect the hardware characteristics of communication networks. For instance, in [Rescigno 97] this model has been used to study various polling algorithms for communication networks. Responses are accumulated at internal nodes, but only one response can traverse a link during a communication step.

Another reason for modelling the agent distribution mechanism as above is that in this way it is generally possible to limit the dramatic burst of traffic incurring in the vicinity of the monitoring station by the injection of agents. Additionally, this is the approach used in the simulations reported in Chapter 8 in order to avoid buffer overflows at intermediate nodes and link saturation. It should be mentioned that, ideally, agent deployment should not monopolise the network resources available to the monitoring system. This is a more general guideline usually applied to management systems for which a rule of thumb is that the management system should operate in a way to use less than 5% of the available network resources. The remaining 95% should remain available for the normal operation of the managed system.

Let us now assume that the agent location algorithm has computed the locations for p agents, whereby $p<N$. Depending on network topology, location of the monitoring station, and routing distribution tree, there will be situations in which agents will concur for the same link or not. There are two extreme cases, as depicted in Figure 6-2: a) agents have mutually exclusive

distribution paths, thus they do not compete for links under any circumstance; b) agents share exactly the same distribution path.

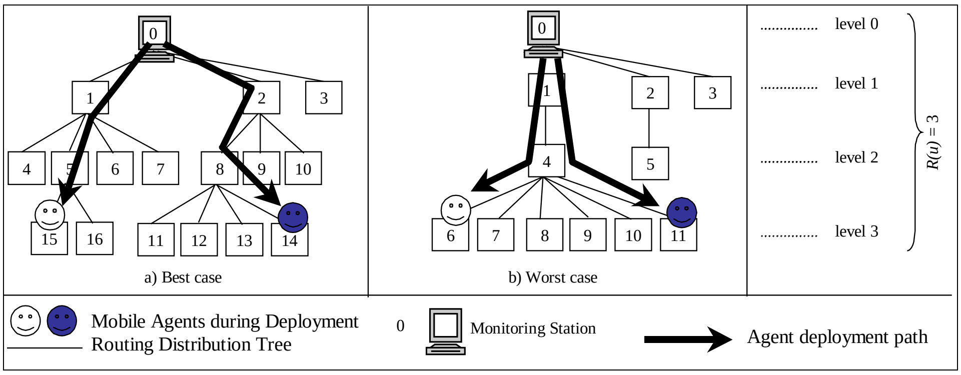


Figure 6-2: Examples of agent deployment. a) the agents traverse different portions of the distribution tree; b) the agents have overlapping distribution paths.

In the first case agent distribution is realised with the maximum possible parallelism and, thus, in minimal time. The upper bound on distribution time is given by the agent which has to travel further. In the worse case is achieved if we assume that at least one agent has target destination at distance *R(u)* from the monitoring station. In this case the distribution time can be expressed as:

$$DIST_{time,cent,a} = (TRANSM_{time} + FORW_{time}) * R(u)$$

whereby $TRANSM_{time}$ is the time taken by an agent to traverse a single link and $FORW_{time}$ is the forwarding delay incurred at intermediate nodes. The former is typically in the order of tens of milliseconds. The latter depends mainly on the queuing delay at the node and ideally falls in the same order of magnitude as the former.

If agents share the same distribution path (Figure 6-2b) the time to deploy *p* agents from level 0 – *i.e.*, the monitoring station level – to level *R(u)* is given at most by the time to bring *p* agents to level 1 plus the time required by 1 agent to traverse all the other levels. Thus, an upper bound on the total agent distribution time can be expressed as:

$$DIST_{time,cent,b} = (TRANSM_{time} + FORW_{time}) * [p + (R(u) - 1)]$$

The following expression holds:

$$DIST_{time,cent,a} \leq DIST_{t,cent,b}$$

therefore $DIST_{t,cent,b}$ is used as upper bound on distribution time. Its asymptotic complexity will be O(p + $R(u)$). Therefore, the upper bound on the total deployment time given by the sum of the contribution accounting for the computation of agent locations (Section 6.1.1) and the distribution time can be expressed as:

$$DEPL_{time,cent} \propto c_1 *[O(p+ N^2)] + c_2 *[O(p + R(u))]$$

whereby c_1 and c_2 represent the order of magnitude associated with two terms, respectively. c_2 accounts for transmission and queuing delay and is typically in the order of tens of milliseconds. c_1 may vary greatly depending on the computational resources available at the monitoring station and will typically predominate over c_2. It should be noticed, however, that by adding sufficient processing power this conditions might not hold. Under the assumption of this thesis the predominant term of the overall deployment process tends to be the one accounting for agent location computation, leading to the following expression:

$DEPL_{time,cent} \propto c_1 *[O(p+ N^2)]$

6.2.2 Upper Bounds on Deployment Time for the Distributed Algorithm

Let us recall from Section 6.1.2 that in the case of the distributed agent location algorithm the agent deployment time, $DEPL_{time}$ actually coincides with the time to compute the agent location algorithm, which can be expressed as:

$DEPL_{time,dist} = \{DESERIL_{time} + c*O(N) + \delta_{max}* [CLON_{time} + SERIAL_{time}\ ] + TRANSM_{time} + FORW_{time}\}*R(u) \propto O(N*R(u))$.

Finally, in practice the agent migration contribution tends to predominate (as seen in 6.1.2) leading to the following expression:

$DEPL_{time,dist} \propto O(R(u))$

which is sub-linear with N because $R(u)$ is sub-linear with N for typical network topologies.

6.3 Upper Bounds on Agent Deployment Traffic

The agent deployment traffic is the one incurred in the network by the agent deployment process. The amount of incurred traffic may change significantly between the centralised and the distributed algorithm. In the former case, all the agents are created at the monitoring station where the deployment process starts. In the latter case, only one agent is initially created at the monitoring station. This agent starts the deployment process by cloning one or more agents that are, then, migrated to the second level of the distribution tree. Each of the new agents will then repeat this process until agent deployment is finalised. This approach based on agent cloning is responsible for the optimisation of the deployment process.

Traffic is modelled using an approach similar to [Zegura 97]. It is the sum of packet hops – *i.e.* the number of links traversed by the packets – multiplied by their respective packet size, *b* and packet rate P_r. The traffic $T(v_1,v_2)$ between any two points v_1, $v_2 \in V$ subject to a bit rate B_r is $T(v_1, v_2)= B_r * d(v_1, v_2)$, where $B_r=b*P_r$ and $d(v_1,v_2)$ is the distance between v_1 and v_2. The *distance* in the network is measured using the 'hop' metric, in which each edge has unit weight.

6.3.1 Upper Bounds on Deployment Traffic for the Centralised Algorithm

The upper bound on traffic is calculated by considering the worst case, that is the case in which all of the *p* agents are to be deployed to nodes which are at distance *R(u)* from the monitoring station. Let as assume that all agents have byte size $MA_{size,cent}$. If we assume that the whole deployment process takes a single unit of time, the average traffic incurred by agent deployment, expressed in bits per second, will be:

$$DEPL_{traff\,,cent} = \frac{8 * p * MA_{size,cent} * R(u)}{1\sec} \propto O\big(p * R(u)\big)$$

6.3.2 Upper Bounds on Deployment Traffic for the Distributed Algorithm

In the case of the distributed algorithm the calculation of an upper bound on agent deployment traffic is not trivial since this may vary significantly depending on the agent cloning process. This depends, in turn, on particular topological properties such as the average node degree. Similarly to the approach followed in the previous section, the upper bound is calculated by considering the case in which all the agents are assumed to be targeting level-*R(u)* nodes. However, for the distributed algorithm two extreme upper bounds are computed, an 'optimistic' and a 'pessimistic' one.

The 'optimistic' upper bound on traffic is achieved when no cloning is involved before the last level of distribution tree is reached (Figure 6-3a). This happens when the first agent, created at the monitoring station, traverses levels 1 to (*R(u)*-1) without cloning any new agents and, then, creates the required p agents upon arriving at level (*R(u)*-1). Conversely, the 'pessimistic' case is the one in which all agents are cloned by the first agent at the monitoring node (Figure 6-3b).

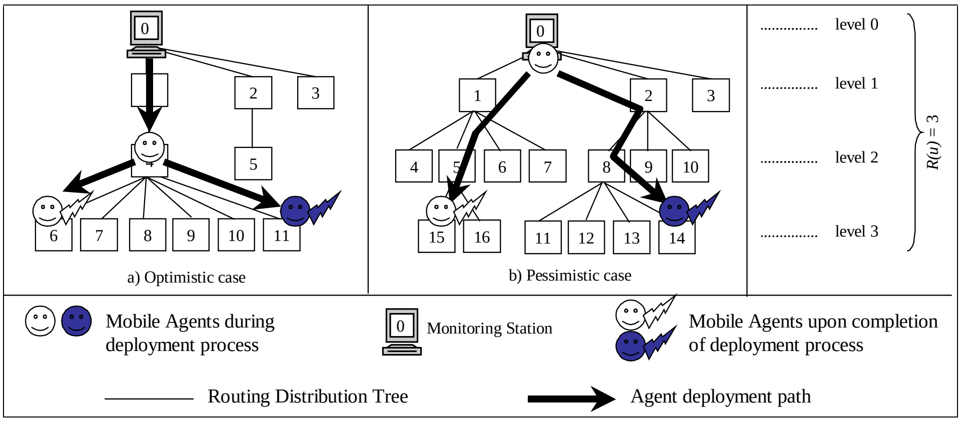


Figure 6-3: Example showing the optimistic and pessimistic upper bounds on deployment traffic in the distributed algorithm.

Let as assume that all agents have byte size $MA_{size,dist}$, and that as usual the number of deployed agents is p and the network radius is $R(u)$. The optimistic upper bound on traffic is given by the traffic incurred when the first agents traverses ($R(u)$-1) links plus the traffic incurred to transmit (p-1) agents from level ($R(u)$-1) to level $R(u)$. Hence the traffic expressed in bits per second will be:

$$DEPL_{traff,dist,opt} = \frac{8 * MA_{size,dist} * \left(R(u) - 1 + p\right)}{1\sec} \propto O\left(p + R(u)\right)$$

The pessimistic upper bound on deployment traffic is analogous to the one of the centralised algorithm and can be expressed as:

$$DEPL_{traff,dist,pess} = \frac{8 * p * MA_{size,dist} * R(u)}{1\sec} \propto O\left(p * R(u)\right)$$

It can be noted that the optimistic and the pessimistic upper bounds are characterised by significantly different scalability. An intermediate behaviour should be expected in practical systems whereby the cloning process is performed progressively at the various levels of the distribution tree.

It should be mentioned that in the case of the distributed algorithm, agents tend to have a larger size than the size of the agents generated by the centralised algorithm. This is because, in the first case, agents need to contain the logic to solve the location problem and perform agent cloning in addition to the logic implementing the actual monitoring task.

Therefore, whenever behaviour similar to the pessimistic case is achieved the centralised algorithm tends to result in smaller traffic. Conversely, the distributed algorithm will result in reduced traffic if the cloning/deployment process can be assimilated to the optimistic case.

6.4 Discussion and Conclusions

This chapter presents the transient analysis of the proposed centralised and distributed algorithms, which find an approximate solution to the agent location (*p-median*) problem. From the point of view of an agent-based distributed monitoring system, this is also the problem of placing *p* agents in the network in such a way to optimise monitoring traffic and response time. The problem of finding an efficient solution to the *p-median* problem is crucial to the realisation of agent-based distributed monitoring since this is, in general, an NP-complete problem.

The main results of this analysis are summarised in Table 6-2. Both the centralised and the distributed algorithm are proved of polynomial complexity. The former is proved $O(p + N^2)$, whereas the latter is typically sub-linear with the number of monitored nodes and independent from the number of agents.

This is an important result if compared with the algorithms proposed in the literature, which are typically of higher polynomial degree. We have discussed in Chapters 2 and 3 how existing algorithms are not only more complex than the proposed one but also unfeasible to solve the agent location problem. A key feature of the distributed version of the proposed algorithm is its ability to be computed on local routing tables rather than on a ‘centrally maintained’ network distance matrix.

	Centralised Agent Location Algorithm	**Distributed Agent Location Algorithm**
Computational Complexity	$O(p + N^2)$	$O(N * R(u)) \approx O(R(u))$
Upper Bounds on Agent Deployment Time	$c_1 *[O(p + N^2)] + c_2 *[O(p + R(u))] \approx c_1 *[O(p+ N^2)]$	$c_1 *O(N * R(u)) + c_2 * O(R(u))$
Typical order of Deployment Time	Depends critically on the amount of computational resources available at the monitoring station	$c_1 \approx$ 10E-6 $* c_2$. If $N <<$ 10E6 → $DEPL_{time} \approx$ seconds * $O(R(u))$ otherwise $DEPL_{time} \approx$ 10E-6*seconds* $O(N*R(u))$
Upper Bounds on Agent Deployment Traffic	$O(p * R(u))$	$O(p + R(u))$ (optimistic) $O(p * R(u))$ (pessimistic)

Table 6-2: Summary of results on the theoretical evaluation of transient behaviour under general conditions.

The other important aspects assessed herein are deployment time and traffic, which represent overheads from the viewpoint of the monitoring system. Upper bounds on these variables have been calculated analytically. In particular, the distributed algorithm based on agent cloning typically results in times in the order *R(u)* seconds. Times of the order of the second are involved in the agent cloning/forwarding process over a single hop. This process is predominant with respect to the other phenomena and it is repeated at most a number of times equal to the network radius, *R(u)*. This behaviour is, hence, linear with the network radius and, in turn, sub-linear with the number of monitored nodes, *N* (since *N* is typically sub-linear with *R(u)*).

This result gives an important hint on the boundaries of applicability of agent-based monitoring. Intuitively, monitoring tasks characterised by durations comparable with, or smaller than, agent deployment times will tend not to benefit from this approach. Conversely, relatively long monitoring tasks will benefit significantly from the proposed approach since the performance advantages deriving from the agent system will pay-off the relatively small delays associated with agent deployment. It should be observed that the aim of the agent system is to minimise steady-state traffic and response time, as assessed in Chapter 8.

Another overhead is agent deployment traffic. With the centralised agent algorithm, traffic is shown to increase as O(*p***R(u)*). The order of magnitude of deployment traffic depends on the size of the agents. Simple tasks can be implemented with agent of the order of kilobytes [Knight 99, Bohoris 00c]. In turn, traffic will be in the order of kilobytes, growing as (*p***R(u)*).

With the distributed agent algorithm, traffic exhibits a worse-case behaviour analogous to the one achieved with the centralised agent algorithm. More optimistically, however, the upper bound on deployment traffic will behave significantly better (O(*p*+*R(u)*)). This significant reduction in traffic can be directly associated to the use of agent cloning.

Clearly, conventional centralised monitoring will not incur any additional overhead but pays this advantage in terms of steady-state performance. Static distributed approaches assume pre-defined, hard-coded monitoring logic; consequently, no deployment overheads are incurred either. In this case, the costs of the agent solution are paid-off with the increased flexibility offered by dynamic agent location. The ability of deciding on the area manager location at run time rather than through off-line computation is particularly important in the case of large-scale, dynamic networked systems.

What is missing to the analysis presented in this chapter is the assessment of the goodness of the agent locations computed by the proposed algorithms. This is the subject of Chapter 8. In the next chapter, we continue our mathematical analysis of the proposed agent system, to find sufficient conditions for agent location near-optimality. Constraining conditions on the network topology are found. However, the simulation work of Chapter 8 will reveal that location near-optimality is achieved by the agent system also for general, Internet-like network topologies.

Chapter 7

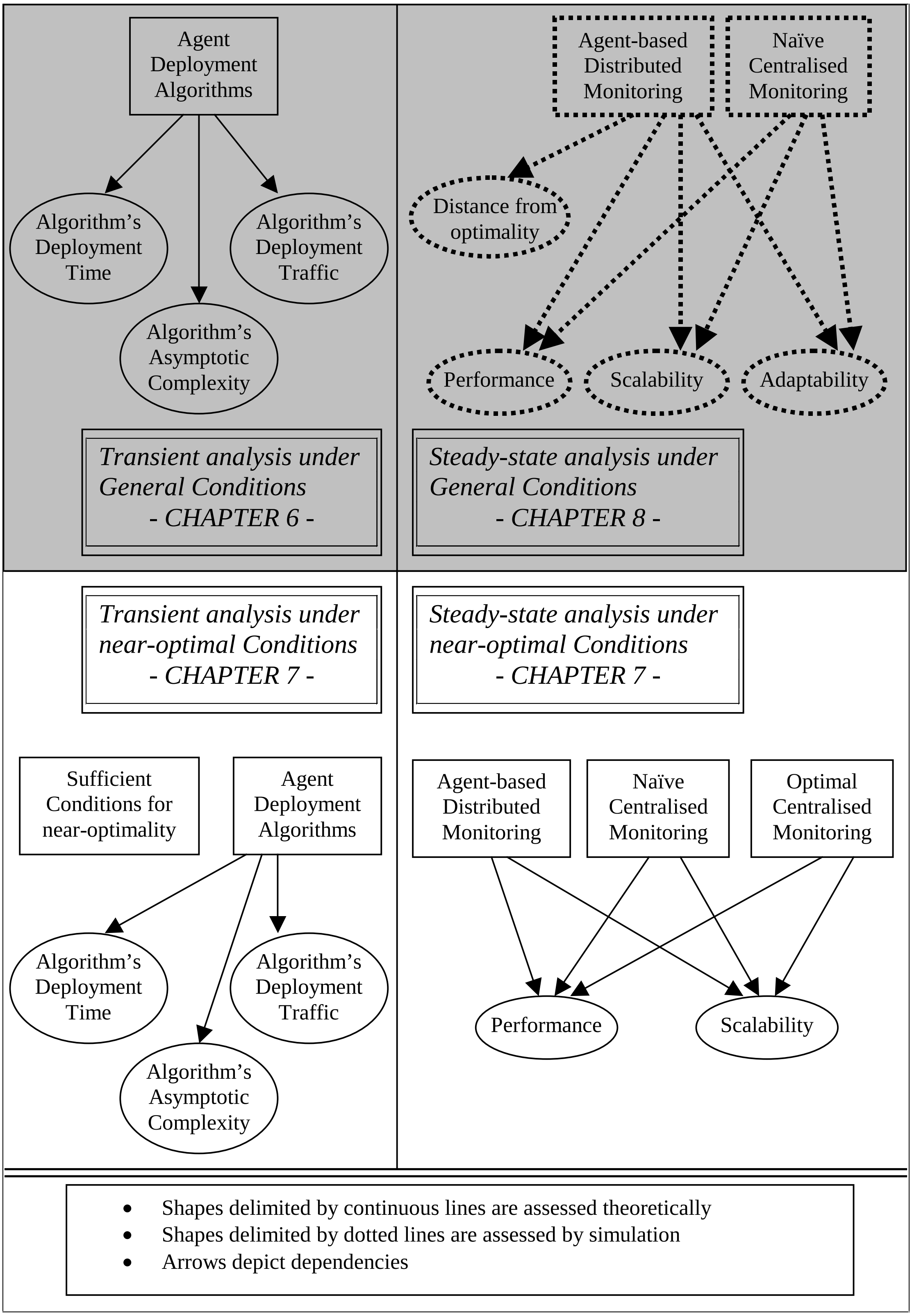
Theoretical Evaluation Under Near-Optimal Conditions

The previous chapter has covered the transient analysis of the proposed agent-based monitoring approach under general conditions. A mathematical-based analysis of the system at steady-state posed numerous difficulties. Steady-state assessment was, hence, performed through simulation, as reported in Chapter 8.

Herein we open a parenthesis to look more closely at the various aspects of agent-based distributed monitoring under near-optimal conditions. Near-optimality is intended in the sense of overall traffic minimisation.

A schematic representation of the focus of this chapter is depicted in Figure 7-1. We first give conditions for which the agent location algorithm finds near-optimal locations for the agents (Section 7.1). Such conditions are given on the network spanning tree routed at the monitoring station. We prove that if the network admits a spanning tree having *n*-ary balanced sub-trees and the number of agent is $p=d(u)*n^{L-1}$, whereby L is an integer smaller than the network radius, all agents will be placed at distance L from the monitoring station. Those agents will be located near-optimally in the network.

In the second part of the chapter, we present a theoretical assessment of agent-based monitoring at steady-state, under the given near-optimal conditions. Mathematical models of naïve centralised polling (Section 7.2), optimal centralised polling (Section 7.3), and agent-based distributed polling (Section 7.4) are presented. This is followed by a comparative analysis of performance and scalability among the various approaches (Section 7.5).


Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Transient analysis under General Conditions - CHAPTER 6 -
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Distance from optimality
Performance
Scalability
Adaptability
Steady-state analysis under General Conditions - CHAPTER 8 -
Transient analysis under near-optimal Conditions - CHAPTER 7 -
Steady-state analysis under near-optimal Conditions - CHAPTER 7 -
Sufficient Conditions for near-optimality
Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Optimal Centralised Monitoring
Performance
Scalability

- Shapes delimited by continuous lines are assessed theoretically
- Shapes delimited by dotted lines are assessed by simulation
- Arrows depict dependencies

Figure 7-1: Schematic representation of the focus of Chapter 7.

In the final part, we present a theoretical assessment of agent-based monitoring at transient time, under the given near-optimal conditions. We consider the two flavours of the agent deployment algorithm of Chapter 4. Models accounting for agent deployment traffic and time are presented in Section 7.6. In contrast with the transient analysis of Chapter 6, we manage to find mathematical models describing the transient behaviour of the agent system, rather than just upper bounds on agent deployment overheads. Performance and scalability are finally analysed in Section 7.7.

An initial development of the mathematical models reported herein has been published in [Liotta 01a].

7.1 Sufficient Conditions for Location Optimality

This section elaborates on the optimality of the locations computed with the proposed algorithms. Those algorithms represent, in general, an approximate solution to the *p-median* problem. The evaluation of the distance from optimality for general network topologies is carried out by simulation in the next chapter. Herein, sufficient conditions for agent location optimality are given.

DEFINITIONS:

u is the monitoring station node – *i.e.*, the root node for the spanning tree. *d(u)* is the node degree of node *u* – *i.e.* the number of sub-trees of the spanning tree. The spanning tree includes the routes which connect the root node, *u* with every other monitored node in the network. It is built on the routing tables which are computed by the network routing algorithm. *p* is the number of disjoint partitions of the network. Each partition is served by a different agent. Thus, *p* is also the number of agents. $\aleph$ is the set of positive integers excluding zero. *R(u)* is the radius of the network centred in *u* – *i.e.*, the maximum distance from *u* to any other node in the network. If a hop-count metric is adopted for the distance, *R(u)* is also the maximum number of levels of the spanning tree. Finally, an *n*-ary tree is a tree in which every non-leaf node has exactly *n* descendent nodes, that is every non-leaf node has node degree equal to (*n*+1). In a balanced tree every branch has the same depth, that is every leaf node is at distance *R(u)* from the root node *u* (we are assuming a ‘hop-count’ distance metric). Figure 7-2 depicts the assumed *n*-ary balanced spanning tree, whereby T_1, T_2, …, $T_{d(u)}$, are *n*-ary balanced sub-trees of depth (*R(u)*-1).

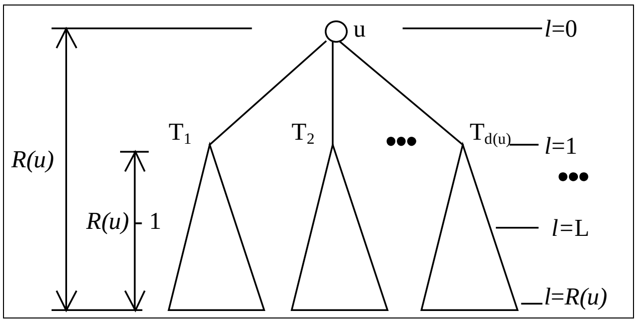


Figure 7-2: An example spanning tree having *n*-ary balanced sub-trees.

THEOREM 1:

Sufficient conditions for the proposed agent location algorithms to find a near-optimal solution to the *p-median* problem are:

1. the spanning tree rooted at *u* and built on the network routing tables has *d(u) n*-ary, balanced sub-trees;
2. $p = d(u) * n^{(L-1)}$ with $L \in \aleph$ and $1 < L \leq R(u)$

PROOF:

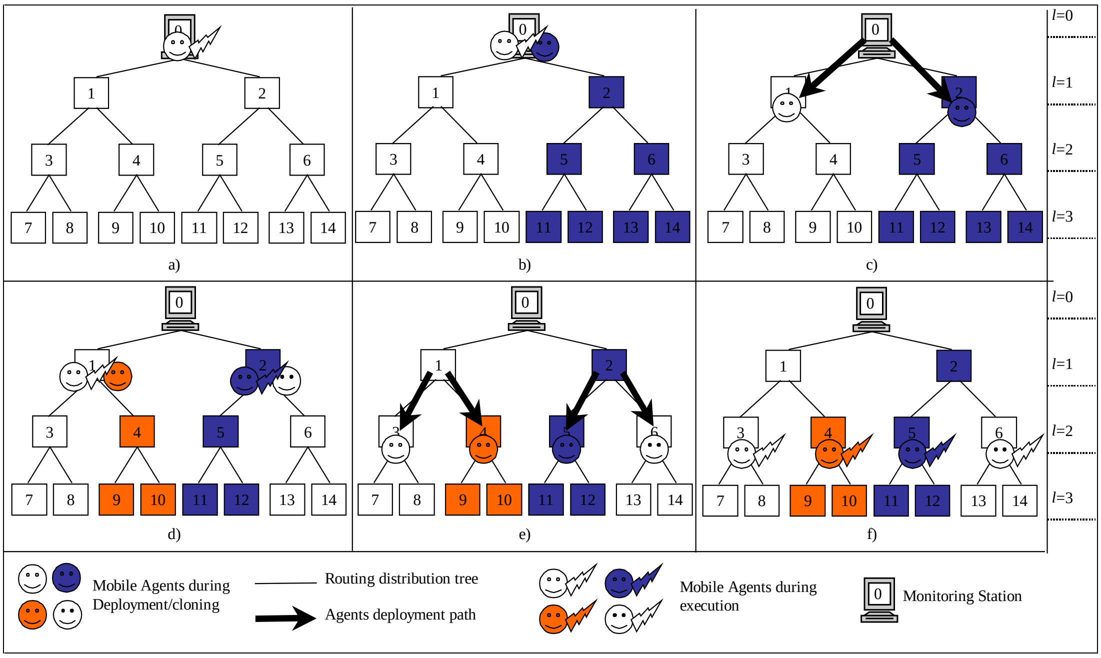
We need to prove that the *p* locations computed by the location algorithm are near-optimal, *i.e.* they result in near-minimal incurred monitoring traffic. That, in turn, means that the sum of the distances between each agent and the corresponding nodes belonging to the agent partition is near-minimal. In the following, *distances* in the network are measured using the ‘hop-count’ metric in which each edge has unit weight. Other than the hop-count metric may be used, though the analysis presented herein would need to be adapted to those case.

Links are modelled as in [Rescigno 97]: all links are assumed to be equal in capacity and latency; links are modelled as half-duplex channels that can be traversed by one packet at the time (packets have mutually exclusive access to links).

The theorem is proved in three parts. First, the agent locations computed by the algorithm under the given conditions are obtained. Then, for each network partition the sum of the distances between the agent location and the nodes in the partition is calculated. Finally, it is shown that the total sum of distances is near-minimal.

PROOF – PART I: AGENT LOCATIONS

Both the centralised and the distributed algorithm described in Chapter 4 will lead to the same locations. Let us describe what happens in the case of the distributed algorithm (pages 84-90) by referring to the example of Figure 7-3.



l=0
l=1
l=2
l=3
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
a)
b)
c)
d)
e)
f)
Mobile Agents during Deployment/cloning
Routing distribution tree
Agents deployment path
Mobile Agents during execution
Monitoring Station

Figure 7-3: Example of agent location computation with the distributed algorithm, for a network having a binary spanning tree.

Initially, only one agent is created at the monitoring station (level *l*=0). Hence, there is only one network partition including all the monitored nodes (Figure 7-3a). This agent divides the network into two partitions, clones a second agent, and assigns it to the second partition (Figure 7-3b). The agents migrate to level *l*=1 to gain a more central location within the partition (Figure 7-3c). Each of them further partitions the network and clones another agent (Figure 7-3d). The four agents estimate that a more convenient location is at level *l*=2 and migrate (Figure 7-3e). Finally, no need for further partitioning/cloning is found and each agent can start monitoring the nodes in its partition. Therefore, all four agents will be placed at level *l*=2 of the spanning tree; and it can be concluded that if $p = d(u) * n^{(L-1)}$ all agents will be located at level *L*. This result is valid more generally under the assumptions of the above theorem. To prove that let us look more closely at the partitioning process. Since the spanning tree is assumed *n*-ary and balanced, a generic agent at a generic level *l* will always see its descendent nodes as a balanced *n*-ary sub-tree. The agents will be in the root of this sub-tree which is also the centre. Hence, whenever an agent is facing the problem of partitioning the sub-tree it will have two options: 1) partition the sub-tree equally and clone (*n*-1) other agents, as exemplified in Figure 7-3; or 2) stop the partitioning/cloning process if no more agents can be cloned.

Agent may adopt different heuristics to drive the cloning process. A simple possibility is for an agent to stop cloning other agents when the number of its monitored nodes becomes smaller than a given threshold. Let us assume that this threshold is reached when $p = d(u) * n^{(L-1)}$. This means that, for the case of *n*-ary balanced spanning tree, agents are located at level *L*, since the maximum number of agents $p = d(u) * n^{(L-1)}$ has already been created when the agents reach level *L*. Therefore, the nodes belonging to level *L* of the spanning tree will be the ones computed by the algorithm as the set of agent locations.

PROOF – PART II: SUM OF DISTANCES

Let us now prove that the set of those locations represents the *p-median* of the network. To do that let us calculate the sum of the distances between an agent and all the nodes in its partition. The computed locations are a *p-median* for the network if the sum of these distances, extended to all the agents, is minimal.

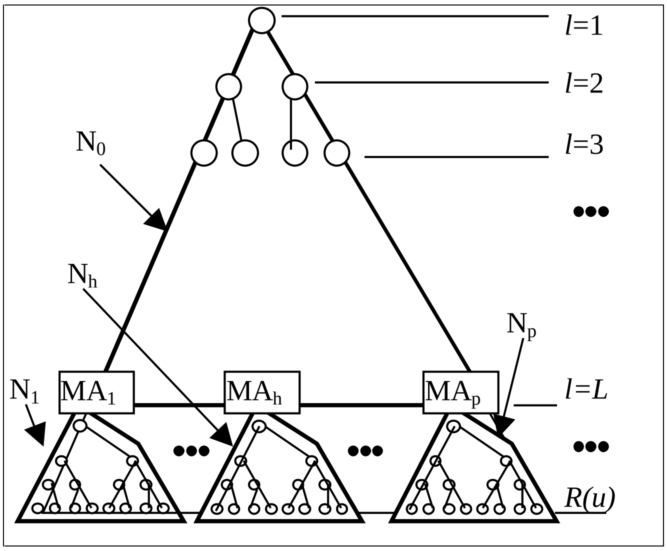


Figure 7-4: Example binary spanning sub-tree depicting the calculation of the total distances between an agent and the nodes in its partition.

First, we need to define a naming scheme for nodes and partitions of the spanning tree. Then, we will specify formally the nodes monitored by each of the agents. Finally, we will compute the distances. Figure 7-4 depicts the generic *n*-ary sub-tree T_r of the spanning tree of Figure 7-2.

A generic node of T_r can be identified as $n_{l,h}$ whereby $l \in (1, 2, ..., R(u))$ identifies particular tree levels and $h \in (1, 2, ..., n^{(l-1)})$ identifies the various nodes at a given level. The set of nodes where the agents are placed is $\{n_{l,h} \in T_r \mid l = \mathrm{L}\ \&\ h \in (1, 2, \ldots, n^{(L-1)})\}$. The sub-tree is partitioned in (h+1) partitions $\{N_0, N_1, \ldots, N_h\}$, whereby

$N_0 \equiv \{\, n_{l,h} \in T_r \mid l \in (1, 2, \ldots, (L-1))\ \&\ h \in (1, 2, \ldots, n^{(l-1)})\}$ and

$N_x \equiv \{\, n_{l,h} \in T_r \mid l \in (L, (L+1), \ldots, R(u))\ \&\ h \in [(x-1)*n^{(l-L)} + 1, \ldots, x*n^{(l-L)})\}$, $x \in \{1, 2, \ldots, p\}$.

The partition associated with the generic agent MA$_x$ will be the union of N_x with some of the nodes belonging to partition N_0. In fact, the nodes of N_0 will be evenly distributed among the agents. Since the number of agents residing at level *L* will be larger than the total number of nodes in N_0, each agent will be in charge of monitoring at most 1 node from N_0. Thus, for each agent the sum of the distances between agent and nodes is given by two terms: the sum of the distances between MA$_x$ and the nodes of N_x plus the contribution due to those nodes of N_0 which belong to the agent. The former can be expressed as:

$$S_{d,N_x} = \sum_{j=1}^{R(u)-L} j*n^j = \sum_{j=o}^{R(u)-L} j*n^j = \frac{(R(u)-L)*n^{(R(u)-L+2)} - (R(u)-L+1)*n^{(R(u)-L+1)} + n}{(n-1)^2}$$

whereby it is assumed (here and in the following) that $n \neq 1$ for the geometric series to converge. n cannot assume zero or negative values. Hence, it is assumed that $n>1$.

The contribution to the sum of distances due to the nodes of N_0 can be calculated by multiplying the portion of those nodes of N_0 belonging to a single agent by the average distance between MA$_x$ and the generic node of N_0. The total number of nodes in N_0 is:

$$|N_0| = \sum_{j=0}^{L-2} n^j = \frac{n^{L-1}-1}{n-1}$$

The total number of agents at level L is $p=n^{L-1}$. The average number of nodes from N_0 belonging to a single agent is $\frac{|N_0|}{p} = \frac{n^{L-1}-1}{(n-1)*n^{L-1}} < \frac{1}{n-1}$. Assuming a hop-count metric, we find that the average distance between an agent and the portion of nodes belonging to N_0 is

$$\frac{\sum_{j=1}^{L-1} j}{L-1} = \frac{(L-1)*L}{2*(L-1)} = \frac{L}{2}$$

We should in fact consider that, for any given agent sitting at level L, there is only a single node in N_0 at each level {1, 2, …(L-1)} which is eligible for being monitored by that agent. That node will be one of the direct ascendants of the agent node.

Therefore, the contribution to the total distance due to the nodes of N_0 will be on average:

$$S_{d,N_0} = \frac{|N_0|}{p} * \frac{L}{2} = \frac{n^{L-1}-1}{(n-1)*n^{L-1}} * \frac{L}{2} < \frac{L}{2*(n-1)}$$

Thus, the total sum of distances for a single agent will be $S_{dx} = S_{d,N_0} + S_{d,N_x}$.

PROOF – PART III: NEAR-OPTIMALITY

Having calculated the two different contributions of the total sum of distances for each agent, we can see that if $L \ll R(u)$ the contribution due to the nodes belonging to N_0 tends to be negligible. If we can neglect S_{d,N_0} we can say that MA$_x$ is located in the centre of N_x and, in good approximation, in the centre of the whole agent partition including N_x and the nodes of N_0 belonging to MA$_x$.

Since the root of a balanced tree is also the 1-median of the tree [Reid 91] and having demonstrated that each agent is located in the root of a balanced tree, we can conclude that each agent MA$_x$ is located in the median of N_x. This means that, if we could neglect S_{d,N_0}, each agent would be located optimally within its partition, *i.e.* the monitoring traffic incurred by agents would be minimal. As already said, S_{d,N_0} tends to be negligible but not null. Therefore, agents are located near-optimally and will incur near-minimal monitoring traffic. In conclusion, the set of nodes at level *L* will represent a near-optimal solution to the *p-median* problem for the given network.

COROLLARY:

Let us assume that a distributed monitoring system based on the proposed location algorithm is realised. Sufficient conditions for the system to incur near-minimal monitoring traffic are:

1. the spanning tree rooted at *u* and built on the network routing tables has *d(u) n*-ary, balanced sub-trees;
2. $p = d(u) * n^{(L-1)}$ with $L \in \aleph$ and $L \leq R(u)$

PROOF:

We recall that, within the context of this thesis, the traffic $T(n_{l1,h1}, n_{l2,h2})$ between any two nodes $n_{l1,h1}, n_{l2,h2} \in V$ is defined as $T(n_{l1,h1}, n_{l2,h2}) = b*P_r* d(n_{l1,h1}, n_{l2,h2})$, whereby *b* is the packet size, P_r is the packet rate, and $d(n_{l1,h1}, n_{l2,h2})$ is the distance between the nodes.

The total traffic incurred by the *p* agents in a sub-tree of the spanning tree will be sum of the traffic incurred by each individual agent. The traffic incurred by the generic agent MA$_x$ can be expressed as:

$$T_x = T_{N_x} + T_{N_0} = \sum_{j=1}^{R(u)-L} b * P_r * j * n^j + b * P_r * \frac{|N_0|}{p} * \frac{L}{2} = b * P_r * \left(S_{d,N_x} + S_{d,N_0}\right)$$

whereby P_r is the polling rate expressed in polls per unit of time.

Therefore, by direct application of the above theorem it can be concluded that the traffic is near-minimal, since the sum of distances is near-minimal.

7.2 Steady-state Models of Naïve Centralised Polling under Near-Optimal Conditions

A common approach to centralised monitoring is the *centralised polling* technique. In this case, we have a centralised polling station that monitors a set of nodes (the monitored nodes) according to a two-step process. It first issues requests or *polls* to the monitored nodes; then each node sends a unique response back to the station. Polls are usually issued on a periodic basis. Therefore, the station can rely on periodic responses from the monitored system. Response packets are processed, for instance, to detect problems or build performance statistics.

In a Local Area Network (LAN), centralised polling may be implemented by broadcasting the request packets and collecting the responses at the station. When the monitored system crosses the boundaries of a LAN the broadcast phase is substituted by two alternative mechanisms. One possibility is to adopt a point-to-point communication model whereby the station sends individual *polls* to each monitored node (multiple unicasting). This is usually referred to as *Naïve Centralised Polling*. A more elaborate possibility is to follow the multicasting model to issue the *polls*. This approach is termed *Optimal Centralised Polling* because, among all possible centralised polling-based monitoring solutions, this is the one that results in minimal incurred traffic.

In this section, polling traffic and polling response time incurred under the near-optimal conditions given above are calculated. In this case, only traffic and response time at steady-state need to be calculated, since no agent deployment is involved. Models for the case of optimal centralised polling are in Section 7.3.

The naïve polling process is exemplified by the example in Figure 7-5. In the context of this thesis, monitored nodes are assumed to be interconnected through an internetwork such as the Internet. Nodes are interconnected by point-to-point links. For simplicity, and without loss of generality, we assume that all networked nodes are being monitored by the polling station. In this case, the 'request' phase consists of a *point-to-point request* mechanism analogous to the one depicted in Figure 7-5a.

During the request phase, a number of packets equal to the total number of nodes need to be transmitted from level 0 to level 1. Similarly, a number of packets equal to the number of nodes belonging to levels 2 and 3 needs to travel from level 1 to level 2; and a number of packets equal to the number of nodes belonging to level 3 needs to travel from level 2 to level 3. During the response phase, the reverse process is performed (Figure 7-5b).

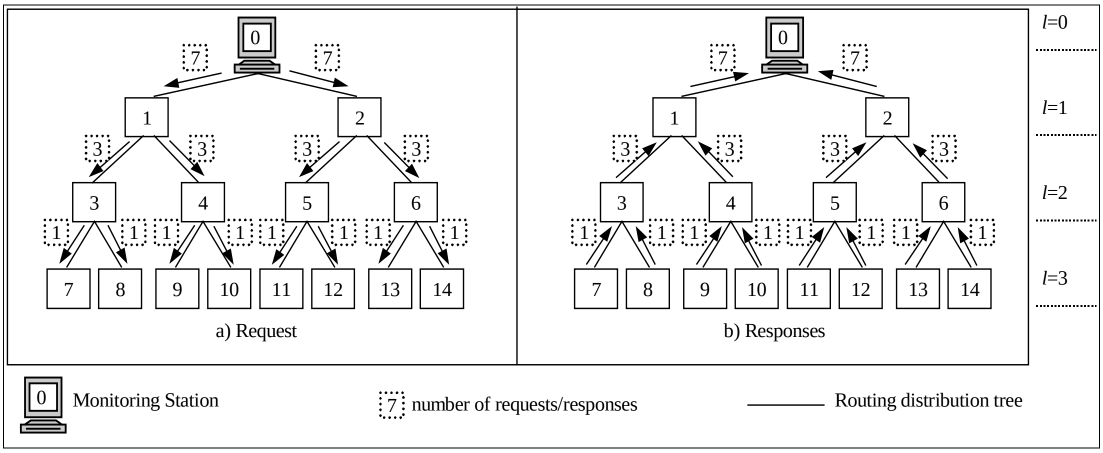


Figure 7-5: Example of centralised naïve polling under near-optimal conditions.

7.2.1 Steady Traffic in Naïve Centralised Polling

To calculate the incurred monitoring traffic let us consider first the request traffic. The number of packets transmitted from level *l* to level (*l*+1) can be expressed as $d(u)*\sum_{i=l}^{R(u)-1} n^i$. The bit rate between those levels will be $P_r * 8 * b_b * d(u) * \sum_{i=l}^{R(u)-1} n^i$, whereby P_r is the packet rate and b_b the size in bytes of a single request packet. Therefore, the total traffic incurred during the requesting phase is $\sum_{l=0}^{R(u)-1}\left(P_r * 8 * b_b * d(u) * \sum_{i=l}^{R(u)-1} n^i\right)$. The traffic due to the response packets can be calculated similarly and will be $\sum_{l=0}^{R(u)-1}\left(P_r * 8 * b_r * d(u) * \sum_{i=l}^{R(u)-1} n^i\right)$ whereby b_r is the size in bytes of a single response packet. If for the sake of simplicity response packets are assumed all of equal size the total monitoring traffic will be:

$$MON_{traff,ncp} = \sum_{l=0}^{R(u)-1}\left(P_r * 8 * b_b * d(u) * \sum_{i=l}^{R(u)-1} n^i\right) + \sum_{l=0}^{R(u)-1}\left(P_r * 8 * b_r * d(u) * \sum_{i=l}^{R(u)-1} n^i\right)$$

that is

$$MON_{traff,ncp} = P_r * d(u) * 8 * (b_b + b_r) * \sum_{l=0}^{(R(u)-1)} \sum_{i=l}^{(R(u)-1)} n^i$$

If we develop the double series as indicated in Appendix we obtain the following expression:

$$\boxed{MON_{traff,ncp} = P_r * d(u) * 8 * (b_b + b_r) * \left\{ \frac{R(u) * n^{R(u)} * (n-1) - n^{R(u)} + 1}{(n-1)^2} \right\}}$$

To assess how traffic increases with scale we can study the above expression for

[P_r, *R(u)*, *N*] → ∞, achieving:

$$MON_{traff,ncp} \propto O\left(P_r * R(u) * n^{R(u)}\right)$$

For the particular network topology assumed here, we have that $n^{R(u)} \propto N$ whereby N is the number of monitored nodes. Hence:

$$\boxed{MON_{traff,ncp} \propto O\left(P_r * R(u) * N\right)}$$

7.2.2 Steady Response Time in Naïve Centralised Polling

The response time is the span of time elapsed between the issuing of the first request packet by the monitoring station and the arrival of the last response packet at the station. The total 'point-to-point request' time is the span of time elapsed between the issuing of the first request packet by the monitoring station and the arrival of the last request packet at the nodes.

Request time may vary significantly depending on the order in which request packets are issued by the station. The upper bound on request time is represented by the case in which request packets are sent first to those nodes that are nearer to the station and then incrementally to the further ones. We assume a network model in which not more than one packet can simultaneously traverse a link and a packet takes a unit of time to traverse a link.

For the example of Figure 7-5a the upper bound on request time will be the time to transmit 7 packets from level 0 to level 1 (7 units of time), plus the time to transmit 3 packets between levels 1-2 (3 units of time), plus the time to transmit 1 packet between levels 2-3 (1 unit of time). The upper bound on request time would then be equal to 11 units of time.

More generally the upper bound on request time, for $0 \le P_r < 1$ (P_r being expressed in polls per second), can be expressed as:

$$REQ_{time,ncp,ub} = \sum_{i=0}^{R(u)-1} n^i + \sum_{i=0}^{R(u)-2} n^i + ... + \sum_{i=0}^{R(u)-R(u)} n^i = \sum_{j=1}^{R(u)} \sum_{i=0}^{(R(u)-j)} n^i$$

by developing first the inner geometric series and assuming $n \neq 1$ we obtain:

$$REQ_{time,ncp,ub} = \sum_{j=1}^{R(u)} \frac{n^{R(u)-j+1}-1}{n-1} = \sum_{j=1}^{R(u)} \frac{n^{R(u)}}{n^{j-1}*(n-1)} - \sum_{j=1}^{R(u)} \frac{1}{n-1} = \frac{n^{R(u)}}{(n-1)} * \sum_{j=1}^{R(u)} \frac{1}{n^{j-1}} - \frac{R(u)}{(n-1)}$$

The last geometric series can be solved by substituting $j = (x+1)$, obtaining

$$REQ_{time,ncp,ub} = \frac{n^{R(u)}}{(n-1)} * \sum_{x=0}^{R(u)-1} \frac{1}{n^x} - \frac{R(u)}{(n-1)} = \frac{n^{R(u)}}{(n-1)} * \frac{(1/n)^{R(u)}-1}{(1/n)-1} - \frac{R(u)}{(n-1)}$$

Finally, by further development and simplification we obtain the following expression for the upper bound on request time:

$$REQ_{time,ncp,ub} = \frac{n^{R(u)+1} - n * R(u) + R(u) - n}{(n-1)^2}$$

If $P_r \geq 1$ the above term need to be multiplied by P_r, since in our model links can be simultaneous traversed only by one packet, yielding to the following expression:

$$REQ_{time,ncp,ub} = \begin{cases} \dfrac{n^{R(u)+1} - n * R(u) + R(u) - n}{(n-1)^2} & 0 \leq P_r < 1 \\ P_r * \dfrac{n^{R(u)+1} - n * R(u) + R(u) - n}{(n-1)^2} & P_r \geq 1 \end{cases}$$

A more optimistic upper bound on request time is represented by the case in which request packets are sent first to those nodes that are further from the station and then incrementally to the closer ones. For the example of Figure 7-5a the request time will be the time to transmit all messages through the first link that would add up to 7 units of time.

The time taken to the response packets to travel back to the monitoring station is equal to the previous upper bound on request time. This can be inferred by observing the example of Figure 7-5b. In this case the total time will be the time to transmit 1 packet between levels 3-2, plus the time to transmit 3 packets between levels 2-1, plus the time to transmit 7 packets between levels 1-0. This will add up to 11 units of time.

The total monitoring time will be obtained by combining the ‘request’ term with the ‘response’ one. If we follow the network model suggested by Rescigno, no more than one packet can simultaneously traverse a link [Rescigno 97]. In that case, the ‘request’ and the ‘response’ phases tend to have a very limited overlapping and, hence, an upper bound on monitoring time is obtained by adding the two terms. This leads to the following two expressions:

$$\boxed{\begin{aligned} & MONIT_{time,ncp,ub} \leq REQ_{time,ncp,ub} + RESP_{time,ncp,ub} = \\ & \qquad = \begin{cases} 2 * \dfrac{n^{R(u)+1} - n * R(u) + R(u) - n}{(n-1)^2} & 0 \leq P_r < 1 \\ 2 * P_r * \dfrac{n^{R(u)+1} - n * R(u) + R(u) - n}{(n-1)^2} & P_r \geq 1 \end{cases} \end{aligned}}$$

$$MONIT_{time,ncp,oub} \le REQ_{time,ncp,oub} + RESP_{time,ncp,oub} = \\ = \begin{cases} \dfrac{n^{R(u)} + n*R(u) - R(u)}{n-1} + \dfrac{n^{R(u)+1} - n*R(u) + R(u) - n}{(n-1)^2} & 0 \le P_r < 1 \\ P_r * \left[\dfrac{n^{R(u)} + n*R(u) - R(u)}{n-1} + \dfrac{n^{R(u)+1} - n*R(u) + R(u) - n}{(n-1)^2} \right] & P_r \ge 1 \end{cases}$$

It should be observed that this model provides only a first approximation of the actual monitoring process, according to the assumed half-duplex communication model. In this case, links are used in mutual exclusion by packets (not more than one packet can simultaneously traverse a link). In practice, ‘response’ packets travel in the opposite direction of ‘request’ ones. The two processes can, then, take place with a larger extent of parallelism than the one captured by the above model, which would result in smaller monitoring time. With this respect, the models reflect the worse-case scenario, providing upper bounds on monitoring time. The choice of the half-duplex rather than full-duplex link model has been taken in order to build models that could be compared with those presented in [Rescigno 97] where the half-duplex model is adopted.

To evaluate the asymptotic behaviour of response time we consider the above optimistic upper bound, which operates at

$$O\left(P_r * \left(n^{R(u)-1} + {R(u)}/{n} \right) \right)$$

For the particular network topology assumed here, we have that $n^{R(u)} \propto N$ whereby N is the number of monitored nodes. Hence:

$$MONIT_{time,ncp,oub} \propto O(P_r * (N + R(u)))$$

7.3 Steady-state Models of Optimal Centralised Polling under Near-Optimal Conditions

The naïve approach to perform polling by first issuing request packets and then gathering the responses sums up to a poor polling algorithm. Rescigno has proposed a more efficient algorithm in which the polling station sends a request only to its neighbour nodes that, in turn, duplicate the request and forward it to their respective neighbours [Rescigno 97]. This process proceeds similarly to a *multicasting* model until all nodes have been reached by a request.

A more formal description of this algorithm, denominated *optimal centralised polling algorithm*, is abstracted below from the original work proposed by Rescigno. This denomination derives from the fact that the author has proved that this algorithm minimises the response time of a polling-based monitoring system for a particular class of network topologies. After describing this algorithm, we prove that networks admitting *n*-ary balanced spanning trees fall in this class. Finally, we derive models for polling traffic and response time.

7.3.1 Optimal Centralised Polling Algorithm

Let $T_u(G)$ be any spanning tree of G rooted at u, whereby $G=(V, E)$ is a graph and u the monitoring station node. Assume that each node knows the graph topology and the identity of polling station, u and that each node x constructs by itself the same spanning tree $T_u(G)$ rooted at u. This will be the case if the procedure which constructs $T_u(G)$ is identical at the nodes. The procedure performed by each node x in a polling execution, using $T_u(G)$, is as follows.

Node x waits until receiving the query from its parent in $T_u(G)$; then it sends its response to the parent and delivers the query to its sons in $T_u(G)$. When x receives responses from its sons, its sends them, one at a time, to its parent. Therefore, the query travels from the polling station u to the leaves in $T_u(G)$, while each response propagates from any node to u along the edges of $T_u(G)$. The polling algorithm that uses $T_u(G)$ is shown in Listing 7-1, which is copied from [Rescigno 97]. We denote by Q the query that u sends to every node in G and by R_x the response that node $x \in V \setminus \{u\}$ transmits back to u.

```
FOR $x \in V$ DO in parallel
   IF $Q$ is received THEN
      in the next step, send $Q$ to all the sons
      in $T_u(G)$ and send $R_x$ to the parent in
      $T_u(G)$

   WHILE there are responses of descendents DO
      send a response to the parent in $T_u(G)$
END FOR
```

Listing 7-1: Rescigno's Optimal centralised polling algorithm.

Rescigno has proved that the algorithm is optimal – *i.e.*, minimises overall polling time – for the class of networks admitting a particular spanning tree, called *Polling Tree*, $PT_u(G)$ defined as follows.

DEFINITION:

A *Polling Tree* of G rooted at *u*, $PT_u(G)$ is any rooted spanning tree of *G* with root *u* having *d(u)* main subtrees, $T_r=(V(T_r),E(T_r))$ for $r\in N(u)$, whereby *N(u)* is the set of neighbour nodes of *u*, *N* is the total number of nodes in the graph and each T_r has the following properties:

IF (2 * *R(u)*) > $\lceil (N\text{-}1)/d(u) \rceil$ + 2 THEN

1.1) height(T_r) ≤ (*R(u)* – 1);

1.2) there are at most two nodes in T_r at depth *l*, for each *l* ≤ (height(T_r) - 1); there is exactly one node at depth *l* = height(T_r);

OTHERWISE

1.3) $|V(T_r)| \in \{ \lceil (N\text{-}1)/d(u) \rceil, \lfloor (N\text{-}1)/d(u) \rfloor \}$;

1.4) there are at least two nodes at each depth *l* ≤ (height(T_r) - a_r)

$$a_r = \begin{cases} 1 & if\left|V(T_r)\right| = \left\lceil \dfrac{n-1}{d(u)} \right\rceil, \\ 2 & if\left|V(T_r)\right| \neq \left\lceil \dfrac{n-1}{d(u)} \right\rceil. \end{cases}$$

We now use Rescigno’s findings to prove the following theorem.

THEOREM 2:

Any rooted spanning tree of *G* with root *u*, having *d(u)* main sub-trees, $T_r=(V(T_r),E(T_r))$ for $r\in N(u)$, with every T_r being *n*-ary balanced trees, is also a polling tree, $PT_u(G)$.

PROOF:

If a graph admits a spanning tree with *d(u)* *n*-ary balanced trees T_r we have that the total number of nodes is

$$N = d(u) * \sum_{i=0}^{R(u)-1} n^i = d(u) * \frac{n^{R(u)}-1}{n-1}. \text{ Thus,}$$

$$\left\lceil \frac{N-1}{d(u)} \right\rceil + 2 = \left\lceil \frac{1}{d(u)} * \left(d(u) * \frac{n^{R(u)}-1}{n-1} - 1 \right) \right\rceil + 2 > 2 * R(u)$$

as can be shown by solving the above disequation. An analytical solution of the above disequation presents some difficulties due to the presence of the *ceiling* function. A graphical solution is shown in Figure 7-6, where

$$\left\lceil \frac{1}{d(u)} * \left(d(u) * \frac{n^{R(u)}-1}{n-1} - 1 \right) \right\rceil + 2 - 2 * R(u)$$

is shown to be positive and monotonically increasing when *n* and *R(u)* grow.

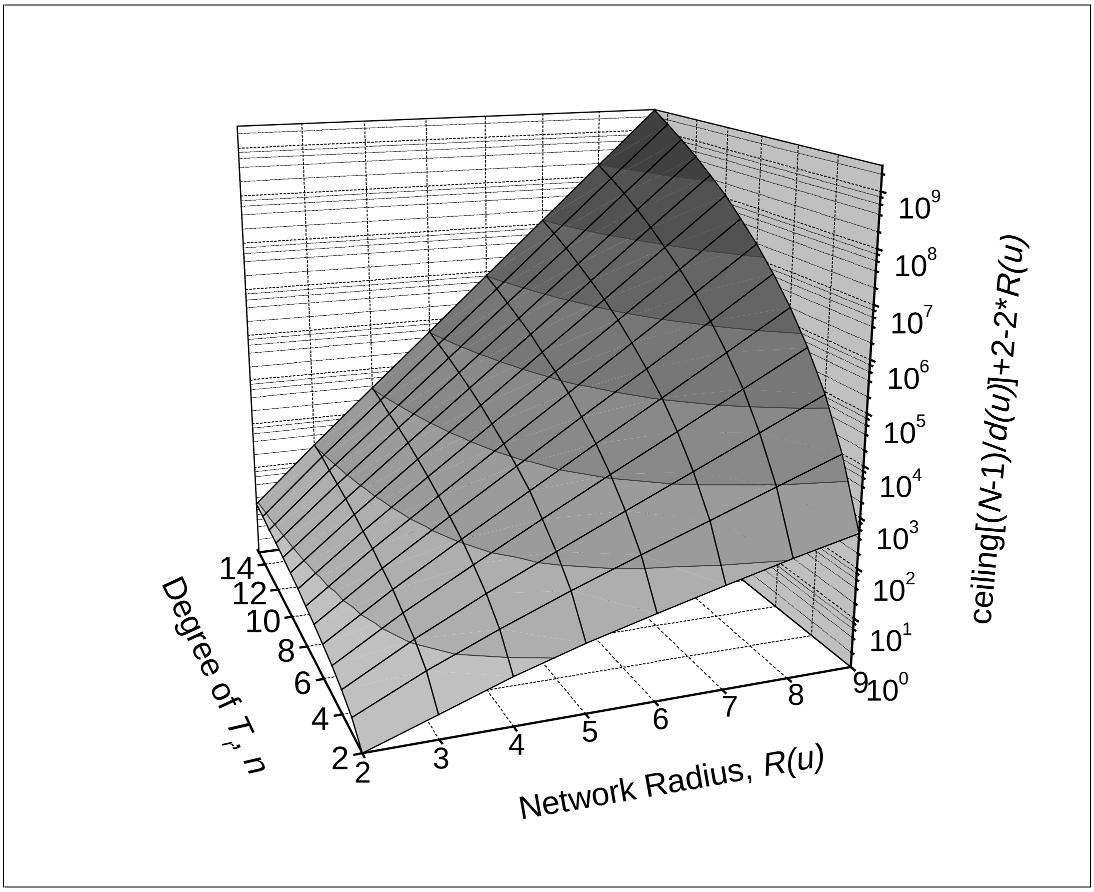


Figure 7-6: Graphical solution of the disequation which proves theorem 2.

Thus, conditions 1.3) and 1.4) need to be satisfied to prove the theorem.

The spanning tree of *G* is assumed *n*-ary and balanced, as depicted in Figure 7-2. Hence, the number of nodes in each sub-tree, T_r will be $|V(T_r)| = (N-1) / d(u)$, which satisfies condition 1.3). As far as condition 1.4) is concerned, a_r=1 since $|V(T_r)| = (N-1) / d(u) = \lceil (N-1) / d(u) \rceil$. The number of nodes at level $l \leq$ (height(T_r)- a_r) = (height(T_r)-1) is certainly at least equal two 2 since the each T_r is assumed *n*-ary with *n*>1. Therefore, condition 1.4) is satisfied as well and we can conclude that the set of sub-trees T_r, $r \in \{1, \ldots, d(u)\}$ is a polling tree for *G* and, in turn, Rescigno’s polling algorithm is optimal for spanning trees having *n*-ary balanced sub-trees.

This algorithm can be further illustrated with the simple example depicted in Figure 7-7 in which the packet transmission involved during the multicast and response process is shown for the case of a spanning tree admitting binary sub-trees. The calculation of traffic and response time for the general case in which T_r is *n*-ary are calculated below.

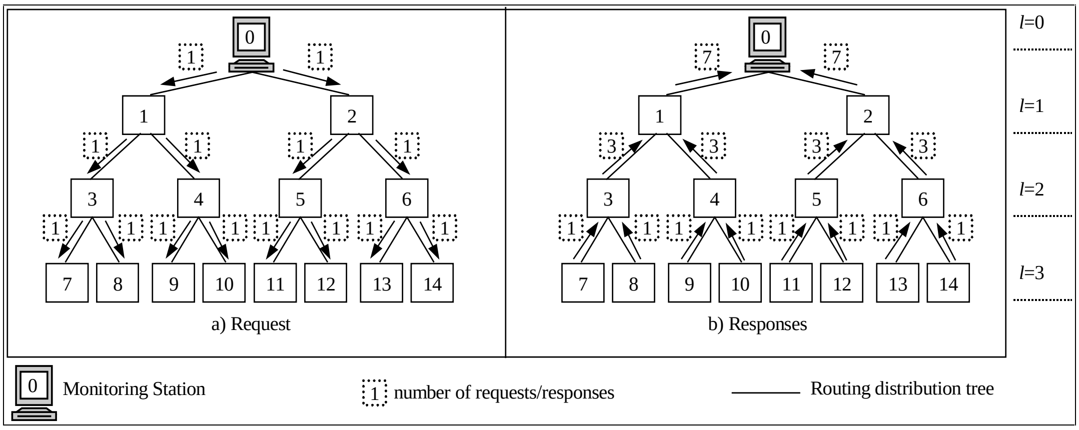


Figure 7-7: Example of centralised optimal polling under near-optimal conditions.

7.3.2 Steady Traffic in Optimal Centralised Polling

This algorithm allows us to reduce significantly the number of request packets. In fact, there will be only one packet per link during a polling operation. Thus, for each sub-tree, T_r the number of packets multiplied by the links traversed by those packets will be $\sum_{i=0}^{R(u)-1} n^i = \left(n^{R(u)}-1\right)/\left(n-1\right)$ and the total multicast traffic will be:

$$MULT_{traff,ocp} = d(u) * P_r * 8 * b_b * \frac{n^{R(u)}-1}{n-1}$$

The response packets will incur the same traffic as for the naïve algorithm, that is

$$RESP_{traff,ocp} = \sum_{l=0}^{R(u)-1} \left(P_r * 8 * b_r * d(u) * \sum_{i=l}^{R(u)-1} n^i \right)$$

Following the same procedure detailed in Section 7.2.1 (page 147) we obtain

$$RESP_{traff,ocp} = d(u) * P_r * 8 * b_r * \left\{ \frac{R(u) * n^{R(u)} * (n-1) - n^{R(u)} + 1}{(n-1)^2} \right\}$$

Therefore, the monitoring traffic incurred by the optimal centralised polling algorithm will be:

$$\begin{aligned} & MON_{traff,ocp} = MULT_{traff,ocp} + RESP_{traff,ocp} = \\ & = d(u) * P_r * 8 * b_b * \frac{n^{R(u)}-1}{n-1} + d(u) * P_r * 8 * b_r * \left\{ \frac{R(u) * n^{R(u)} * (n-1) - n^{R(u)} + 1}{(n-1)^2} \right\} = \\ & = d(u) * P_r * 8 * (b_b + b_r) * \left\{ \frac{n^{R(u)}-1}{n-1} + \frac{R(u) * n^{R(u)} * (n-1) - n^{R(u)} + 1}{(n-1)^2} \right\} \end{aligned}$$

As far as scalability is concerned, the above expression operate at:

$$\boxed{MON_{traff,ocp} \propto O\left(P_r * R(u) * n^{R(u)}\right)}$$

For the particular network topology assumed here, we have that $n^{R(u)} \propto N$ whereby N is the number of monitored nodes. Hence:

$$\boxed{MON_{traff,ocp} \propto O\left(P_r * R(u) * N\right)}$$

Which shows that despite incurring less traffic than naïve polling, optimal polling does not result in increased scalability. This is due to the fact that, although the multicasting approach to sending request packets is more efficient than the mechanism used in naïve polling, response mechanisms are essentially the same. In other words, the response process predominates over the request process in the case of optimal polling. Therefore, the savings associated to optimal polling are mostly related the request phase.

7.3.3 Steady Response Time in Optimal Centralised Polling

We have already proved with Theorem 2 that Rescigno's polling algorithm is optimal for graphs admitting spanning trees which contain *d(u)* *n*-ary balcanced sub-trees. We can, then, reuse the expression of response time presented in [Rescigno 97], that is:

$$MON_{time,ocp} = \begin{cases} 2 & if\left(R(u)=1\right) \\ \max\left\{\left\lceil \dfrac{(N-1)}{d(u)} \right\rceil + 2; \quad 2 * R(u)\right\} & if\left(R(u) \geq 2\right) \end{cases}$$

This expression assumes one polling operation per unit of time, that is P_r=1.

In general, this means that response time operates at O(P_r, N, $R(u)$). It should be noted that the dependency on P_r is related to the way links are modelled *i.e.* links can be traversed by one packet at the time (packets have mutually exclusive access to links). Thus, increasing values of polling rate result in linearly increasing queuing delays.

In Theorem 2 we have also illustrated that

$$\left\lceil \frac{N-1}{d(u)} \right\rceil + 2 = \left\lceil \frac{1}{d(u)} * \left(d(u) * \frac{n^{R(u)} - 1}{n-1} - 1 \right) \right\rceil + 2 > 2 * R(u)$$

Hence for P_r=1,

$$MON_{time,ocp} = \begin{cases} 2 & if\left(R(u)=1\right) \\ \left\lceil \dfrac{(N-1)}{d(u)} \right\rceil + 2 & if\left(R(u) \geq 2\right) \end{cases}$$

In our case, (N-1)/$d(u)$ is the total number of nodes in T_r, since all T_r are assumed n-ary and balances and, then, equal. Hence, $\lceil (N-1)/d(u) \rceil = (N-1)/d(u) = (n^{R(u)}-1)/(n-1)$. Finally, by simple developments of the terms we obtain:

$$MON_{time,ocp} = \begin{cases} \left\{ \begin{array}{ll} 2 & if\left(R(u)=1\right) \\ \dfrac{n^{R(u)}+2*n-3}{n-1} & if\left(R(u)\geq 2\right) \end{array} \right\} & 0 \leq P_r < 1 \\ \left\{ \begin{array}{ll} 2 & if\left(R(u)=1\right) \\ P_r * \dfrac{n^{R(u)}+2*n-3}{n-1} & if\left(R(u)\geq 2\right) \end{array} \right\} & P_r \geq 1 \end{cases}$$

The above function operates at

$$MON_{time,ocp} \propto O\left(P_r * n^{R(u)-1}\right) \propto O\left(P_r * N\right)$$

Capturing the behaviour as a function of $R(u)$ is not easy under the particular assumptions. If we look at the original formulation of $MON_{time,ocp}$ given in page 155 above, the function max{., $R(u)$} suggests an operation which is at most O($R(u)$). We can then conclude that monitoring time operates at most as

$$MON_{time,ocp} \propto O\left(P_r, N, R(u)\right)$$

which, if performance indicators are considered individually, is characterised by the same level of scalability as its naïve polling counterpart. This is due to fact that, although the multicasting approach to sending request packets is more efficient than the mechanism used in naïve polling, response mechanisms are essentially the same. In other words, the response process predominates over the request process in the case of optimal polling. Therefore, the savings associated to optimal polling are mostly related to the request phase.

7.4 Steady-state Models of Agent-based Distributed Polling under Near-Optimal Conditions

The remaining part of this chapter is dedicated to the analysis of the agent-based distributed monitoring approach proposed in Chapter 4, under the near-optimal conditions given in Section 7.1. The following sub-sections present models for assessing performance and scalability at steady-state, that is after agent deployment is completed. The transient analysis, including the assessment of the performance overheads incurred during agent deployment is the subject of Section 7.6.

The agent-based monitoring algorithm has already been discussed extensively in the previous chapters. The centralised and distributed versions of agent deployment do not differ as far as the steady-state analysis is concerned since they both result in identical agent locations. Hence, one steady-state model will be derived. Conversely, transient models will be derived separately.

Figure 7-8 illustrates the three steps of agent-based monitoring. First, each agent issues poll requests to every node of its partition (Figure 7-8a). This is done according to a point-to-point ‘request issuing’ model. For brevity, we refer to this step as ‘request’ step. An alternative possibility would be to multicast the *polls*. This may lead to reductions in traffic and, presumably, time. However, it would be associated with the additional delay involved in the construction of the multicast trees (one per network partition or per agent). This overhead may not suit the requirements of the assumed dynamic networked system where agents may frequently change their location to provide adaptation. If a multicasting approach were adopted for issuing polls, agent migration overheads would be aggravated by the need to re-build multicast trees. Therefore, multiple unicasting is used within each network partition to aim at simplicity and reduced agent migration latency, assuming relatively fine-grained network partitions – *i.e.* the number of monitored nodes per agent is kept relatively small for efficiency reasons.

The second step of agent-based monitoring involves the issuing of response packets by the ‘polled’ nodes (Figure 7-8b). Each node sends a unique response to the requesting agent following the *unicast* model. This is termed the ‘response’ step.

Finally, each agent pre-processes the collected information and informs the monitoring station (Figure 7-8c). We term this last phase the ‘delivery’ step. It is here that data semantic compression or aggregation takes place. Delivery can be performed according to three different modalities: 1) the monitoring station can query agents using a polling-based approach; 2) agents periodically notify results to the monitoring station (notification-based approach); or 3) agents send notifications to the monitoring station only under specific circumstances, such as threshold crossing (alarm-based approach).

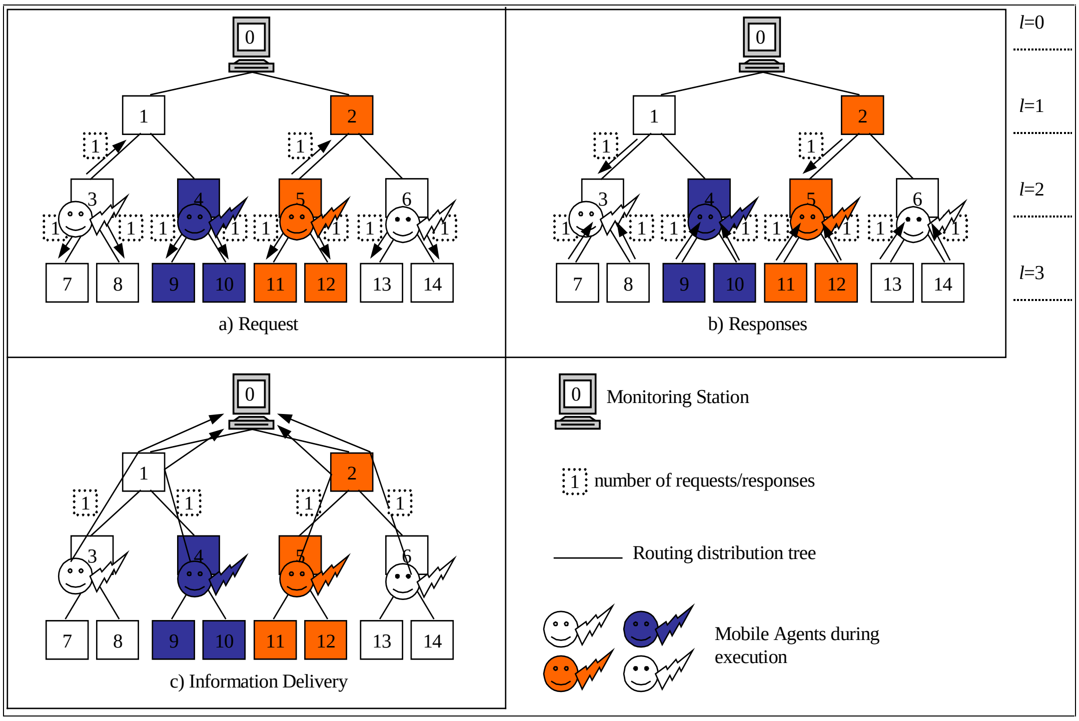


Figure 7-8: Example of distributed agent-based polling under near-optimal conditions.

No matter which modality is used to deliver information to the central station, the amount of traffic exiting the agent can be assumed to be significantly smaller than the traffic incurred to collect the raw information in the first place (through the first and second steps). Therefore, the agent system brings two advantages in comparison to centralised polling: first, it reduces the data collection traffic by offering decentralised polling stations; second, it operates local data semantic compression at the agent location. The models presented in the next sections will help quantifying these advantages.

7.4.1 Steady Traffic in Distributed Agent-based Polling

If we refer back to Figure 7-4 (page 143), the traffic incurred by a generic agent MA_x is the result of two contributions introduced by the nodes of N_x and N_0 respectively. Hence, the total request traffic can be expressed as:

$$REQ_{traff,MA} = \sum_{x=1}^{p} MA_x = \sum_{x=1}^{p} P_r * d(u) * 8 * b_b * \left(S_{d,N_x} + S_{d,N_0}\right)$$

whereby $S_{d,Nx}$ and $S_{d,N0}$ are defined in Section 7.1. Hence by simply substituting the above expressions we abtain:

$$REQ_{traff,MA} = P_r * d(u) * 8 * b_b * \left(\frac{(R(u)-L) * n^{(R(u)-L+2)} - (R(u)-L+1) * n^{(R(u)-L+1)} + n}{(n-1)^2} + \frac{p * L}{2 * (n-1)} \right) =$$

$$= P_r * d(u) * 8 * b_b * \left(\frac{(R(u)-L) * n^{(R(u)-L+2)} - (R(u)-L+1) * n^{(R(u)-L+1)} + n}{(n-1)^2} + \frac{d(u) * n^{L-1} * L}{2 * (n-1)} \right)$$

Similarly, the total traffic incurred by the response packets travelling back to agents can be expressed as:

$$RESP_{traff,MA} = \sum_{x=1}^{p} MA_x = \sum_{x=1}^{p} P_r * d(u) * 8 * b_r * \left(S_{d,N_x} + S_{d,N_0} \right) =$$

$$= P_r * d(u) * 8 * b_r * \left(\frac{(R(u)-L) * n^{(R(u)-L+2)} - (R(u)-L+1) * n^{(R(u)-L+1)} + n}{(n-1)^2} + \frac{d(u) * n^{L-1} * L}{2 * (n-1)} \right)$$

which differs from $REQ_{traff,MA}$ only in the size of the packets, b_r.

The delivery traffic between agents and monitoring station depends on the rate notification packets are sent by agents which we term notification rate, Not_r. Not_r is a deterministic value if agents are notification-based. Conversely, it will be a probabilistic value if agents are alarm-based. In any case $Not_r \leq P_r$ and, typically, $Not_r << P_r$.

Assuming that the size of packets sent to the monitoring station is b_{not} the delivery traffic of a single agent is given by the bit rate multiplied by the number of traversed links ($8*b_{not}*Not_r*L$). Hence the traffic incurred by p agents will be:

$$DELIV_{traff,MA} = p*8*b_{not}*Not_r*L = 8 * b_{not} * d(u) * Not_r * L * n^{L-1}$$

Finally, the overall monitoring traffic will be given by the sum of the above terms:

$$\boxed{\begin{aligned} & MONIT_{traff,MA} = REQ_{traff,MA} + RESP_{traff,MA} + DELIV_{traff,MA} = \\ & = P_r * d(u) * 8 * (b_b + b_r) * \left(\frac{(R(u)-L) * n^{(R(u)-L+2)} - (R(u)-L+1) * n^{(R(u)-L+1)} + n}{(n-1)^2} + \frac{d(u) * n^{L-1} * L}{2 * (n-1)} \right) + \\ & \qquad\qquad + 8 * b_{not} * d(u) * Not_r * L * n^{L-1} \end{aligned}}$$

which operates at

$$O\left(P_r * \left\{ (R(u)-L) * n^{R(u)-L} + L * n^{L-2} \right\} + Not_r * L * n^{L-1} \right)$$

For the particular network topology assumed here, we have that: $n^{R(u)} \propto N$, whereby N is the number of monitored nodes; $n^L \propto p$, whereby p is the number of MAs; and $L \propto \log(p)$. Hence:

$$MONIT_{traff,MA} = O\left(P_r * \left\{ (R(u) - \log p) * (N - p) + p * \log p \right\} + P_r * p * \log p \right)$$

That is

$$MONIT_{traff,MA} = O(P_r, R(u), N, p \log p)$$

If we consider the performance indicators individually, the asymptotic behaviour of the agent approach is analogous to that of the centralised approaches. However, the agent approach as a whole incurs less traffic, as will result from the comparative analysis reported in Section 7.5.1.

7.4.2 Steady Response Time in Distributed Agent-based Polling

Similarly to monitoring traffic, response time results from three contributions: request issuing, collection of responses by the agents, and delivery of information from agent to monitoring station (Figure 7-8). Agents request, response, and delivery packets traverse mutually exclusive paths under near-optimal conditions. Thus, all agents are characterised by the same response time.

The time taken for an agent to issue poll requests (to all the nodes belonging to its network partition) is the maximum between the contribution due to N_x and N_0, respectively. That is, for $0 \leq P_r < 1$:

$REQ_{time,MA} = \max \{ REQ_{time,MA,Nx} , REQ_{time,MA,No} \}$

Similarly,

$RESP_{time,MA} = \max \{ RESP_{time,MA,Nx} , RESP_{time,MA,No} \}$

$REQ_{time,MA,Nx}$ can be calculated similarly to the way the request time of the naïve algorithm has been calculated (Section 7.2.2). Agents have access to routing information and can, then, be programmed to send poll requests first to the further nodes. Hence, the upper bound on request time equivalent to the optimistic upper bound described in Section 7.2.2. This is the number of time units to send $\sum_{j=0}^{R(u)-L-1} n^j$ from level L to (L+1) plus further ($R(u)$-L-1) units of time to transmit the packet in parallel through the remaining levels. It is assumed that a packet takes a unit of time to traverse any link – *i.e.* links are all equal. Thus,

$$REQ_{time,MA,N_x} = \sum_{j=0}^{R(u)-L-1} n^j + R(u) - L - 1 = \frac{n^{R(u)-L} - 1}{n-1} + R(u) - L - 1$$

$REQ_{time,MA,Nx}$ is at most equal to the time to reach the farthest node in N_0 which is (L-1) units of time. This is also the upper bound on $REQ_{time,MA,No}$. Thus, an upper bound on request time will be:

$$\boxed{REQ_{time,MA} = \begin{cases} \max\left\{\dfrac{n^{R(u)-L}-1}{n-1}+R(u)-L-1; \quad (L-1)\right\} & 0 \le P_r < 1 \\ P_r * \max\left\{\dfrac{n^{R(u)-L}-1}{n-1}+R(u)-L-1; \quad (L-1)\right\} & P_r \ge 1 \end{cases}}$$

RESP$_{time,MA,Nx}$ can be calculated similarly to the pessimistic upper bound of Section 7.2.2, yielding to the following expressions:

$$RESP_{time,MA,N_x} = \sum_{j=0}^{R(u)-L-1} n^j + \sum_{j=0}^{R(u)-L-2} n^j + \ldots = \frac{n^{R(u)-L+1} - n*(R(u)-L)+(R(u)-L)-n}{(n-1)^2}$$

$$\boxed{RESP_{time,MA} = \begin{cases} \max\left\{\dfrac{n^{R(u)-L+1} - n*(R(u)-L)+(R(u)-L)-n}{(n-1)^2}; \quad (L-1)\right\} & 0 \le P_r < 1 \\ P_r * \max\left\{\dfrac{n^{R(u)-L+1} - n*(R(u)-L)+(R(u)-L)-n}{(n-1)^2}; \quad (L-1)\right\} & P_r \ge 1 \end{cases}}$$

The time units required to deliver a notification between agent and monitoring station is the time to deliver (*p*/*d(u)*)=2$^{L\text{-}1}$ messages from level *L* back to level 0. The bottleneck is represented by the links connecting level 0 with level 1 since those are the ones carrying the largest number of messages. Lower levels will carry portions of the total messages. Hence, the delivery time is the time to transmit n$^{L\text{-}1}$ packets through the bottleneck plus the time for the first packet to reach level 1, which is equal to (*L*-1); yielding to the following expression:

$$\boxed{DELIV_{time,MA} = DELIV_{time,MA_x} = \begin{cases} 0 & L \cong 0 \\ \begin{cases} n^{L-1} + L - 1 & 0 \le Not_r < 1 \\ Not_r * \left(n^{L-1} + L - 1\right) & Not_r \ge 1 \end{cases} & 0 < L \le R(u) \end{cases}}$$

The monitoring response time will be expressed by the sum of the above three terms, *REQ*$_{time,MA}$, *RESP*$_{time,MA}$, and *DELIV*$_{time,MA}$. For the scalability analysis the following expression hold:

$$REQ_{time,MA} \propto O\left(P_r * \max\left\{n^{R(u)-L-1} + R(u) - L - 1; (L-1)\right\} \propto P_r * \max\left\{n^{R(u)-L-1}; (L-1)\right\}\right)$$

$$RESP_{time,MA} \propto O\left(P_r * \max\left\{n^{R(u)-L-1}; (L-1)\right\}\right)$$

$$DELIV_{time,MA} \propto O\left(Not_r * n^{L-1}\right)$$

Hence,

$$\boxed{MON_{time,MA} \propto O\left(P_r * \max\left\{n^{R(u)-L-1}; (L-1)\right\} + Not_r * n^{L-1}\right); \quad 0 < L \le R(u)}$$

For the particular network topology assumed here, we have that: $n^{R(u)} \propto N$, whereby N is the number of monitored nodes; $n^{L} \propto p$, whereby p is the number of MAs; and $L \propto \log(p)$. Hence:

$$\boxed{MON_{time,MA} \propto O\left(P_r, R(u), N, p\right); \quad 0 < L \leq R(u)}$$

As for the traffic expression, if we consider the performance indicators individually, the asymptotic behaviour of the agent approach is analogous to that of the centralised approaches. However, the agent approach as a whole is characterised by reduced response time, as will result from the comparative analysis reported in Section 7.5.2.

7.5 Comparative Steady-state Analysis

In this section, we use the mathematical models presented in the previous sections to compare performance and scalability of centralised and distributed polling at steady-state. We do not consider here the agent deployment overheads, which are analysed in the following Sections 7.6 and 7.7. Traffic models and response time models are treated separately.

7.5.1 Analysis of Traffic Models at Steady State

The traffic models for the cases of centralised and agent-based distributed polling derived in the previous sections are depicted in Figure 7-9. Each graph represents a contour plot where ranges of traffic values are denoted on an XY grid using a greyscale map. X and Y axes represent increasing values of network radius and degree of the sub-trees of the spanning tree, respectively. Traffic ranges are in bit per second and are reported on a logarithmic scale. Darker tones of grey correspond to lower traffic ranges.

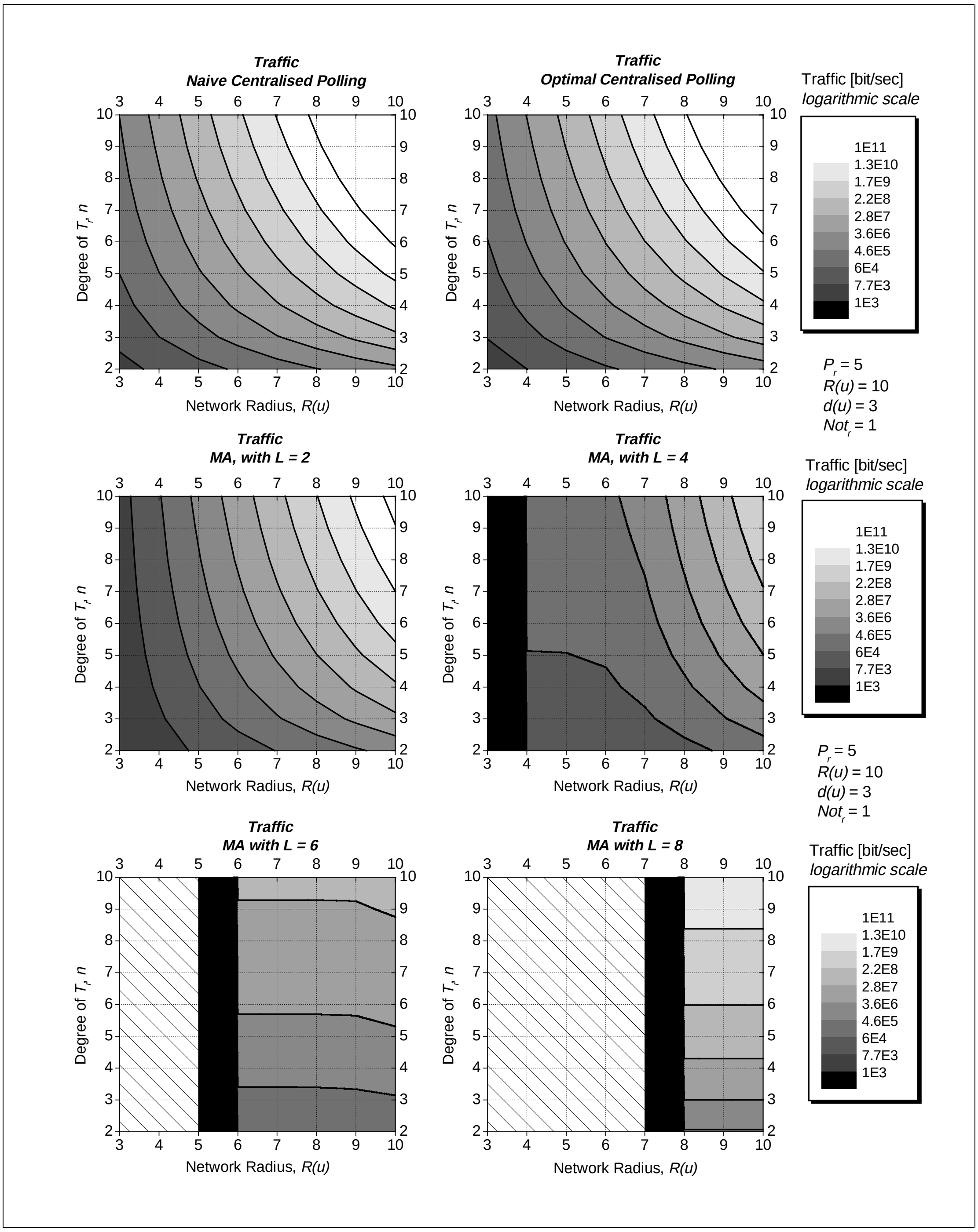


Figure 7-9: Contour plots depicting steady-state traffic for the cases of centralised and agent-based polling.

By comparing the top two graphs we can notice that ‘optimal’ centralised polling results in a very limited reduction in traffic, if compared with the naïve algorithm. In fact, both approaches lead to traffic values which fall in the same order of magnitude. This is due to the fact that the optimal algorithm still follows a centralised model and optimises only the request process but not the response one.

The remaining four graphs depict agent-based distributed polling for different configurations. *L* is the level at which agents are placed within the network spanning tree. Increasing values of *L* reflect configurations with proportionally increasing number of agents. In fact, the number of agents is expressed as $p=d(u)*n^{L-1}$. From the shift in ranges between the centralised and the agent-based solutions it can be noticed that the latter results in traffic values which are 1 to 3 orders of magnitude smaller, depending on the number of agents. This reduction in traffic is a direct consequence of the distributed nature of the agent approach.

The traffic profiles of the various solutions are captured from a different perspective in Figure 7-10, which represents both centralised and agent-based polling in a single graph for a better comparison. Traffic is plotted against increasing values of *n* (the degree of the spanning tree) and *R(u)* (network radius). That is, traffic against network scale is reported. In particular, the independent values are projected to the *W*-axis, which can be defined as: $W : \begin{cases} Z = 0 \\ Y = X \end{cases}$, whereby the *Z*-axis depicts traffic values, the *X*-axis depicts *n* values, and the *Y*-axis depicts *R(u)* values. In other words, Figure 7-10 represents the 45°-section through the *Z*-axis in the *XY* plane.

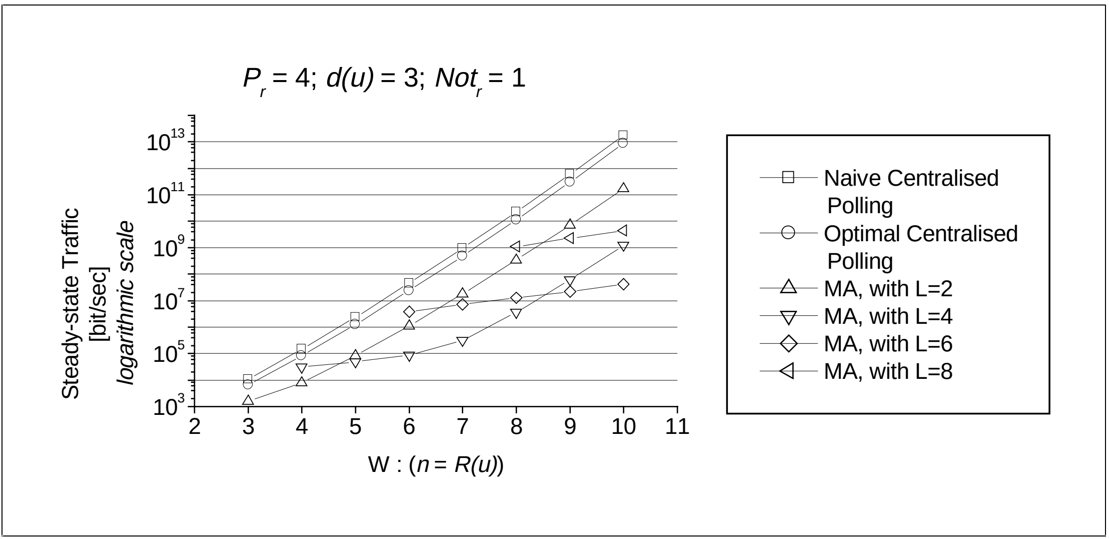


Figure 7-10: 45°-section through the Z axis (Steady-state Traffic) in the *n-R(u)* plane.

Two observations can be made. First, all agent configurations perform and, all in all, scale better than the two centralised algorithms. Second, agent performance and scalability improve with *L* until a given point and start degrading for *L* larger than a threshold value. This particular aspect is more evident from Figure 7-11, which plots steady-state traffic against *L* (keeping all other parameters constant).

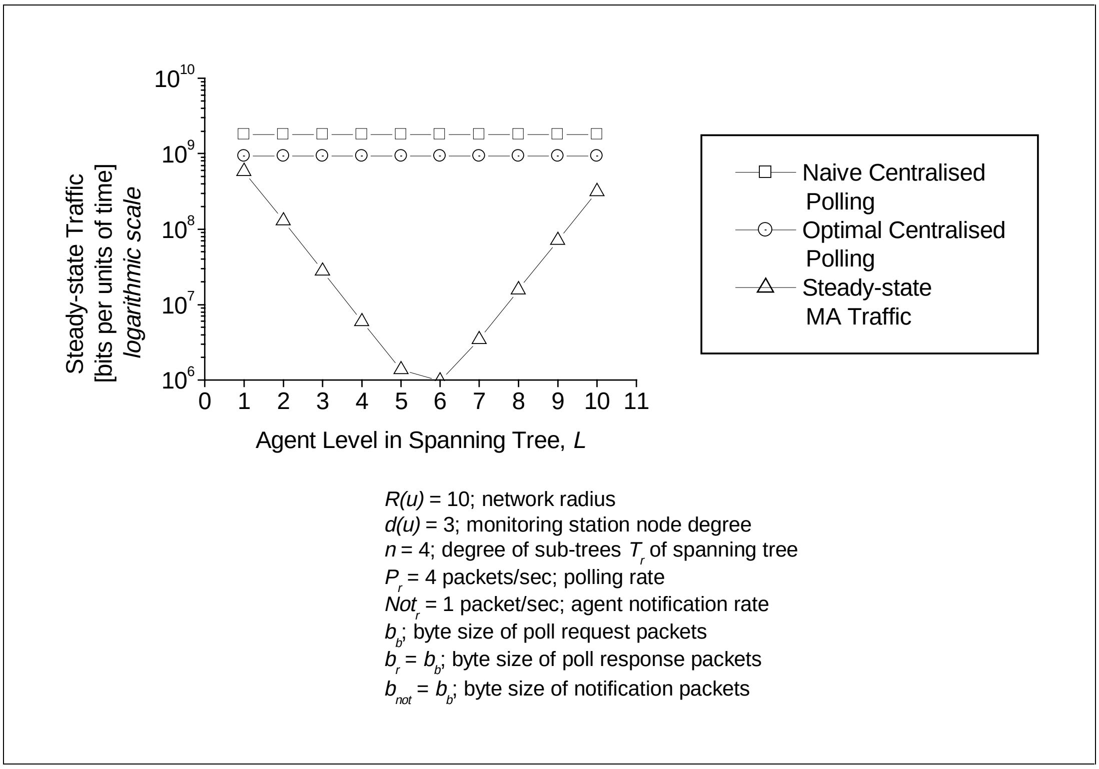


Figure 7-11: Steady-state monitoring traffic of centralised and agent-based systems plotted against *L*.

Clearly, *L* affects only the agent solutions, since there are no agents in the case of centralised polling. That is why the traffic profiles are horizontal lines when centralised polling is adopted. In the agent approach steady-state monitoring traffic decreases sharply in the first range of *L*-values, reaches a local minimum (for the example configuration this is achieved for *L*=0.6**R(u)*) and then increases rapidly for larger values of *L*.

This result had not been anticipated before modelling the agent system but was, then, explained by looking more closely at the various contributions of the incurred traffic. Two contrasting factors contribute towards steady-state monitoring traffic. The first one is the traffic incurred by poll requests sent by the agents and their related responses. The second one is the delivery traffic that is incurred by the notification packets sent from the agents to the monitoring station. The traffic models presented in the above sections show that the first factor operates at

$$O\left(P_r * \left\{(R(u)-L) * n^{R(u)-L} + L * N^{L-2}\right\}\right)$$

whereas the second one operates at $O\left(Not_r * L * n^{L-1}\right)$. For relatively small values of *L*, the monotonically decreasing term $\left(P_r * \left\{(R(u)-L) * n^{R(u)-L}\right\}\right)$ predominates. On the contrary for relatively large values of *L*, the monotonically increasing term

$\left(P_r * L * N^{L-2} + Not_r * L * n^{L-1}\right)$ will predominate.

Figure 7-11 has been obtained by configuring the system in such a way as to obtain comparable contributions from those two terms in order to underline this particular feature of the agent system. In general, the 'delivery' term will be comparably smaller than the 'collection' one if delivery traffic is kept small. This may be achieved, for instance, with agents capable of performing significant data compression or when the agent notification rate (from agent to monitoring station) is negligible in comparison with the value of polling rate adopted by the agent for data collection.

From Figure 7-11 we can conclude that arbitrary agent configurations may lead to limited reduction in traffic. For instance, in the example agent configuration if the number of agents is relatively too small or too large (and we exclude the extreme conditions of L=0 or $L=R(u)$), there is only one order of magnitude difference in traffic between centralised and agent-based monitoring. To achieve reductions in traffic of up to three orders of magnitude the agent system should be configured in a way which corresponds to the local minimum of Figure 7-11.

Conclusions on scalability can be drawn by looking directly at the traffic models presented in the above sections, summarised in Table 7-1. Both centralised algorithms operate at $O(P_r * R(u) * N)$. If we consider the scalability indicators individually, we notice that the agent approach is characterised by the same level of scalability, despite as a whole, it scales and performs better. Agents scale better than the former for any L>0. In this case the agent approach results in an improvement in the order of n^L for $R(u) \rightarrow \infty$. Finally, as expected, for L=0 there will be no difference in scalability between the two approaches. This is the case in which all agents would be located at the monitoring station, which is equivalent to the centralised model.

	Asymptotic behaviour
Naïve centralised polling	$O\left(P_r * R(u) * n^{R(u)}\right) \propto O\left(P_r * R(u) * N\right)$
Optimal centralised polling	$O\left(P_r * R(u) * n^{R(u)}\right) \propto O\left(P_r * R(u) * N\right)$
Agent-based distributed polling	$O\left(P_r * \left\{\left(R(u) - L\right) * n^{R(u)-L} + L * N^{L-2}\right\}\right) + O\left(Not_r * L * n^{L-1}\right)$ $\propto O\left(P_r, R(u), N, p \log p\right)$

Table 7-1: Summary of steady-state traffic results.

7.5.2 Analysis of Response Time Models at Steady State

The analysis of the response time models is carried out similarly to the one of the traffic models and it will be shown that the results are qualitatively the same. Response time models for the cases of centralised and agent-based distributed polling are depicted in the contour plots of Figure 7-12. Ranges of response time values are denoted on a $n*R(u)$ grid using a greyscale map. Those values are expressed in units of time and are reported in a logarithmic scale. Darker tones of grey correspond to lower response times.

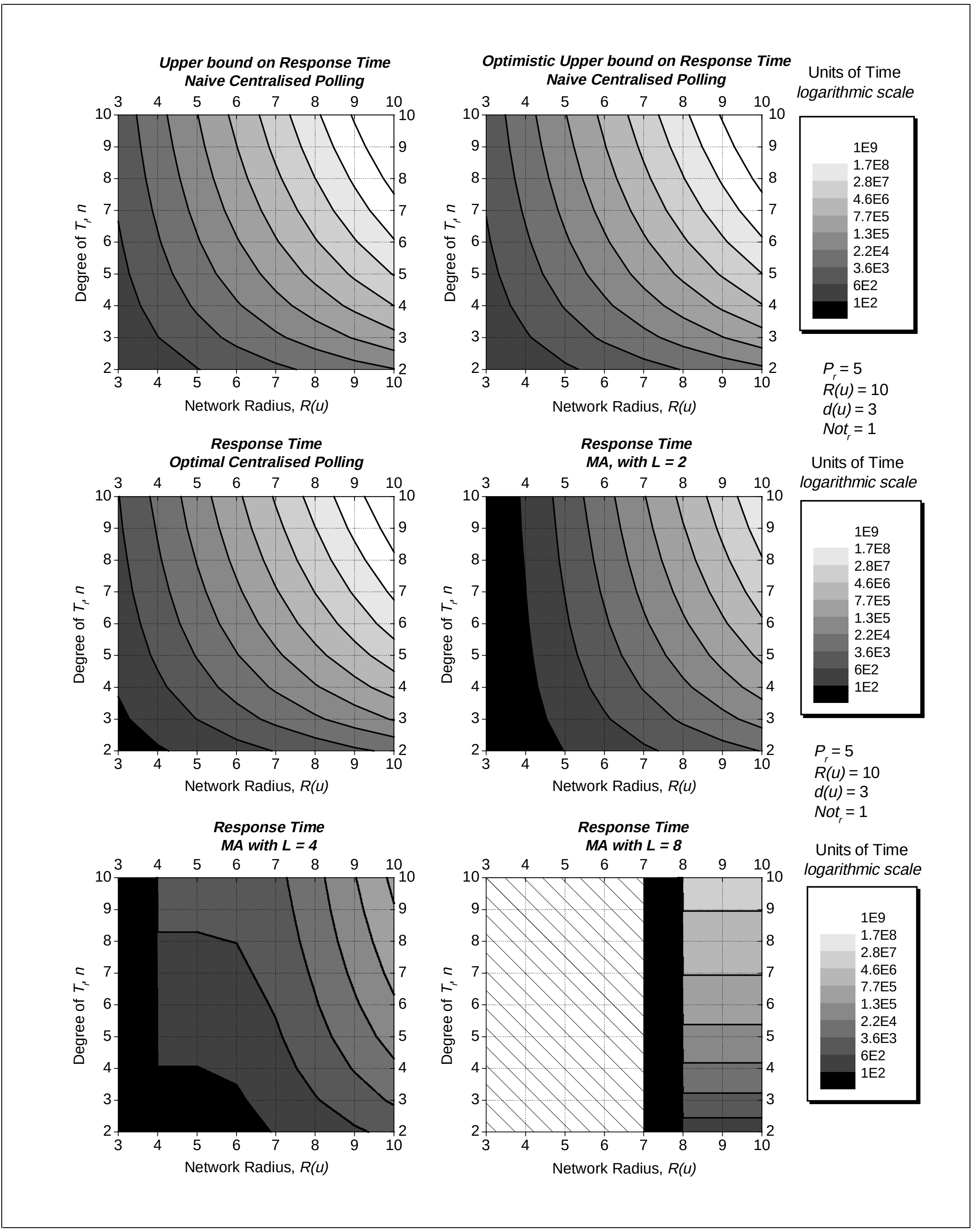


Figure 7-12: Contour plots depicting steady-state response time for the cases of centralised and agent-based polling.

The first two graphs represents two different upper bounds on the response time achieved with the naïve model. They are not markedly different. The profiles of optimal centralised polling are slightly better, whilst significant improvements are achieved with agents. Again, from the shift in ranges between the centralised and the agent-based solutions it can be noticed that the latter results in response time values which are 1 to 3 orders or magnitude smaller, depending on the number of agents.

Similar conclusions can be drawn from Figure 7-13, which combines all response time profiles in a single graph for a better comparison. This represents the 45º-section through the *Z*-axis in the *XY* plane, whereby the *Z*-axis depicts response times in a logarithmic scale, the *X*-axis depicts *n* values, and the *Y*-axis depicts *R(u)* values.

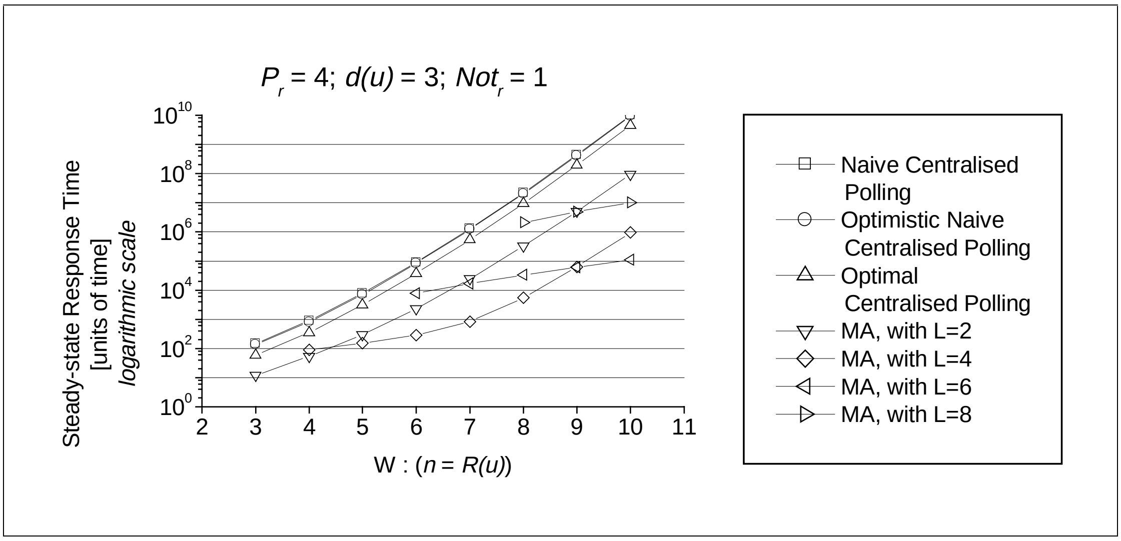
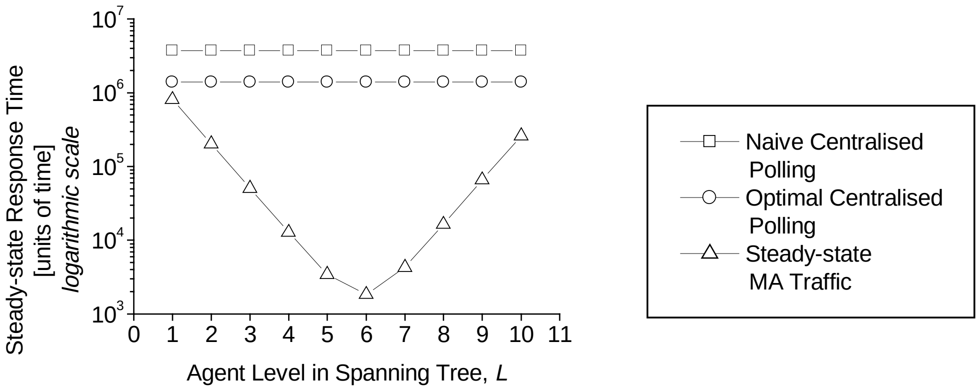


Figure 7-13: 45º-section through the Z axis (Steady-state response time) in the *n-R(u)* plane.

This figure exhibits the same qualitative behaviour as Figure 7-10 and, thus, does not need much further comment. The same applies to Figure 7-14 in which agent-based response time exhibits a local minimum, similarly to the traffic function of Figure 7-11.



R(u) = 10; network radius
d(u) = 3; monitoring station node degree
n = 4; degree of sub-trees T_r of spanning tree
P_r = 4 packets/sec; polling rate
Not_r = 1 packet/sec; agent notification rate
b_b; byte size of poll request packets
$b_r = b_b$; byte size of poll response packets
$b_{not} = b_b$; byte size of notification packets

Figure 7-14: Steady-state monitoring response time of centralised and agent-based systems plotted against *L*.

More quantitative conclusions can be drawn by looking directly at the response time models presented in the previous sections, summarised in Table 7-2. From the scalability viewpoint, if we consider the scalability indicators (P_r, *R(u)*, *N*, and *p*) separately, the three approaches are characterised by the same level of scalability, though as a whole, agents scale and perform better than centralised polling. Finally, as expected, for *L*=0 there will be no difference in scalability between the two approaches. This is the case in which all agents would be located at the monitoring station, which is equivalent to the centralised model.

	Asymptotic behaviour
Naïve centralised polling	$O\left(P_r * \left(R(u) + n^{R(u)}\right)\right) \propto O\left(P_r * \left(R(u) + N\right)\right)$
Optimal centralised polling	$O\left(P_r, N, R(u)\right)$
Agent-based distributed polling	$O\left(P_r * \max\left\{n^{R(u)-L-1}; (L-1)\right\} + Not_r * n^{L-1}\right)$ $\propto O\left(P_r, R(u), N, p\right);$

Table 7-2: Summary of steady-state response time results.

7.6 Transient Models of Distributed Agent-based Polling

The transient analysis concerns the study of agent deployment overheads, namely deployment traffic and deployment time. These have been already extensively discussed in the previous chapters for general network topologies. The models presented in the following sections allow for a more in-depth analysis of those overheads under the near-optimal conditions specified in Theorem 1.

Deployment overheads are calculated separately for the case of an agent capable of performing cloning and the one of an agent incapable of cloning. Figure 7-15 illustrates these cases for a simple case in which the graph admits spanning trees having balanced binary sub-trees T_r.

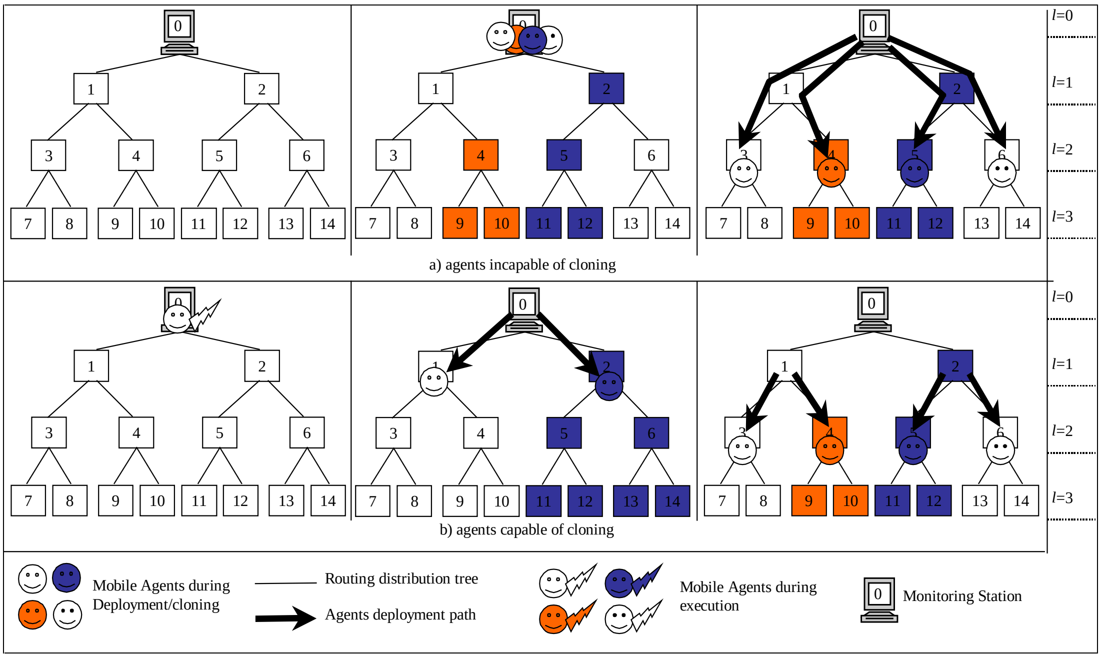

a) agents incapable of cloning
b) agents capable of cloning
l=0
l=1
l=2
l=3
Mobile Agents during Deployment/cloning
Routing distribution tree
Agents deployment path
Mobile Agents during execution
Monitoring Station

Figure 7-15: Illustration of agent deployment under near-optimal conditions.

7.6.1 Deployment Traffic in Agents Incapable of Cloning

We have already proved with theorem 1 that a number of agent equal to $p=d(u)*n^{L-1}$ will be generated at the monitoring station and subsequently sent at level L of the spanning tree (Figure 7-15a). Each agent will traverse L levels. Assuming that $MA_{size,NC}$ is the agent size and that the whole deployment process takes a unit of time, the total deployment traffic expressed in bit per second will be:

$$\boxed{DEPL_{traff,MA,NC} = \frac{8 * MA_{size,NC} * p * L}{1\text{sec}} = 8 * MA_{size,NC} * L * d(u) * n^{L-1}}$$

which scales as

$$DEPL_{traff,MA,NC} \propto O\left(L * n^{L-1}\right) \propto O(p \log p)$$

7.6.2 Deployment Time in Agents Incapable of Cloning

The total agent deployment time can be calculated by considering the time to deploy $(p/d(u))=n^{L-1}$ agents in one sub-tree, T_r. In fact, agents targeting different sub-trees will not concur over the same links and will be deployed in parallel.

The total deployment time is given by the units of time required to transmit (n^{L-1}) agents between levels 0-1, plus the units of time to transmit (n^{L-1})/n agents between levels 1-2, and so on. The number of agents transmitted between levels l-(l+1) will be (n^{L-1})/(n^{l}). Hence the total number of time units will be $\sum_{i=1}^{L} n^{L-i}$ and total deployment time will be:

$$DEPL_{time,MA,NC} = \left(TRANSM_{time} + FORW_{time}\right) * \sum_{i=1}^{L} n^{L-i}$$

$$\sum_{i=1}^{L} n^{L-i} = \sum_{j=0}^{L-1} n^{L-j-1} = n^{L-1} * \sum_{j=0}^{L-1} \left(\frac{1}{n}\right)^{j} = \frac{n^{L}-1}{n-1}$$

Hence:

$$DEPL_{time,MA,NC} = \left(TRANSM_{time} + FORW_{time}\right) * \frac{n^{L}-1}{n-1}$$

which scales as

$$DEPL_{time,MA,NC} \propto O\left(n^{L-1}\right) \propto O(p)$$

7.6.3 Deployment Traffic in Agents Capable of Cloning

Cloning will be performed at each step, as illustrated in Figure 7-15b. In each sub-tree T_r there will be n^{l} agents simultaneously traversing different links between levels l-(l+1), with $l < L$. Thus, the total traffic will be

$$DEPL_{traff,MA,Cl} = \begin{cases} \dfrac{8 * MA_{size,Cl} * d(u)}{1\sec} * \sum_{l=0}^{L-1} n^{l} = 8 * MA_{size,Cl} * d(u) * \dfrac{n^{L}-1}{n-1} & L > 0 \\ 0 & L = 0 \end{cases}$$

whereby $MA_{size,Cl}$ is the size in bytes of an agent capable of cloning.

The above expression operates at:

$$DEPL_{traff,MA,Cl} \propto O\left(n^{L-1}\right) \propto O(p)$$

7.6.4 Deployment Time in Agents Capable of Cloning

In this case there will never be agents concurring for the same link, since agents are cloned to follow mutually exclusive paths. Thus, the deployment time can be derived directly from the upper bound found in Chapter 6 (Section 6.2.2, page 132), with the following substitutions:

$\delta_{max} = n$; $R(u) = L$;

Hence,

$$DEPL_{time,MA,Cl} = \{DESERIL_{time} + c * O(n) + n * [CLON_{time} + SERIAL_{time}] + TRANSM_{time} + FORW_{time}\} * L$$

In practice $c*O(n)$ is negligible since c is several order of magnitude smaller that the other terms. Therefore,

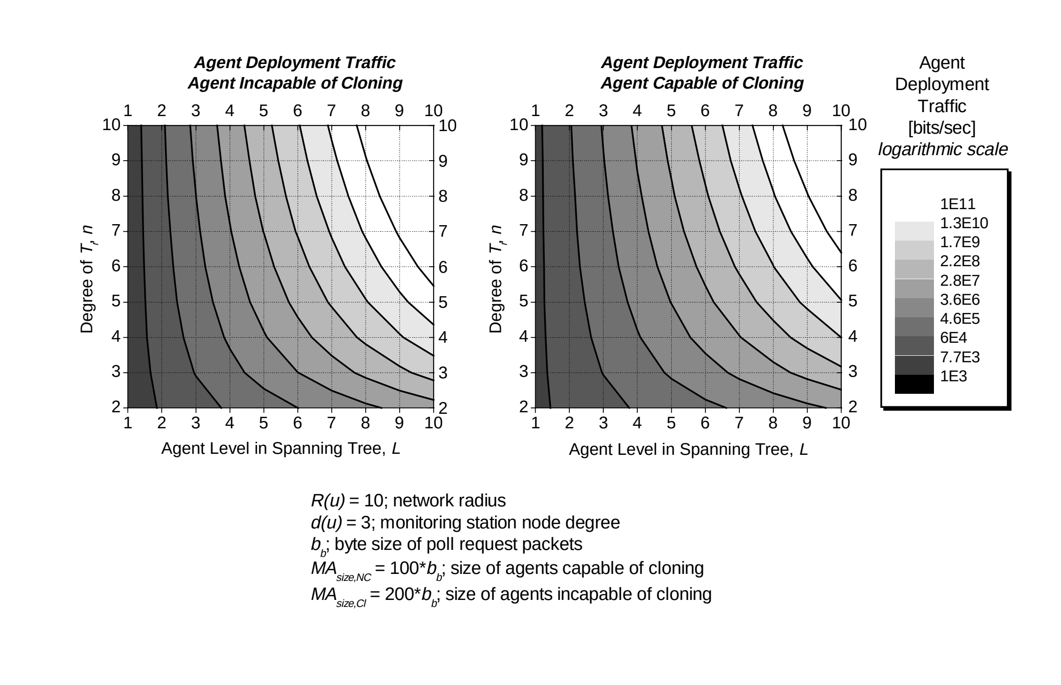
$$DEPL_{time,MA,Cl} \propto O(L) \propto O(p)$$

7.7 Comparative Transient Analysis

In this section we use the mathematical models presented in the previous sections to study two main forms of overhead introduced by the agent solutions, namely agent deployment traffic and agent deployment time. We study the impact that agent cloning may have on such overheads.

7.7.1 Analysis of Agent Deployment Traffic

The deployment traffic models presented above are depicted in Figure 7-16 in the form of contour plots. Similarly to the contour plots discussed earlier, ranges of traffic values are denoted on an XY grid using a greyscale map. Traffic ranges are expressed in bit per second and are reported on a logarithmic scale. The *Y*-axis still reports increasing values of *n*, whereas the *X*-axis reports increasing values of *L*.



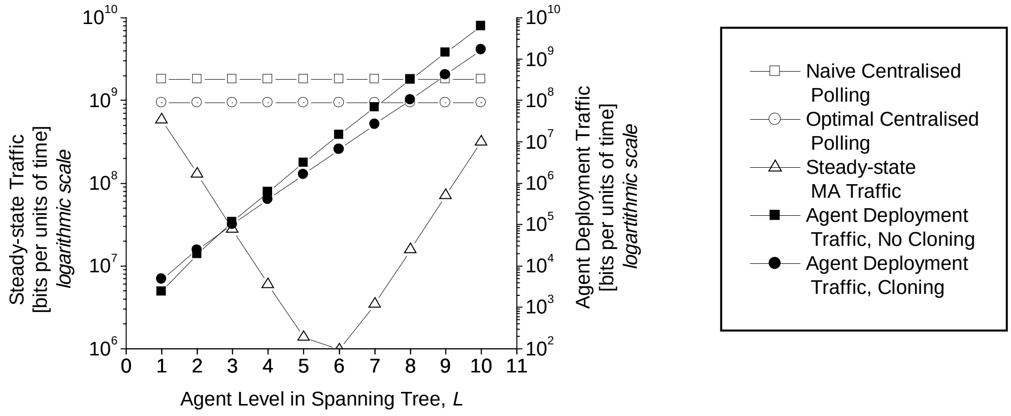
R(u) = 10; network radius
d(u) = 3; monitoring station node degree
b_b; byte size of poll request packets
$MA_{size,NC}$ = 100*b_b; size of agents capable of cloning
$MA_{size,Cl}$ = 200*b_b; size of agents incapable of cloning

Figure 7-16: Contour plots depicting deployment traffic for the case of agent capable and incapable of cloning, respectively.

The fact that no significant difference appears between the two agent solutions – *i.e.*, agent capable and incapable of cloning, respectively – should not be misinterpreted. This is due to the fact the those graphs focus on relatively small values of *L* and *n* for which the difference in traffic is not remarkable. This is true more generally for small-scale agent configurations, that is when the involved number of agents to be deployed is relatively small. Under these circumstances the advantage of minimising the number of agents by means of cloning techniques tends to be outweighed by the fact that agents capable of cloning tend to be larger in size, since they need to incorporate more information and additional logic to handle the cloning process.

Agents capable of cloning tend to be more effective at larger scales, as can be observed from the traffic models. Agents incapable of cloning operate at O($L*n^{L-1}$), whereas agents capable of cloning operate only at O(n^{L-1}).

Figure 7-17 combines in the same plot both steady-state monitoring traffic and agent deployment traffic for a better comparison. Two different *Y*-axes are used for monitoring traffic and agent deployment traffic, respectively. Results are reported on logarithmic scales, which use different scale factors and ranges. Traffic values are plotted against *L* to show the impact of the number of agents on overheads.



R(u) = 10; network radius
d(u) = 3; monitoring station node degree
n = 4; degree of sub-trees T_r of spanning tree
P_r = 4 packets/sec; polling rate
Not_r = 1 packet/sec; agent notification rate
b_b; byte size of poll request packets
$b_r = b_b$; byte size of poll response packets
$b_{not} = b_b$; byte size of notification packets
$MA_{size,NC} = 100*b_b$; size of agents capable of cloning
$MA_{size,Cl} = 200*b_b$; size of agents incapable of cloning

Figure 7-17: Comparison between steady-state monitoring traffic and agent deployment traffic.

L values obviously affect only the agent solutions. That is why the traffic profiles are horizontal lines when centralised polling is adopted. The difference in slope between the two agent deployment lines illustrates a corresponding difference in scalability.

Finally, though the agent deployment traffic is shown to be comparable or even significantly larger than steady-state monitoring traffic it should be noted that in practice the former is a one-off traffic, whereas the latter persists during the whole duration of a monitoring task. Hence, traffic overheads are significant for short-lived monitoring tasks, whilst they become increasingly negligible for long-term tasks.

A simple comparison between the results obtained under near-optimal conditions and the ones achieved in Chapter 6 for the general case is now carried out (Table 7-3). It should be mentioned that the results presented under the “general conditions” column refer to upper bounds on deployment traffic that where calculated considering the worst-case scenario in which agent were deployed at distance *R(u)* from the monitoring station. In this case, in the “near-optimal conditions” column we have that *L=R(u)*.

	Agent Deployment Traffic	
	General Conditions	**Near-Optimal Conditions**
Agent incapable of cloning (centralised agent location algorithm)	*O(p*R(u))*	$O\left(L * n^{L-1}\right) \propto O\left(p \log p\right)$
Agents capable of cloning (distributed agent location algorithm)	*O(p+R(u))*	$O\left(n^{L-1}\right) \propto O\left(p\right)$

Table 7-3: Comparison of deployment traffic between general and near-optimal conditions.

Under near-optimal conditions we have that $n^{L-1} \propto p$. Hence, as far as the centralised agent location algorithm is concerned, the models of agent deployment traffic under general and near-optimal conditions are consistent.

The same conclusion can be drawn for the case of the distributed agent location algorithm. In fact, if $n^{L-1} \propto p$ and $R(u) = L$ we have that $O(p+R(u)) \propto O(n^{L-1}+L) \propto O(n^{L-1})$.

7.7.2 Analysis of Agent Deployment Time

The agent deployment time models presented above are depicted in Figure 7-18 in the form of contour plots. Similarly to the above contour plots, ranges of deployment time values are denoted on a *XY* grid using a greyscale map. Times are expressed in units of time and are reported on a logarithmic scale. The *Y*-axis still reports increasing values of *n*, whereas the *X*-axis reports increasing values of *L*.

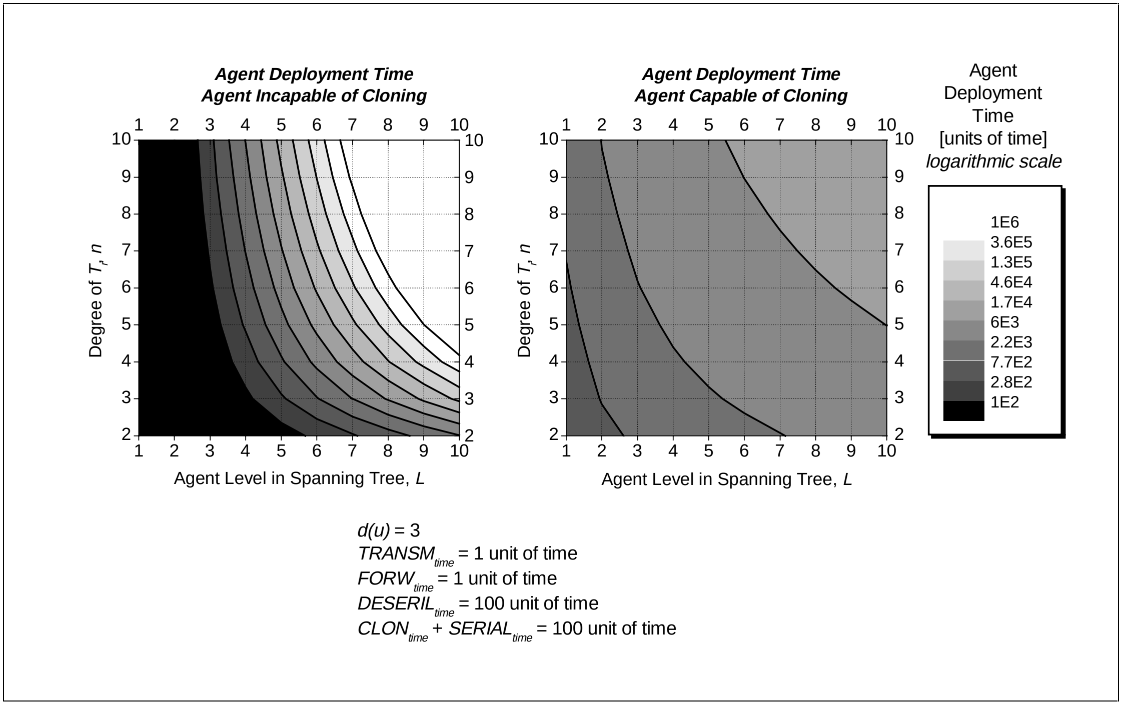
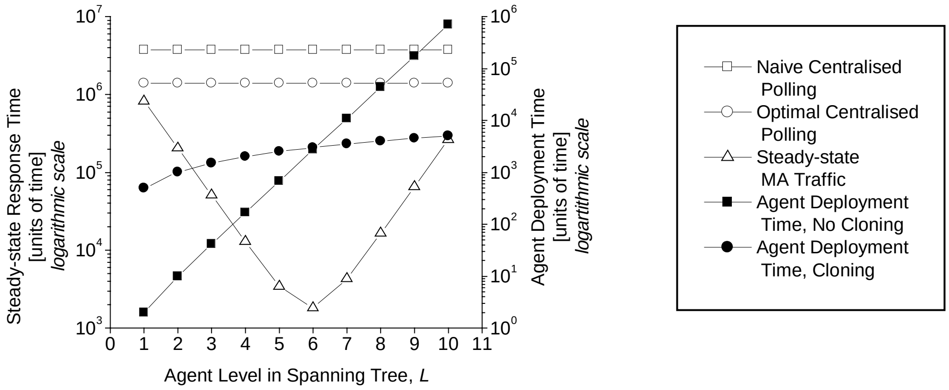


Figure 7-18: Contour plots depicting deployment time for the case of agent capable and incapable of cloning, respectively.

The difference between the two agent solutions is more remarkable in the case of deployment time than in the one of traffic. This is even more evident from Figure 7-19, which combines both steady-state monitoring response time and agent deployment time. Two different *Y*-axis are used for response times and deployment times respectively. Results are reported on logarithmic scales, which use different scale factors and ranges.



$R(u)$ = 10; network radius
$d(u)$ = 3; monitoring station node degree
n = 4; degree of sub-trees T_r of spanning tree
P_r = 4 packets/sec; polling rate
Not_r = 1 packet/sec; agent notification rate
b_b; byte size of poll request packets
$b_r = b_b$; byte size of poll response packets
$b_{not} = b_b$; byte size of notification packets
$MA_{size,NC} = 100*b_b$; size of agents capable of cloning
$MA_{size,Cl} = 200*b_b$; size of agents incapable of cloning

Figure 7-19: Comparison between steady-state monitoring response time and agent deployment time.

Figure 7-19 is qualitatively very similar to Figure 7-17, except for the larger difference in slope between the two agent solutions. This means that agent deployment time is more sensitive to the type of agent solution than agent deployment traffic. To understand this lets us refer back to the models presented above. Agents capable of cloning are deployed less efficiently at low scales because in this case cloning and serialisation time predominate over the other terms. On the other hand this approach leads to significant reductions in deployment time at larger scales. In fact, agent systems capable of cloning operate at O(log(p)), whereas when agent incapable of cloning are adopted the system operates at O(p).

A simple comparison between the results obtained under near-optimal conditions and the ones achieved in Chapter 6 for the general case can be carried out by looking at Table 7-4. The results presented in the first row for the case of agents incapable of cloning seem inconsistent with each other. This is due to the fact that in the case of agents incapable of cloning an upper bound was given in Chapter 6, whereas a precise evaluation is calculated for the near-optimal case. Since

$n^{L-1} \propto p$ the near-optimal case operates at $O(p)$ whereas the general case operates at most at $O(p+N^2)$.

	Agent Deployment Time	
	General	**near-Opt.**
Agent incapable of cloning (centralised agent location algorithm)	$O(p+N^2)$	$O(n^{L-1}) \propto \mathrm{O}(p)$
Agents capable of cloning (distributed agent location algorithm)	$O(R(u))$	$O(L) \propto \mathrm{O}(\log(p))$

Table 7-4. Comparison of deployment time between general and near-optimal conditions.

In the case of agents capable of cloning the results presented in the second row are equivalent if we notice that $L=R(u)$.

7.8 Discussion and Conclusions

This chapter assesses agent-based distributed monitoring under near-optimal conditions. After finding these conditions, models aimed at studying centralised and distributed monitoring at steady state are presented. Agent-based distributed polling is compared and contrasted with two different centralised approaches, the naïve approach and an optimised version of it proposed by Rescigno [Rescigno 97].

Table 7-5 summarises the resulting steady-state models. The advantages of agent-based distributed monitoring in terms of performance and scalability are quantitatively evident.

	Steady-state Traffic **(under near-optimal conditions)**
Naïve centralised polling	$O\left(P_r * R(u) * n^{R(u)}\right) \propto O\left(P_r * R(u) * N\right)$
Optimal centralised polling	$O\left(P_r * R(u) * n^{R(u)}\right) \propto O\left(P_r * R(u) * N\right)$
Agent-based distributed polling	$O\left(P_r * \left\{\left(R(u) - L\right) * n^{R(u)-L} + L * N^{L-2}\right\}\right) + O\left(Not_r * L * n^{L-1}\right)$ $\propto O\left(P_r, R(u), N, p \log p\right)$
	Steady-state Response Time **(under near-optimal conditions)**
Naïve centralised polling	$O\left(P_r * \left(R(u) + n^{R(u)}\right)\right) \propto O\left(P_r * \left(R(u) + N\right)\right)$
Optimal centralised polling	$O\left(P_r, N, R(u)\right)$
Agent-based distributed polling	$O\left(P_r * \max\left\{n^{R(u)-L-1}; \left(L-1\right)\right\} + Not_r * n^{L-1}\right)$ $\propto O\left(P_r, R(u), N, p\right);$

Table 7-5: Summary of results on steady-state performance under near-optimal conditions.

An interesting conclusion that can be drawn from Figure 7-11 and Figure 7-14 is that both traffic and response time can be minimised not only by acting on agent locations, but also by adopting the ‘reasonable’ number of agents for a given monitoring task. We have found that while a relatively small number of agents does not allow a full exploitation of decentralisation, a relatively large number of agents tends to be associated with a increased information flow between the agents and the monitoring station. Therefore, there is a trade-off decision to be taken which can be based on heuristics, as discussed in the next Chapter for the case of realistic network topologies.

We have discussed various possibilities for agents to deliver pre-processed monitoring information to the monitoring station. Agents may be ‘polled’ semantically compressed information by the station; alternatively periodic or event-driven notifications may be sent by the agents. No matter which modality is used to deliver information to the central station, the amount of traffic exiting the agent can be assumed to be significantly smaller than the traffic incurred to collect the raw information in the first place. Therefore, the agent system brings two advantages in comparison to centralised polling: first, it reduces the data collection traffic by offering decentralised polling stations; second, it operates local data semantic compression at the agent location and, by doing so, it reduces the amount of information which traverses the network.

The main limitation of the agent solutions is represented by the agent deployment overheads, namely agent deployment traffic and agent deployment time (Table 7-6). These are one-off costs paid only initially, *i.e.* at transient time. Once the agents have been deployed the distributed monitoring system is in operation. It has been found that under near-optimal conditions those costs increase linearly. In fact, deployment traffic operates at $O(n^{L-1})$ and, since

$n^{L-1} \propto p$, traffic increases linearly with the number of agents, p. For the same reason, deployment time increases sub-linearly with p.

	Agent Deployment Traffic	**Agent Deployment Time**
Agent incapable of cloning (centralised agent location algorithm)	$O(L*n^{L-1}) \propto O(p*\log(p))$	$O(n^{L-1}) \propto O(p)$
Agents capable of cloning (distributed agent location algorithm)	$O(n^{L-1}) \propto O(p)$	$O(L) \propto O(\log(p))$

Table 7-6: Summary of results on agent deployment traffic and time under near-optimal conditions.

It is essential not to have an excessive number of agents for at least two reasons. First, more agents result in larger deployment overheads. Second steady-state traffic and response time benefit from an increasing number of agents only until a certain point, as shown in Figure 7-11 and Figure 7-14. The agent-to-monitoring station communication load tends to increase with the number of agents, impacting negatively the overall traffic and delay. In fact, the larger the number of agents, the smaller the agent-to-number or nodes ratio will become. In turn, agents will be able to produce a lower level of data aggregation.

Cloning plays a key role in containing the overheads of agent-based monitoring. Hence, it is important to consider its use whenever those overheads are critical, such as in tasks that have a relatively short duration.

A final comment regards the near-optimal conditions that reflect an ideal network topology in which the routing tree, rooted at the monitoring station, is balanced and symmetrical. These conditions are not met in real networks but are representative in the case of hierarchical networks. Hence, the conclusions achieved with this chapter do not have only a theoretical scope. They can also be regarded as a practical reference for the case of hierarchical network topologies.

The case of more general network topologies is discussed in the next Chapter 8, which approaches the assessment of the proposed agent system through simulation.

The present chapter concludes the presentation of the theoretical work carried out in the context of the thesis.

Chapter 8

Simulation Results

While the previous two chapters evaluated the research hypothesis and the proposed agent system in a theoretical fashion, here we carry out a simulation-based analysis of distributed monitoring, at steady-state and under general conditions (Figure 8-1). The purpose is to concentrate on the phenomena which follow the initial agent deployment process and achieve a quantitative evaluation of the performance benefits of the agent solution in comparison to both centralised and static distributed approaches.

This analysis has been carried out by simulation and is presented herein through the most significant simulation results. The Chapter is divided in three parts. The first four sections (8.1 through 8.4) cover a comparative performance and scalability study between centralised and agent-based distributed monitoring. The adopted scalability indicators are: polling rate; number of monitored objects; network diameter; and number of MAs. Sensitivity to number of agents is an important design parameter which is discussed in Section 8.4.

The second part (Section 8.5) aims at assessing the goodness of the agent location algorithm by evaluating its distance from optimality. This is done by comparing the overall performance achieved by the proposed agent location algorithm against the hypothetical case in which those locations were computed optimally.

The final part (Section 8.6) presents an initial study on the ability of such an algorithm to provide adaptation to variations in the network status. The idea is that it is possible to make use of the same logic used during agent deployment for providing adaptation through agent migration at execution time – *i.e.*, during the lifespan of the monitoring activity.

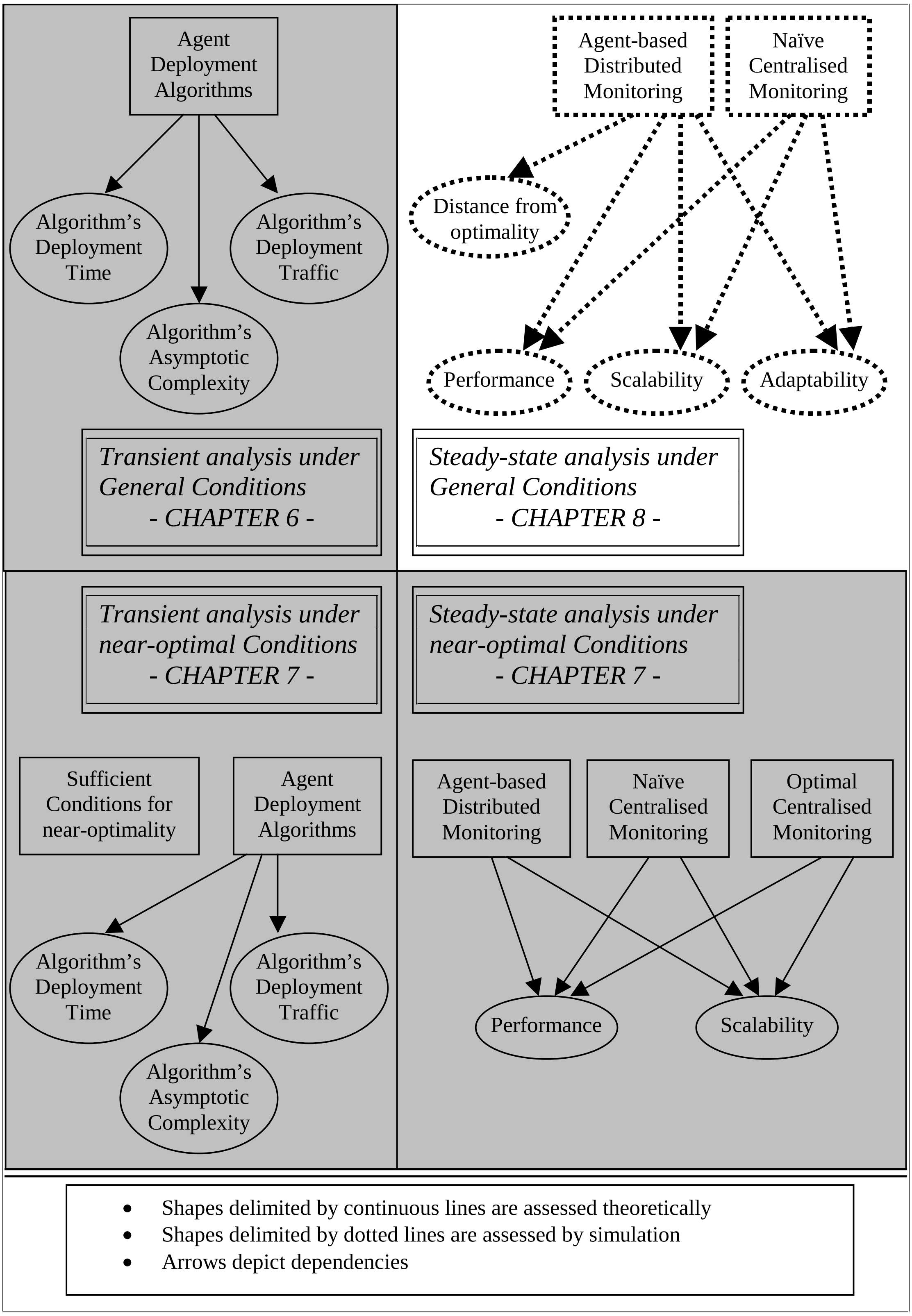

Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Transient analysis under General Conditions - CHAPTER 6 -
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Distance from optimality
Performance
Scalability
Adaptability
Steady-state analysis under General Conditions - CHAPTER 8 -
Transient analysis under near-optimal Conditions - CHAPTER 7 -
Steady-state analysis under near-optimal Conditions - CHAPTER 7 -
Sufficient Conditions for near-optimality
Agent Deployment Algorithms
Algorithm's Deployment Time
Algorithm's Deployment Traffic
Algorithm's Asymptotic Complexity
Agent-based Distributed Monitoring
Naïve Centralised Monitoring
Optimal Centralised Monitoring
Performance
Scalability
Shapes delimited by continuous lines are assessed theoretically
Shapes delimited by dotted lines are assessed by simulation
Arrows depict dependencies

Figure 8-1. Schematic representation of the focus of Chapter 8.

Simulations have been repeated several times to guarantee statistical accuracy. Sections are accompanied by two types of diagram. Statistical box diagrams are used to discuss the validity of the various statistical indicators, such as mean, median, minimum, and maximum values, and to report the main simulation parameters. Best-fit functions are then used to draw conclusions on simulation results. In the remaining part of this chapter, the focus is on the discussion of those results. The actual simulation methodology, validation, and statistical data analysis are described in more detail in Chapter 5 (Section 5.3).

The simulation work is complementary to the theoretical one (conducted in chapter 7) since the network model and class of network topology adopted in the two approaches are different. The models developed by these two approaches are thus not directly comparable. However, their results are consistent, as discussed in the present chapter.

An initial development of the simulation results of this chapter has been published in [Liotta 99c, Liotta 01a, Liotta 01c] and submitted for publication to the IEEE Network Magazine [Liotta 01b].

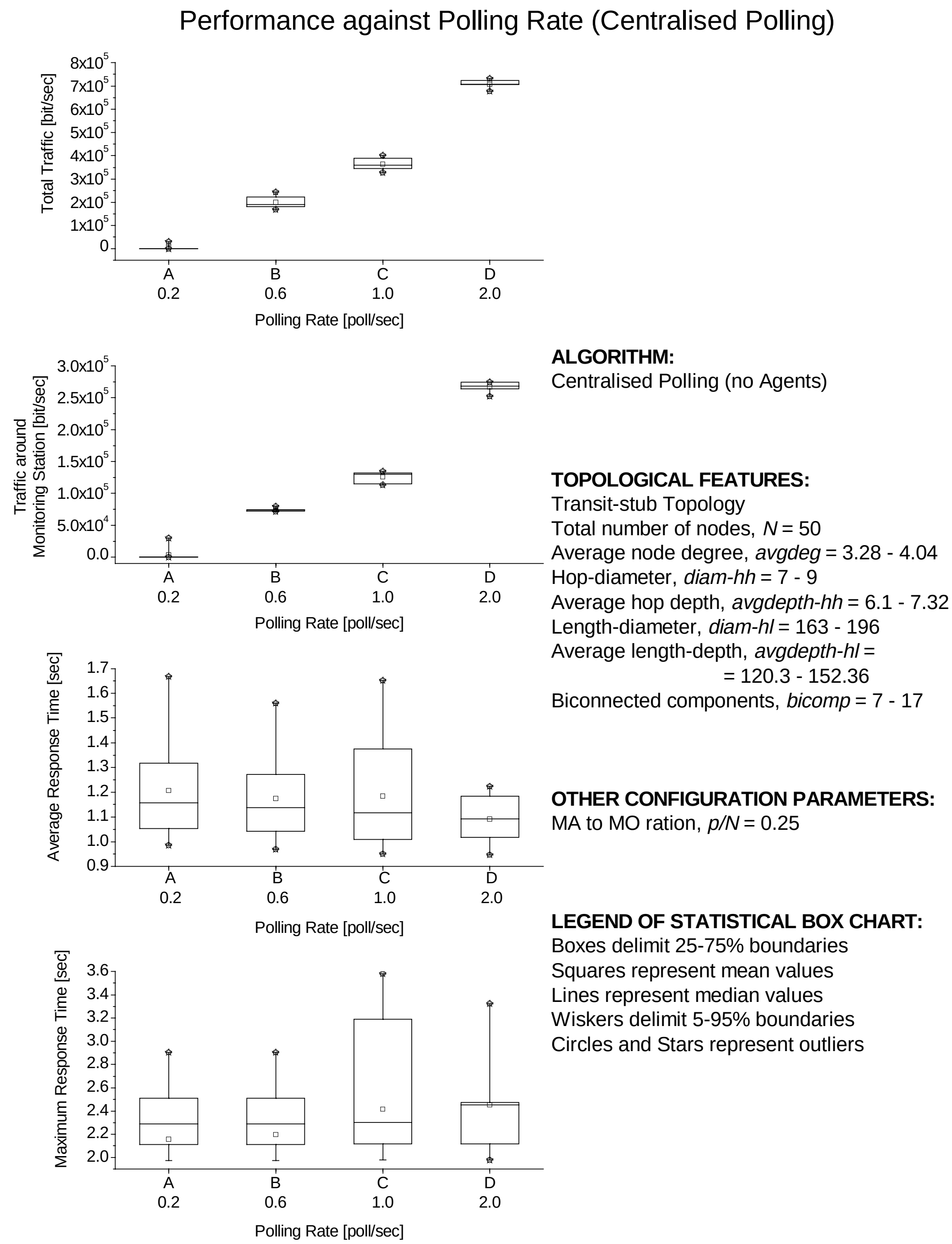
8.1 Performance against Polling Rate

This section intends to capture the differences between centralised and agent-based monitoring when polling rate is adopted as scalability indicator. As for the remaining part of the chapter, the performance indicators are: 1) the total traffic incurred by the monitoring system; 2) the monitoring traffic incurred in the proximity of the monitoring station; 3) the average response time associated with the monitoring system; and 4) the maximum response time of the monitoring system. The motivations for choosing the above indicators have been already discussed in Chapter 5.

Figure 8-2 depicts the impact of polling rate, P_r on performance for the case of centralised polling. Statistical box diagrams have been used because they are well suited to reporting data generated over different runs of the simulations. They summarise the spread of data in a simple diagram that portrays mean, median, first and third quartiles, and data range ([Lewis 99] pp.117-118). Each Y column is represented as a separate box. The Y axes report statistical indicators rather than the actual measured values, which correspond to the configuration parameter reported in the X axis (P_r is reported on a non-linear scale). Boxes are determined by the 25th and 75th percentiles (first and third quartiles respectively); whiskers are determined by the 5th and 95th percentiles; the little squares represent mean values; horizontal lines represent median values; circles depict lowest and highest values; and stars denote the 1-99% range of values.

An important difference between the traffic diagrams and the response ones can be observed. In the former case, means and median values are statistically significantly different for increasing values of polling rate. The 25-75% boxes are, in fact, not overlapping. This is not the case with the latter case. One may think that simulations where not repeated a sufficient number of times to ensure statistical correctness. However, this is not our case; in fact, despite increasing those repetitions, boxes still overlapped. The conclusion drawn was that average and maximum response times were not statistically different in the chosen range of polling rate values. This should not induce the conclusion that response time is not affected by the P_r indicator. Response time is just relatively less sensitive to P_r than traffic.

Response time is expected to grow over a wider range of P_r. However, larger values of P_r could not be chosen in order to prevent portions of the network from being saturated. Network saturation resulted in buffer overflows, a condition under which the monitoring system should not operate to avoid loss of monitoring information or network overloading. Hence, a further study of the system for larger values of polling rate was not needed.


Performance against Polling Rate (Centralised Polling)

ALGORITHM:
Centralised Polling (no Agents)

TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 50
Average node degree, *avgdeg* = 3.28 - 4.04
Hop-diameter, *diam-hh* = 7 - 9
Average hop depth, *avgdepth-hh* = 6.1 - 7.32
Length-diameter, *diam-hl* = 163 - 196
Average length-depth, *avgdepth-hl* =
= 120.3 - 152.36
Biconnected components, *bicomp* = 7 - 17

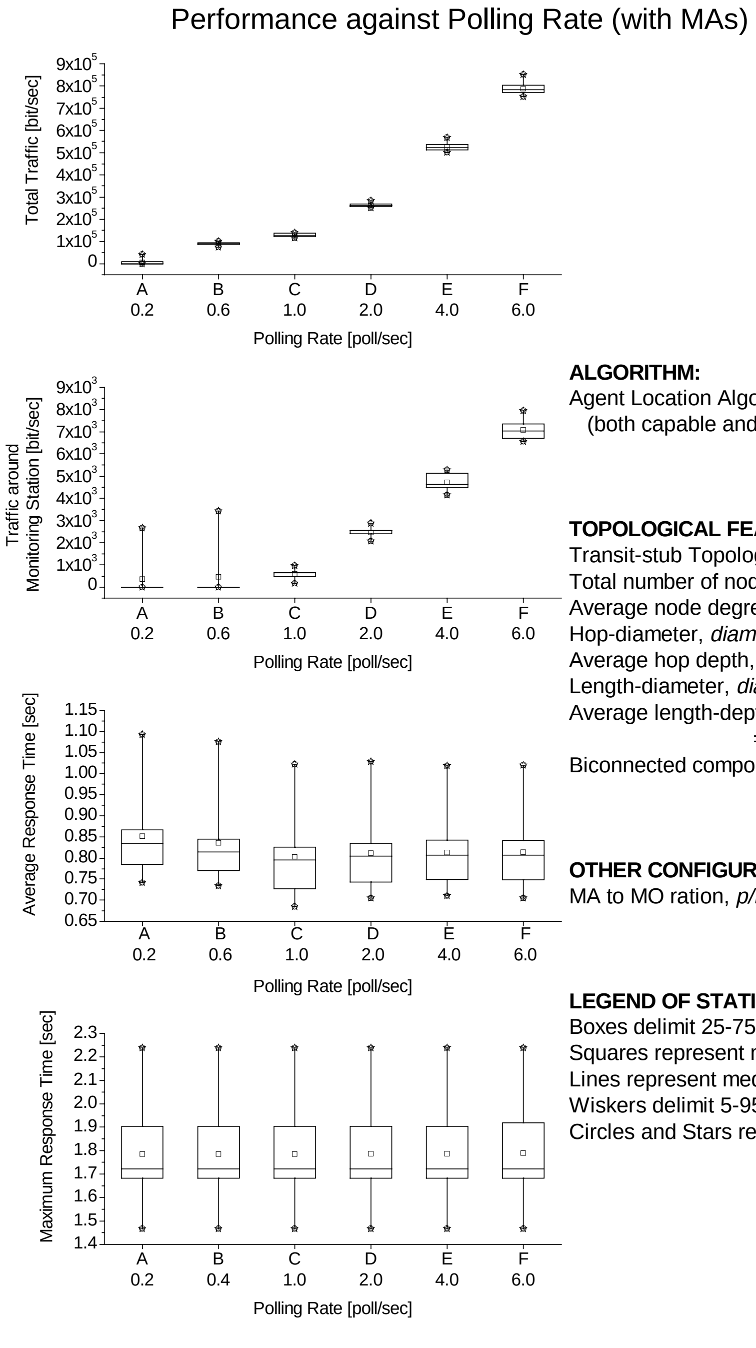
OTHER CONFIGURATION PARAMETERS:
MA to MO ration, *p/N* = 0.25

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-2. Statistical box plots depicting the impact of polling rate on performance in the case of centralised polling.

The significant difference in the spread of values between *traffic* and *response time* diagrams (*i.e.*, difference in height of the statistical boxes), observed in Figure 8-2 is consistently present in all the diagrams presented in the remainder of the present Chapter. This is because response time is affected by the asymmetric nature of the network topology generator largely than traffic. Subsequent simulations are run on topologies which belong to the same family (*e.g.*, they are characterised by comparable values of average node degree, number of nodes, etc.) but exhibit a certain degree of variability on the network diameter. Consequently, the difference between traffic and response time diagrams reflects the fact that the latter is more sensitive to the network diameter than the former. The maximum response time is even more sensitive to network diameter, as results from the statistical box diagrams presented above as well as hereafter.

Similar conclusions can be drawn from Figure 8-3, which depicts the impact of polling rate on performance for the case of agent-based polling. Again, changes in traffic were statistically significant whereas response time was statistically constant.



ALGORITHM:
Agent Location Algorithm
(both capable and incapable of cloning)

TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 50
Average node degree, *avgdeg* = 3.28 - 4.04
Hop-diameter, *diam-hh* = 7 - 9
Average hop depth, *avgdepth-hh* = 6.1 - 7.32
Length-diameter, *diam-hl* = 163 - 196
Average length-depth, *avgdepth-hl* =
= 120.3 - 152.36
Biconnected components, *bicomp* = 7 - 17

OTHER CONFIGURATION PARAMETERS:
MA to MO ration, *p/N* = 0.25

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-3. Statistical box plots depicting the impact of polling rate on performance in the case of agent-based polling.

Figure 8-4 reports the same simulation data in a more abstract way, which allows for further interpretation of the results. Only two indices of central tendency are reported, mean and median, along with their related best-fit functions. In addition, for a better comparison, results obtained for the two cases of centralised and agent-based polling are reported in the same plots.

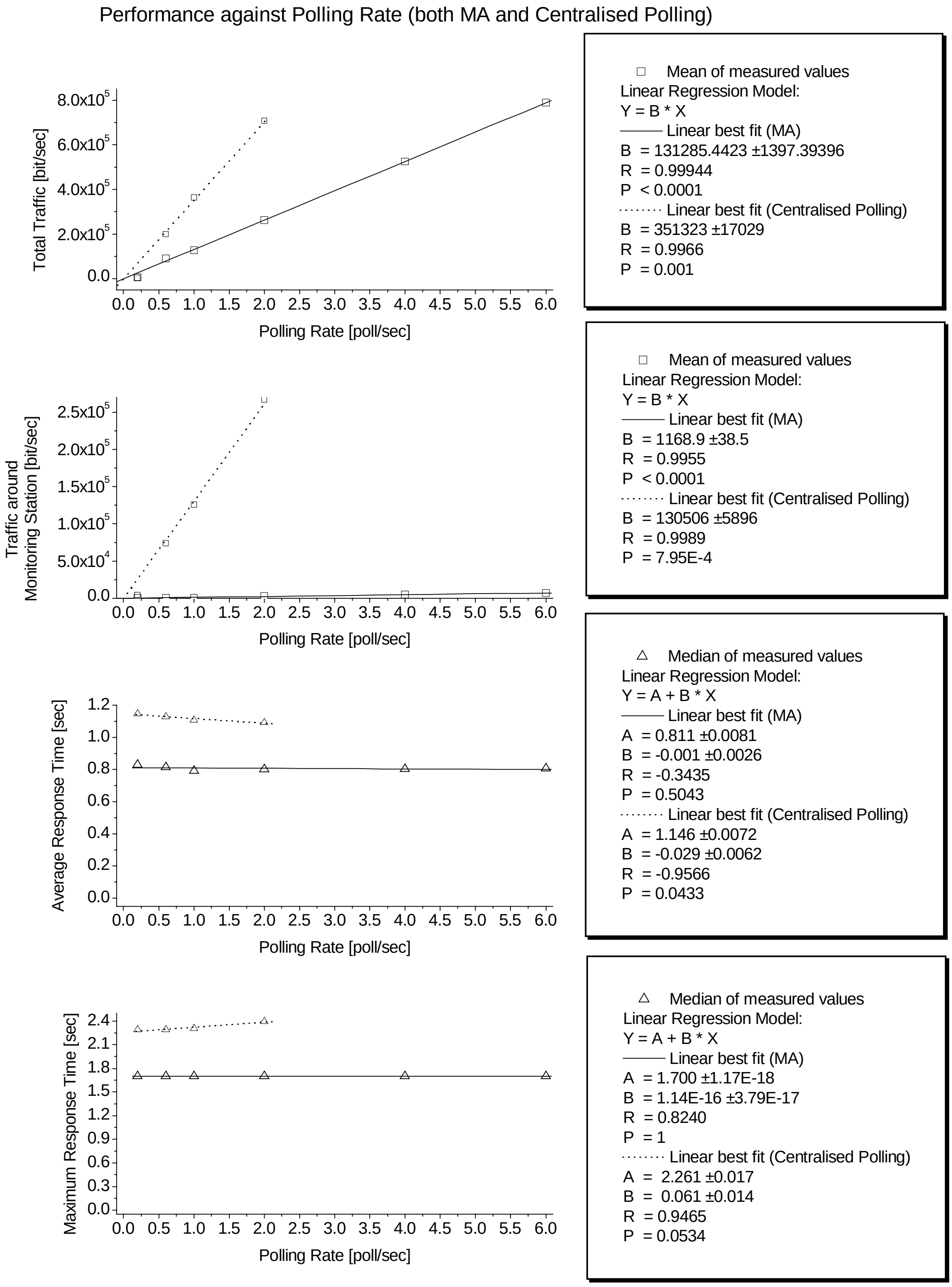


Figure 8-4. Linear best-fit performance functions based on the polling rate scalability indicator.

The reason for using two different indices of central tendency for traffic and response time is that these parameters are characterised by different features. As explained in [Jain 91], the mean is a representative index when total is of interest. Traffic is computed as weighted sum of packet rates, and it is therefore well represented by the mean index. In the case of response time, total is still of interest but the distribution of the various measurements tends to be skewed due to the presence of spurious outliers. That is why the median has been adopted, since this is more resistant to several outlying values.

Linear best fit functions have been computed with the Microcal Origin statistical tool, using the linear regression technique. In Figure 8-4, *A* is the *regression constant i.e.* the intercept value of the linear regression line on the Y axis. *B* is the *regression coefficient i.e.* the slope value of the regression line; *R* is the *correlation coefficient*; and *P* represents the probability that *R* is zero. Values following the “±” symbol are standard error ranges.

It is worth recalling from statistics theory that the regression line represents a perfect fit with the data if R^2=1. Conversely, if the data points are not linearly correlated, *R* will tend to zero ([Lewis 99] Chapter 14 and 15). We can then conclude that all of the best-fit lines of Figure 8-4 are very accurate since R^2≅1 and *P*≅0.

Negative values of the correlation coefficient indicate that one of the variables (either *X* or *Y*) tends to decrease as the other one increases. However, for the case of the ‘average response time’ plots, negative values of *R* should be considered as an anomaly introduced by slightly negative slopes, *B* of the linear-fit functions. *B* values are, in fact, so close to zero that can be approximated to zero.

As already mentioned, the linearity exhibited by the four performance parameters versus polling rate was expected, since the monitoring system was assumed to operate seamlessly – *i.e.*, the traffic incurred by the system was negligible with respect to the overall traffic. Non-linearity was observed for values of polling rate larger than the ones depicted in Figure 8-4. However, the non-linear zone was correlated to buffer overflows, which caused packet loss and were due to network bottlenecks. In other words, the monitoring system was not operating properly in the non-linear zone, which has consequently been discarded from the analysis.

All the plots of Figure 8-4 show that a significant performance improvement can be achieved with the agent approach. The improvement was, again, expectable since the agent solution exploits distribution. The simulations, however, allow us to draw quantitative conclusions based on typical configuration parameters – *e.g.*, topology, number of nodes, number of agents, polling rates, etc. In the simulated system, the agent solution injected approximately 37% of the total traffic incurred with centralised polling. Even more significantly, the traffic incurred in the proximity of the polling station was cut down to 0.8% of the traffic generated with centralised polling. This is, again, in line with the natural ability of agent systems to drag load away from a central processing point, which is usually the system bottleneck.

The agent approach is also well suited to reducing response time by placing computational entities near to monitored entities. For the example system configuration, both the average and maximum response time achieved with agents were approximately 70-75% of their centralised counterparts.

From the scalability viewpoint, the following observations can be made:

1. Both approaches scale as O(P_r), whereby P_r is the polling rate: this is in agreement with the fact that the physical phenomenon beyond the two approaches is the same and is linear under typical operation conditions. This is also in agreement with the finding of Chapter 7.

2. Traffic injected by the agent solution grows at a smaller pace since the slope of the best-fit line is smaller. In the example agent configuration referred to herein, the slope of centralised polling is nearly 5 times larger than the one of the agent system. The relation between the slope of the agent system and its configuration parameters is further discussed in Section 8.4.

3. Sensitivity of response time to P_r is very small in comparison to the one of traffic.

4. The agent approach can sustain larger values of P_r. In the example agent configuration, agents sustained values of P_r that were three times larger than those of the centralised approach. This is directly related to the fact that agents incur a smaller amount of traffic and, thus, the non-linear region is reached for larger values of P_r.

5. From the asymptotic behaviour point of view, the results of this chapter are consistent with those of Chapter 7 (both traffic and response time operate at O(P_r) under near-optimal network conditions as well as general ones).

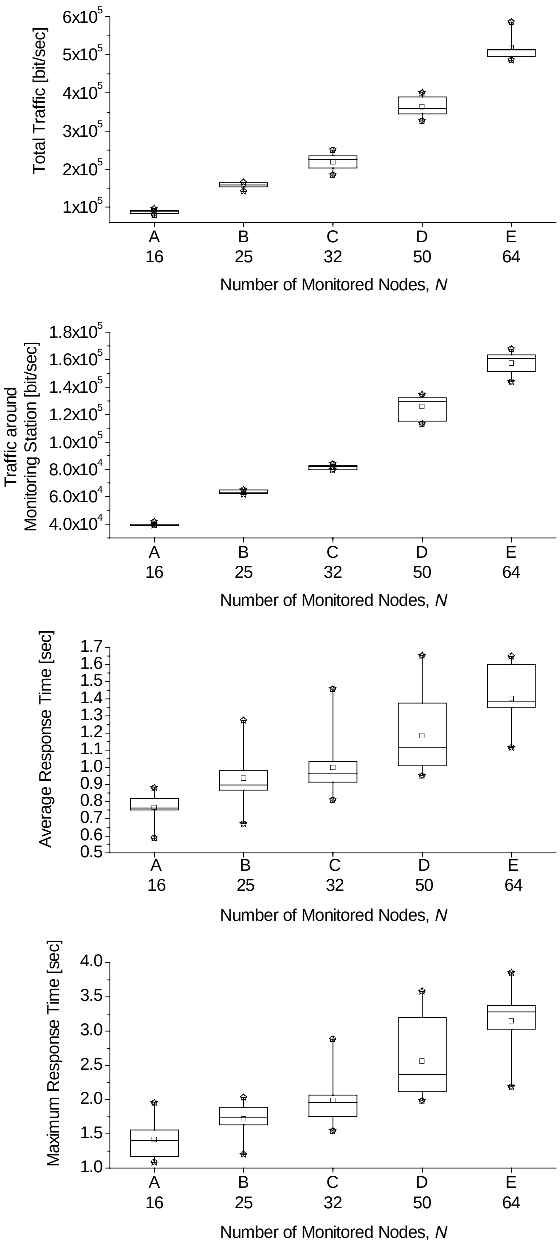
Both traffic and response time are significantly affected by the various agent configuration parameters. Therefore, the above conclusions can be generalised only upon showing that they are valid also when other scalability indicators are adopted. This is done in the following sections.

8.2 Performance against Number of Monitored Objects

This section evaluates the differences between centralised and agent-based monitoring whereby the number of monitored objects, N is adopted as scalability indicator. Plots and conventions are the same as the ones introduced in the previous section.

Figure 8-5 depicts the impact that the number of monitored objects has on the various performance parameters in the case of centralised polling. It can be observed that the central index indicators exhibit statistically different values in the case of traffic since the boxes are not overlapping. A similar conclusion can be drawn in the case of response time, though small overlaps are present in the following ranges: *N*=25-32; *N*=32-50. These are caused by the fact that these ranges are relatively small. However, all in all, an increase in response time is observed when *N* is increased; it is, then, possible to apply regression techniques to create best-fit functions.

Performance against Number of Monitored Nodes - (Centralised Polling)



ALGORITHM:
Centralised Polling (no Agents)

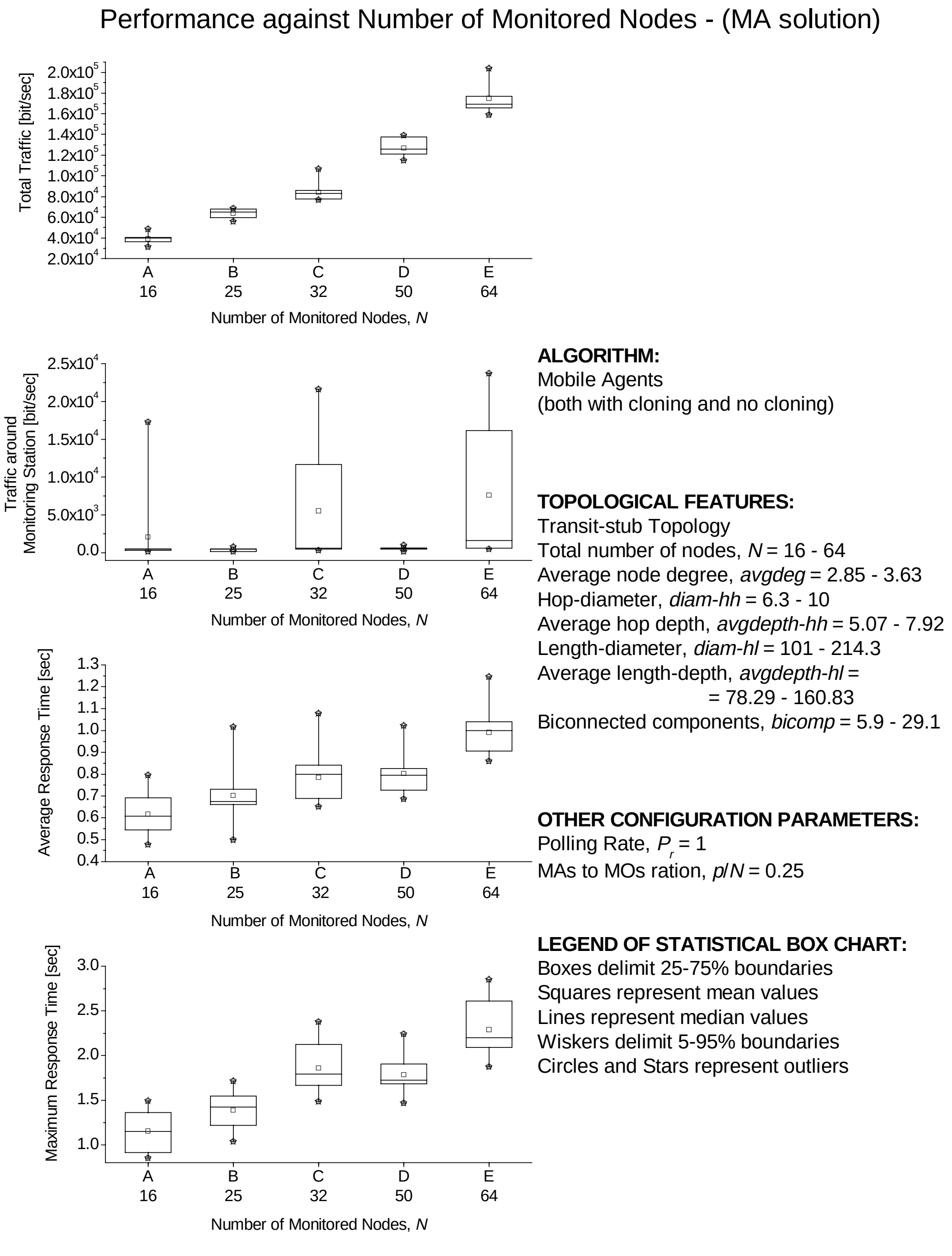
TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 16 - 64
Average node degree, *avgdeg* = 2.85 - 3.63
Hop-diameter, *diam-hh* = 6.3 - 10
Average hop depth, *avgdepth-hh* = 5.07 - 7.92
Length-diameter, *diam-hl* = 101 - 214.3
Average length-depth, *avgdepth-hl* =
= 78.29 - 160.83
Biconnected components, *bicomp* = 5.9 - 29.1

OTHER CONFIGURATION PARAMETERS:
Polling Rate, P_r = 1
MAs to MOs ration, *p*/*N* = 0

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-5. Statistical box plots depicting the impact of number of monitored objects on performance in the case of centralised polling.

Figure 8-6 depicts the results obtained with the agent-based solution. Again, the overlaps are caused by the inability to distinguish between values that are relatively very close. In particular the traffic measured in the vicinity of the polling station was affected very marginally by *N*. In fact, the agent collection traffic – *i.e.*, incurred by the polling request-response process – does not affect this area; that is why the traffic measured around the station is virtually constant with *N*.



ALGORITHM:
Mobile Agents
(both with cloning and no cloning)

TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 16 - 64
Average node degree, *avgdeg* = 2.85 - 3.63
Hop-diameter, *diam-hh* = 6.3 - 10
Average hop depth, *avgdepth-hh* = 5.07 - 7.92
Length-diameter, *diam-hl* = 101 - 214.3
Average length-depth, *avgdepth-hl* =
= 78.29 - 160.83
Biconnected components, *bicomp* = 5.9 - 29.1

OTHER CONFIGURATION PARAMETERS:
Polling Rate, P_r = 1
MAs to MOs ration, *p*/*N* = 0.25

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-6. Statistical box plots depicting the impact of number of monitored objects on performance in the case of agent-based polling.

Linear best-fit functions relative to the above two diagrams are reported in Figure 8-7. The only difference with the diagrams of Figure 8-4 is that here the index of central tendency adopted for traffic around the monitoring station is the median instead of the mean. This has been done to filter out the dramatic effects introduced by large outliers. It can be noted that traffic diagrams are analogous to those of Figure 8-4. The total traffic incurred by the agent solution is in the order of 30% of its centralised counterpart; and the traffic measured around the monitoring station is in the order of 0.5% of its centralised counterpart. Similarly, with agents response time is reduced to values that are in the order of 50% of their centralised counterpart.

It can be concluded that these results are consistent with the ones of Figure 8-4. We can still observe the linear behaviour; and performance improvements are in the same order of magnitude. A small non-linear region is expected for $N \rightarrow 0$ because both traffic and response time should tend to zero for $N \rightarrow 0$. However, the study of such condition is uninteresting from the scalability point of view.

Similarly to what observed in Section 8.1, the agent system is expected to be able to operate correctly – *i.e.*, without causing saturation of bottleneck links – at larger values of *N* than its centralised counterpart since the former is characterised by smaller slopes than those of the latter (Figure 8-7).

It may be worth mentioning that the reasons behind the linear behaviour of traffic and response time functions are different. In the case of traffic, the phenomenon is linear as expected because monitoring traffic is directly proportional to the number of objects, since we are keeping the network diameter constant. Response time is linear for a different reason. This is originated by the bottleneck around the monitoring station. Monitoring packets are queued around the station and those queues grow linearly with the number of request packets, which in turn grow linearly with the number of monitored objects.

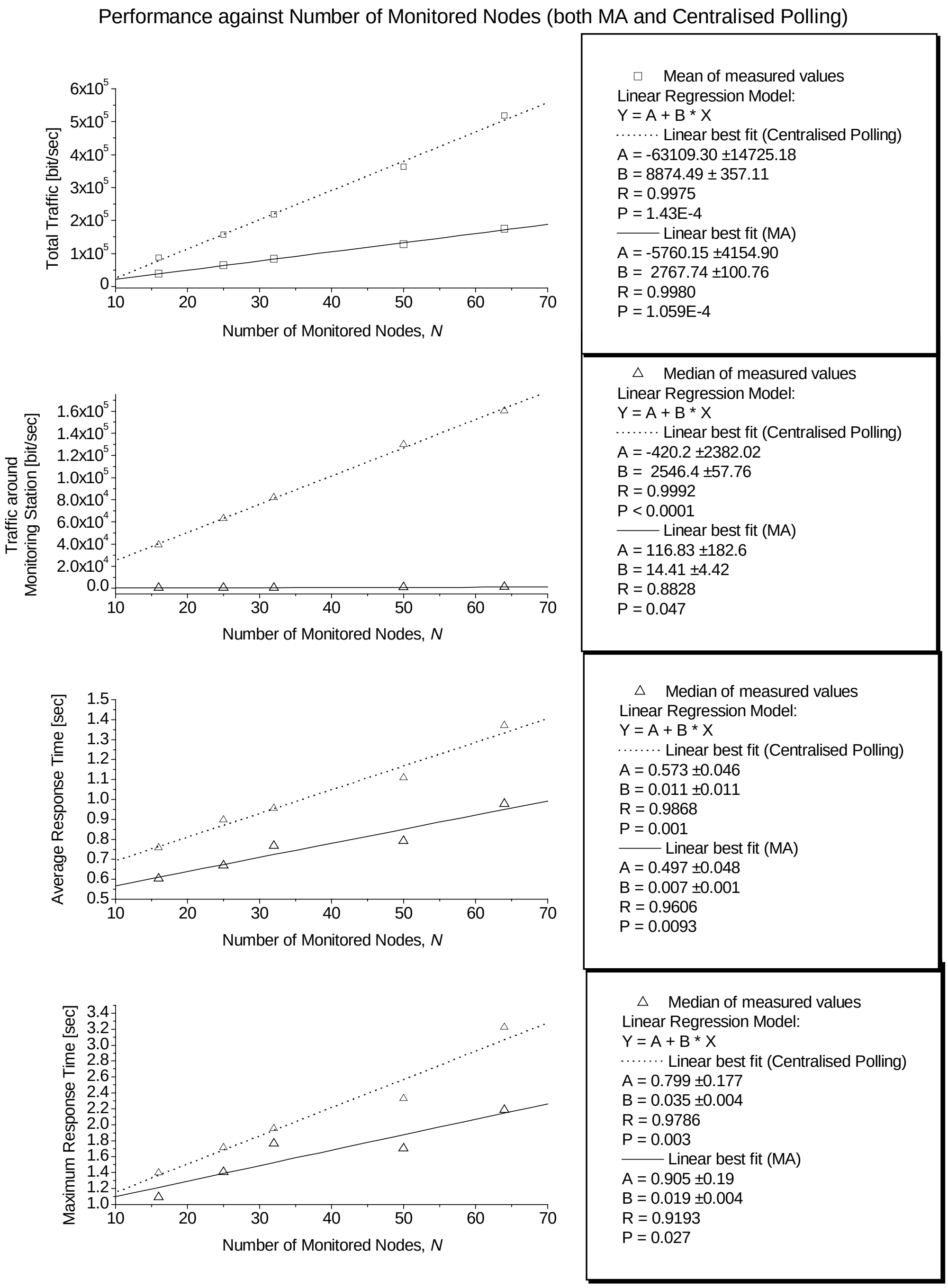


Figure 8-7. Linear best-fit performance functions based on the number of objects scalability indicator.

Finally, if we compare these results with the theoretical ones of Chapter 7, we notice a strong similarity in the behaviour as well as consistency in the asymptotic behaviour. The results summarised in Table 8-1 indicate that if the *N* indicator is individually considered, all performance indicators operate at O(*N*) both under the general conditions considered herein and under the near-optimal conditions of Chapter 7.

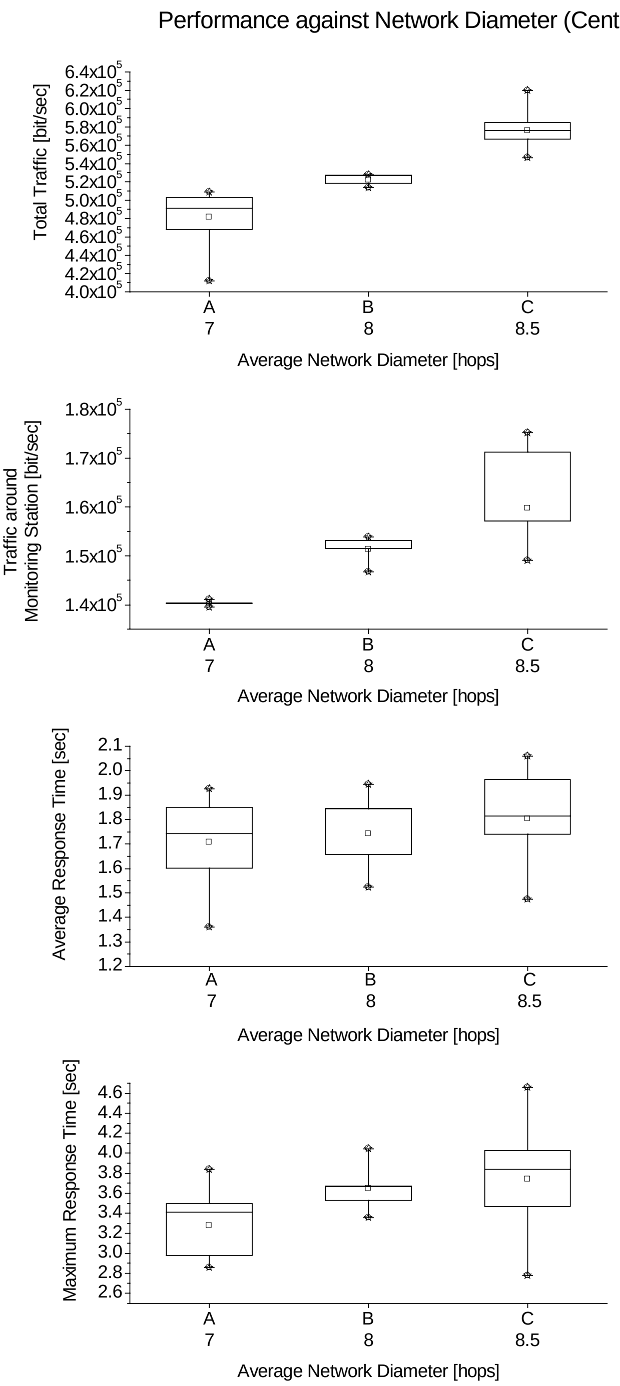
	Steady-state Traffic (under near-optimal conditions)
Naïve centralised polling	$O\left(P_r * R(u) * n^{R(u)}\right) \propto O\left(P_r * R(u) * N\right)$
Optimal centralised polling	$O\left(P_r * R(u) * n^{R(u)}\right) \propto O\left(P_r * R(u) * N\right)$
Agent-based distributed polling	$O\left(P_r * \left\{\left(R(u) - L\right) * n^{R(u)-L} + L * N^{L-2}\right\}\right) + O\left(Not_r * L * n^{L-1}\right)$ $\propto O\left(P_r, R(u), N, p \log p\right)$
	Steady-state Response Time (under near-optimal conditions)
Naïve centralised polling	$O\left(P_r * \left(R(u) + n^{R(u)}\right)\right) \propto O\left(P_r * \left(R(u) + N\right)\right)$
Optimal centralised polling	$O\left(P_r, N, R(u)\right)$
Agent-based distributed polling	$O\left(P_r * \max\left\{n^{R(u)-L-1}; (L-1)\right\} + Not_r * n^{L-1}\right)$ $\propto O\left(P_r, R(u), N, p\right);$

Table 8-1: Summary of results on steady-state performance under near-optimal conditions (from Chapter 7).

8.3 Performance against Network Diameter

This section evaluates the differences between centralised and agent-based monitoring whereby network diameter, *D(u)* is adopted as scalability indicator. Plots and conventions are the same as the ones introduced in the previous sections.

Figure 8-8 depicts the impact that network diameter has on the various performance parameter, in the case of centralised polling. Similarly to the diagrams of the previous sections, this diagram presents a remarkable difference between traffic and response time diagrams. The latter shows wider spread of values which causes partial overlapping in the statistical boxes. The reason is that response time values are much smaller than traffic values, which means that wider ranges of *X* values should be adopted in this case to detect differences among different simulation runs.



ALGORITHMS:
Centralised Naive Polling (no agents)

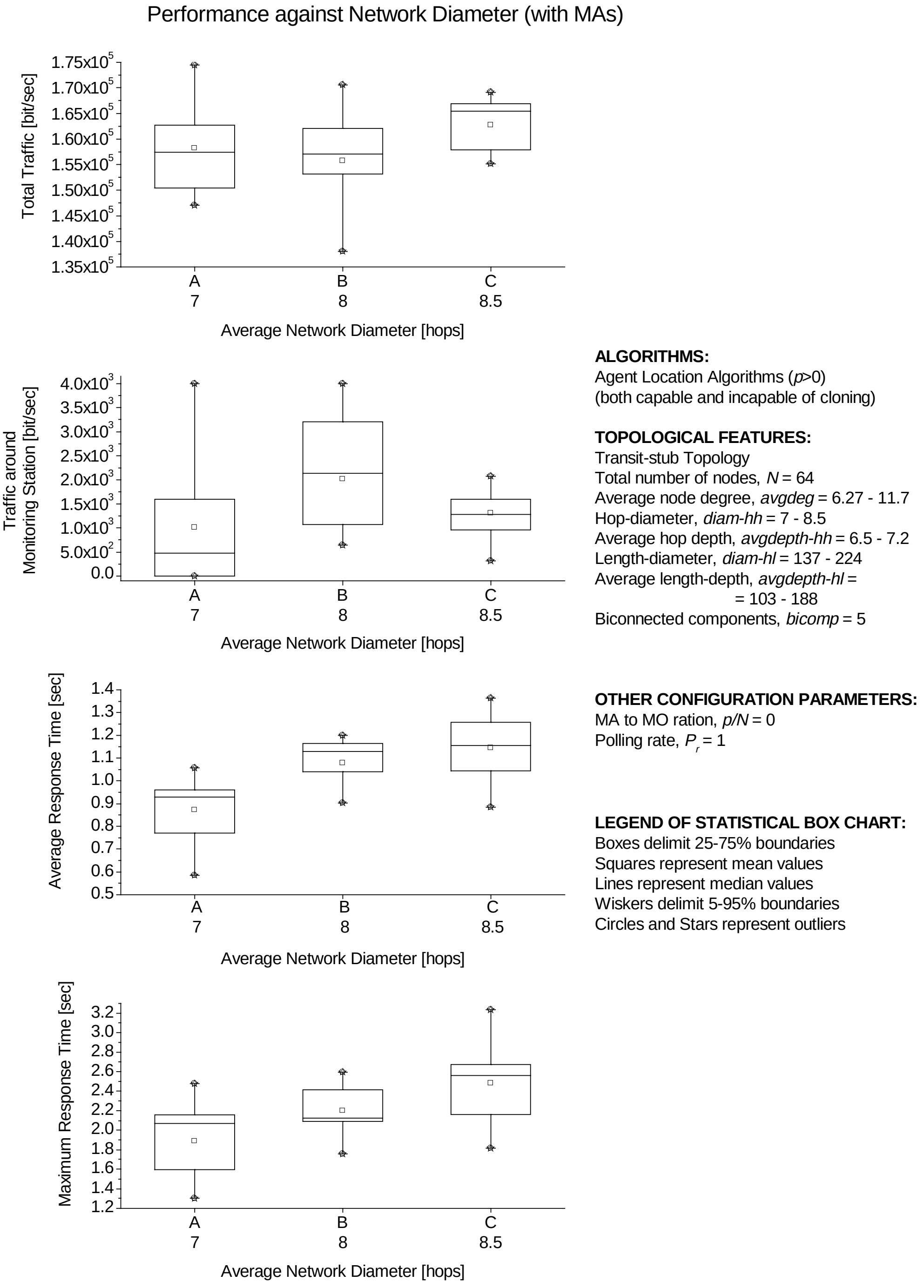
TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 64
Average node degree, *avgdeg* = 6.27 - 11.7
Hop-diameter, *diam-hh* = 7 - 8.5
Average hop depth, *avgdepth-hh* = 6.5 - 7.2
Length-diameter, *diam-hl* = 137 - 224
Average length-depth, *avgdepth-hl* =
= 103 - 188
Biconnected components, *bicomp* = 5

OTHER CONFIGURATION PARAMETERS:
MA to MO ration, *p/N* = 0
Polling rate, P_r = 1

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-8. Statistical box plots depicting the impact of network diameter on performance in the case of centralised polling.

The wide spread of values is even more evident in Figure 8-9. To achieve differences which are statistically more significant, three approaches have been studied. First, increase the range of *D(u)*; second, increase the number of simulation runs; and finally, keep topological parameters as constant as possible when increasing *D(u)*.



ALGORITHMS:
Agent Location Algorithms (*p*>0)
(both capable and incapable of cloning)

TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 64
Average node degree, *avgdeg* = 6.27 - 11.7
Hop-diameter, *diam-hh* = 7 - 8.5
Average hop depth, *avgdepth-hh* = 6.5 - 7.2
Length-diameter, *diam-hl* = 137 - 224
Average length-depth, *avgdepth-hl* =
= 103 - 188
Biconnected components, *bicomp* = 5

OTHER CONFIGURATION PARAMETERS:
MA to MO ration, *p/N* = 0
Polling rate, P_r = 1

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-9. Statistical box plots depicting the impact of network diameter on performance in the case of agent-based polling.

The viability of the first and second approach was limited by intrinsic limitations of the simulator and by simulation times. In fact, the number of nodes increases dramatically with the diameter.

The obstacle to the third approach was the difficulty in minimising the variation in parameters such as the average node degree while increasing network diameter. Wide variations in average node degree (6.27-11.7) were observed in face of small variations in network diameter (7-8.5 or 137-224). It was then concluded that further increases in the range of *D(u)* would have caused more problems than actual benefits.

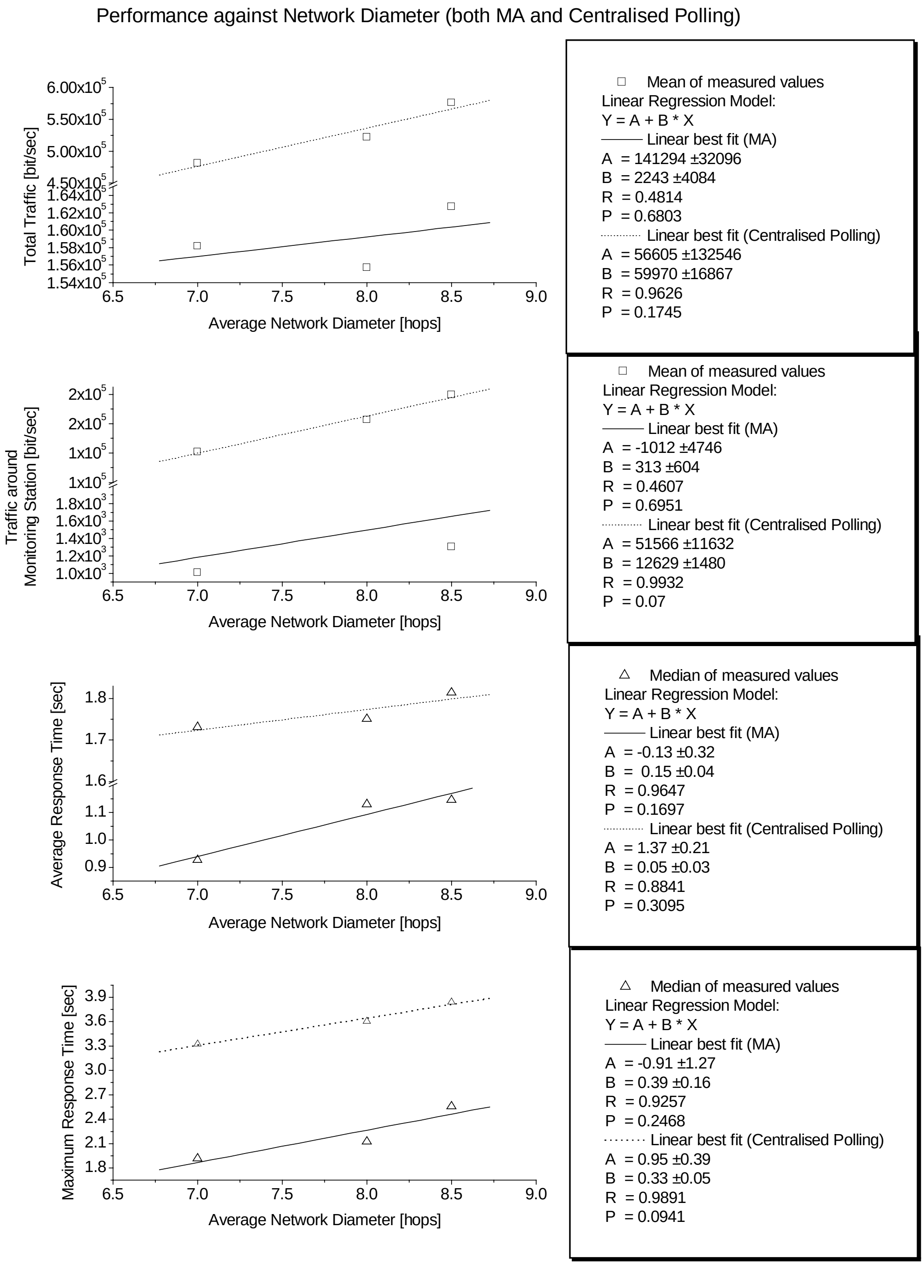


Figure 8-10. Linear best-fit performance functions based on the network diameter scalability indicator.

Despite the above limitation, linear regression was applied to estimate scalability as a function of network diameter (Figure 8-10). The problems identified above are also reflected by the relatively low values of *R* and high values of *P*. Despite that, the results are consistent with the ones of Figure 8-4 and Figure 8-7 and the behaviour is linear with *D(u)*.

Finally, if we compare these results with the theoretical ones of Chapter 7, we notice a strong similarity in the behaviour as well as consistency in the asymptotic behaviour. We recall that *R(u)* is the network radius whereas *D(u)* is its diameter, two indicators which are directly proportional to each other. The results summarised in Table 8-1 indicate that if the *R(u)* indicator is individually considered, all performance indicators operate at O(*R(u)*) both under the general conditions considered herein and under the near-optimal conditions of Chapter 7.

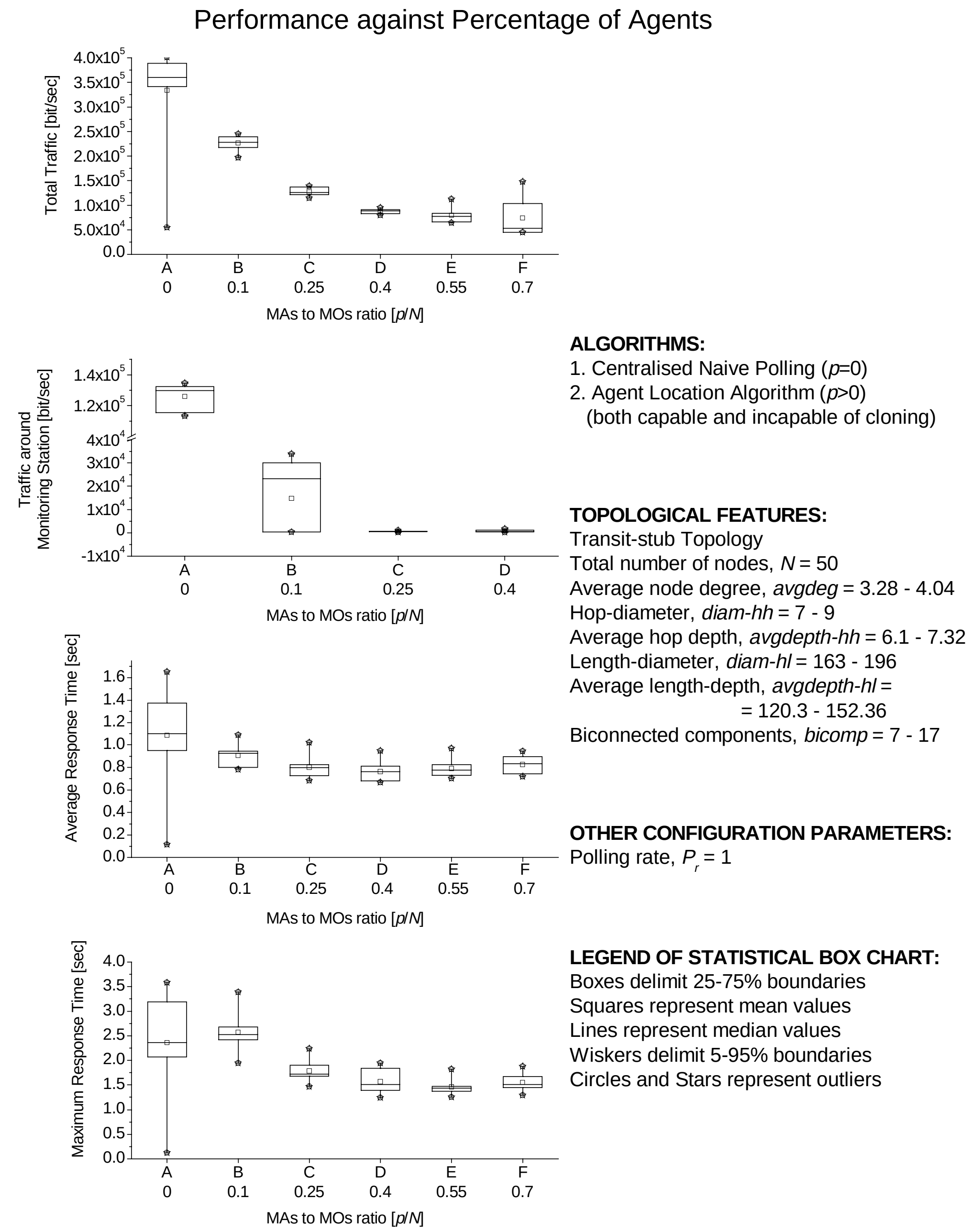
8.4 Performance against Percentage of Agents

The simulations related to the above sections have been carried out by keeping the percentage of agents constant with respect to the total number of monitored nodes. This has been done to study the sensitivity of the agent system to topological parameters (such as *N* and *D(u)*) and to monitoring parameters (such as P_r).

This section takes a step forward by studying the sensitivity to the actual number of agents, when topological and monitoring parameters are kept constant. Deciding on how many agents should be used for a given task and network is not a simple task. This decision tends to be crucial when it comes to minimising overhead to performance ratios.

Figure 8-11 depicts the impact that the percentage of agents has on the various performance parameters, in the form of the usual statistical box diagrams. In this case, the *X* axis reports (on a non-linear scale) the ratio between number of agents and number of monitored nodes, *p/N*.

The first noticeable difference between the centralised approach (p/N=0) and the various agent solutions ($0<p/N\leq1$) is that the former tends to exhibit a larger variance. Typically, the centralised approach is more sensitive to the topological variations among the different simulation runs. For instance, even relatively small variations in node degree in the vicinity of the polling station may introduce new bottlenecks which, in turn, may cause dramatic increase in traffic and/or response time. On the contrary, the agent solution by exploiting distribution is less prone to that problem.



ALGORITHMS:
1. Centralised Naive Polling (*p*=0)
2. Agent Location Algorithm (*p*>0)
(both capable and incapable of cloning)

TOPOLOGICAL FEATURES:
Transit-stub Topology
Total number of nodes, *N* = 50
Average node degree, *avgdeg* = 3.28 - 4.04
Hop-diameter, *diam-hh* = 7 - 9
Average hop depth, *avgdepth-hh* = 6.1 - 7.32
Length-diameter, *diam-hl* = 163 - 196
Average length-depth, *avgdepth-hl* =
= 120.3 - 152.36
Biconnected components, *bicomp* = 7 - 17

OTHER CONFIGURATION PARAMETERS:
Polling rate, P_r = 1

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-11. Statistical box plots depicting the impact of the percentage of agents.

Another reason for difference between centralised polling (*p*=0) and the agent solution (*p*>0) originates in the way simulations are carried out. In the former case, we aimed at placing the central monitoring station in a central node for a fair comparison with the agent approach (we wanted to compare the best configuration of centralised polling with our agent-base system). However, in order to randomise the simulation process it was necessary to repeat the simulations for subsequent randomly generated network topologies. The topology generator does generate topologies belonging to the same family (*e.g.*, characterised by comparable values of average topological parameters), but it does not generate symmetrical networks.

Hence, the algorithm which aims at placing the monitoring station in a central location does not always manage to find such node. That is why the statistical box diagrams presented in this section often exhibit a larger variability for *p*=0.

On the contrary, the fact that the agent approach tends to exhibit a smaller variability (for *p*>0) indicates that the proposed agent location algorithm succeeds in placing the agents near-optimally regardless of the randomisation of the topology generation adopted in subsequent simulations.

Figure 8-12 presents the best-fit functions relative to the above results. This time the curves exhibited non-linear behaviour. Very accurate exponential functions where found to fit traffic values, whereas response time values were best fitted by 3^{rd}-order polynomial functions. Both traffic and response time decreased significantly in the range 0<*p/N*<0.15. However, for larger values of *p/N* the improvement became negligible. Interestingly, response time may even increase in the upper band. In fact, despite the agent-to-monitored object communication load tends to decrease when the number of agents increases, the reverse tends to happen to the agent-to-station load. It was concluded that a larger portion of agents is neither convenient nor useful. In fact, the larger is the number of agents, the larger agent deployment overheads will be. From the above simulations it can be concluded that agent system can introduce significant performance benefits even when the number of agents is relatively small, typically for 0<*p/N*<0.15.

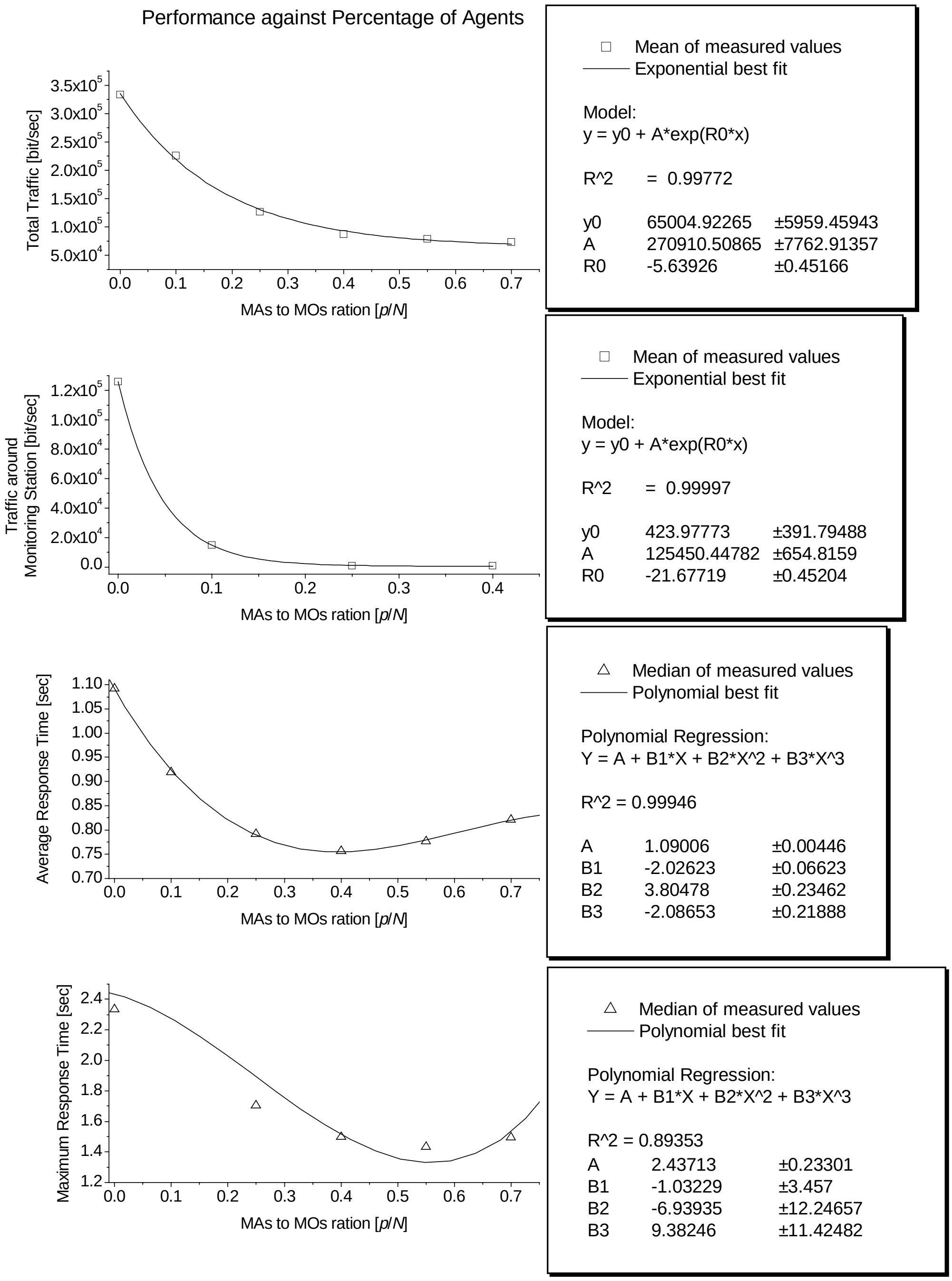


Figure 8-12. Best-fit performance functions based on percentage of agents.

We should add a comment at this stage on the apparent qualitative difference between traffic and response time functions of Figure 8-12. The former were fitted by monotonically decreasing exponential functions, whereas the latter were fitted with polynomial functions. The apparent difference should not induce to the conclusion that traffic does not exhibit the typical ‘U-shaped’ behaviour of response time found also in Chapter 7. The reason for this apparent discrepancy lies in the different order of magnitude of those two functions. In other terms, the simulations were not able to detect the expected increase in traffic in the upper band of *p/N* because, in the example agent system configuration, the increment in agent-to-station load related to agent increase was negligible in comparison to the agent-to-monitored node. In Chapter 7, it was easier to show this particular aspect of the agent system by appropriately changing the configuration parameters. It should be noticed that the scale of traffic functions is in the order of 3E5 whereas typical response time values are in the order of 1 second.

Therefore, we are inclined to conclude that the results obtained herein are consistent with those of Chapter 7. The models of Chapter 7 suggest that both traffic and response time have a qualitatively analogous behaviour, which is not in definite contrast with the findings obtained through simulation.

8.5 Distance from Optimality

In the previous sections we have evaluated performance and scalability of the proposed agent solution in comparison with centralised polling. Here we want to assess the goodness of the proposed agent algorithm by evaluating its distance from optimality. We recall that in Chapter 7 we have found constraining conditions on the network topology which resulted in near-optimality. In this chapter we have relaxed those conditions by considering more general network topologies. Hence, our interest in evaluating distance from optimality through simulation aims at complementing the results of Chapter 7.

This is done by comparing the overall performance achieved when the agents are placed by the proposed algorithm with the hypothetical case in which those agents could be placed near-optimally. This comparison is interesting because it allows a quantitative assessment of the goodness of the algorithm with respect to the best possible case. The reference near-optimal agent location is computed using the software package SITATION, configured to run the *lagrangian* location algorithm, which guarantees at least near-optimality [Daskin 95]. It should be recalled that the *lagrangian* algorithm does not represent a viable alternative to the proposed algorithm mainly because it is computed centrally, on the basis of a network distance matrix, as discussed in Section 3.5, page 67.

The proposed approach is also compared with the case in which locations are computed randomly. This has been done to prove that the proposed solution is significant, by measuring its distance from randomness. The ideas was to discard the possibility that the proposed agent system would result in performance comparable with the those achievable using random location approaches.

In the following subsections, the results of the simulations are presented first individually, for each of the above three cases (Sections 8.5.1, 8.5.2, and 8.5.3). Those results are finally merged and discussed comparatively (Section 8.5.4). The approach followed for the analysis is analogous to the one adopted in the previous sections. However, in this case the analysis is carried out in a more general fashion by adopting *topological* rather than *physical* performance indicators. The first indicator is the network *total hop-distance*. This is directly related to the total steady-state traffic, since the latter is proportional to the incurred bit-rate multiplied by the number of links traversed by packets. Average and maximum hop-distance are also measured.

The second performance indicator is the *maximum weighted distance*. This is directly related to response time, since the latter is proportional to the costs associated to link traversal. For simplicity, we assumed a constant, unitary weight for all the links. Thus, we have simulated an homogeneous network in which the topology was generated with the GT-ITM tool, whereas all links where identical. Average and total weighted distance are also measured and an exponential best-fit model is adopted to interpolate the results for an easier comparison.

It should be mentioned, however, that due to the complexity and long simulation times, best-fit functions are build on just four points. Thus, it is not possible to draw definite conclusions on whether those points operated at polynomial or exponential level.

On the other hand, our aim was to adopt models which allowed a comparison among the various simulation results, rather than studying the physical phenomena behind them. A suitable model to interpolate simulation data was found to be the following:

$$y = P_1 \exp(P_2 x) + P_3 \exp(P_4 x)$$

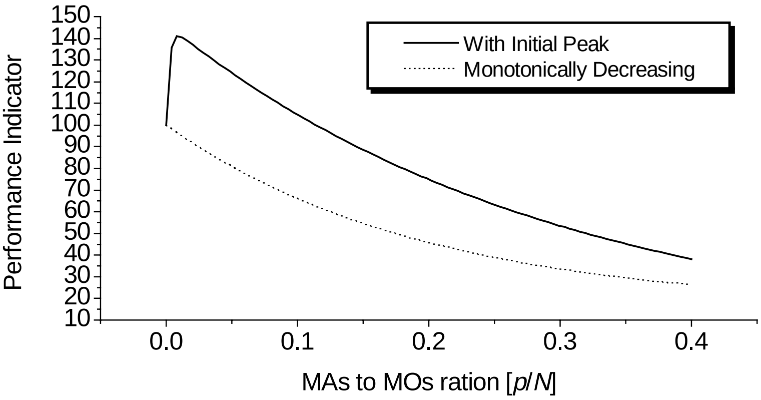


Figure 8-13. Features of the model used to interpolate results.

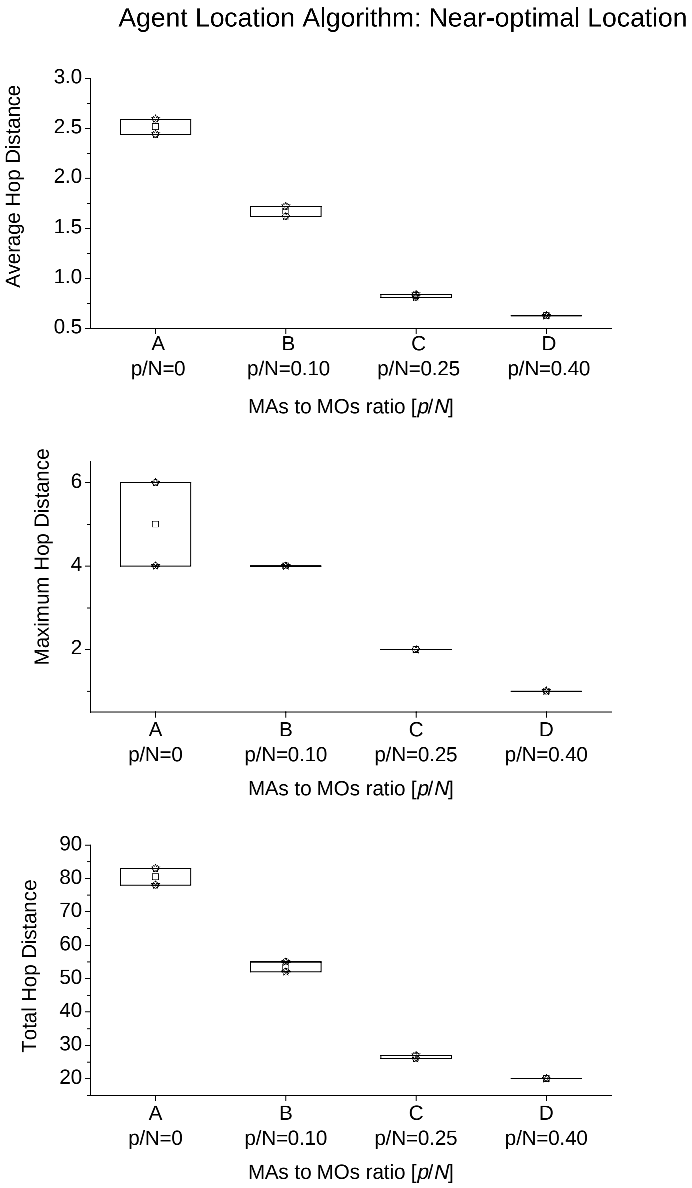
Depending on the values of the parameters (*P_1* to *P_4*), the above function may exhibit one of the two qualitative behaviours depicted in Figure 8-13. The first one (solid line) shows a monotonically decreasing function; whereas the second one (dotted line) exhibits an initial peak, followed by a predominant monotonically decreasing behaviour. In the remaining part of this chapter, we shall encounter both cases.

It should be stressed that the best-fit functions computed hereafter have the purpose of giving an indication of the possible behaviour of the data points found by simulation. However, since these functions are computed on the bases of only four data points it is not possible to draw any definite conclusions on the precise behaviour of the underlying mechanisms.

After considering various polynomial and non-polynomial best-fit models, the exponential model has been adopted because it has been found to approximate fairly well the data points generated by the simulations (*i.e.*, the correlation coefficient R^2 tends to be fairly close to 1). Clearly, we cannot conclude that the underlying mechanisms found by simulations are of exponential nature. A larger number of data points would increase significantly the significance of the best-fit functions. That would in turn require a significant amount of simulations which have not been conducted due to time constraints.

8.5.1 Near-Optimal Agent Location

All the statistical box diagrams presented in this section are characterised by distinctly non-overlapping 25-75% boxes, a clear indication of statistical significance of the measured values. The remaining diagrams depict the exponential best-fit functions, all characterised by relatively high values of R^2, which proves the accuracy of those functions.



TOPOLOGICAL FEATURES:
Total number of nodes, *N* = 32
Average node degree, *avgdeg* = 2.56 - 2.75
Hop-diameter, *diam-hh* = 8 - 10
Average hop-depth, *avgdepth-hh* = 6.12 - 7.97
Biconnected components, *bicomp* = 10 - 19

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-14. Statistical box plots depicting hop-distances achieved with the near-optimal, lagrangian location algorithm.

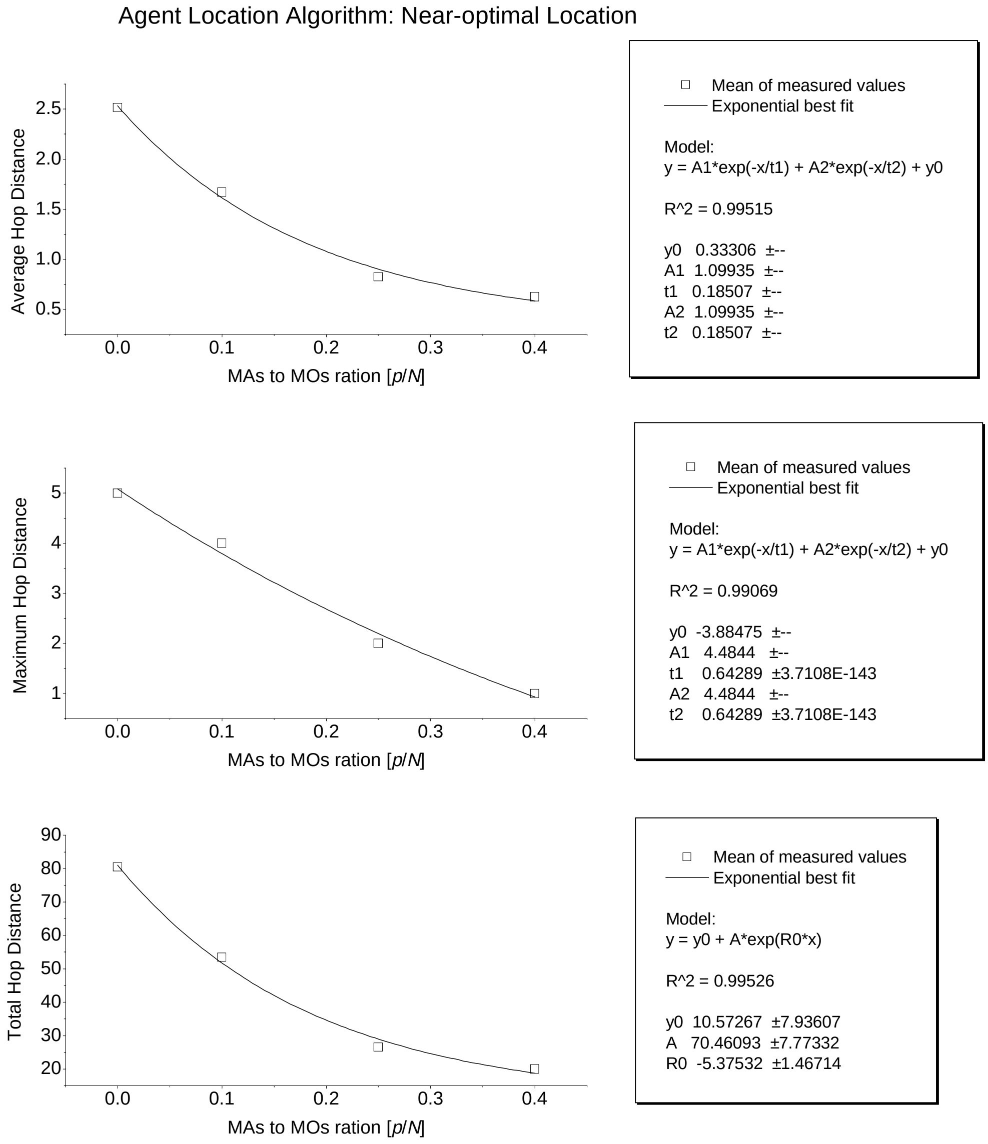
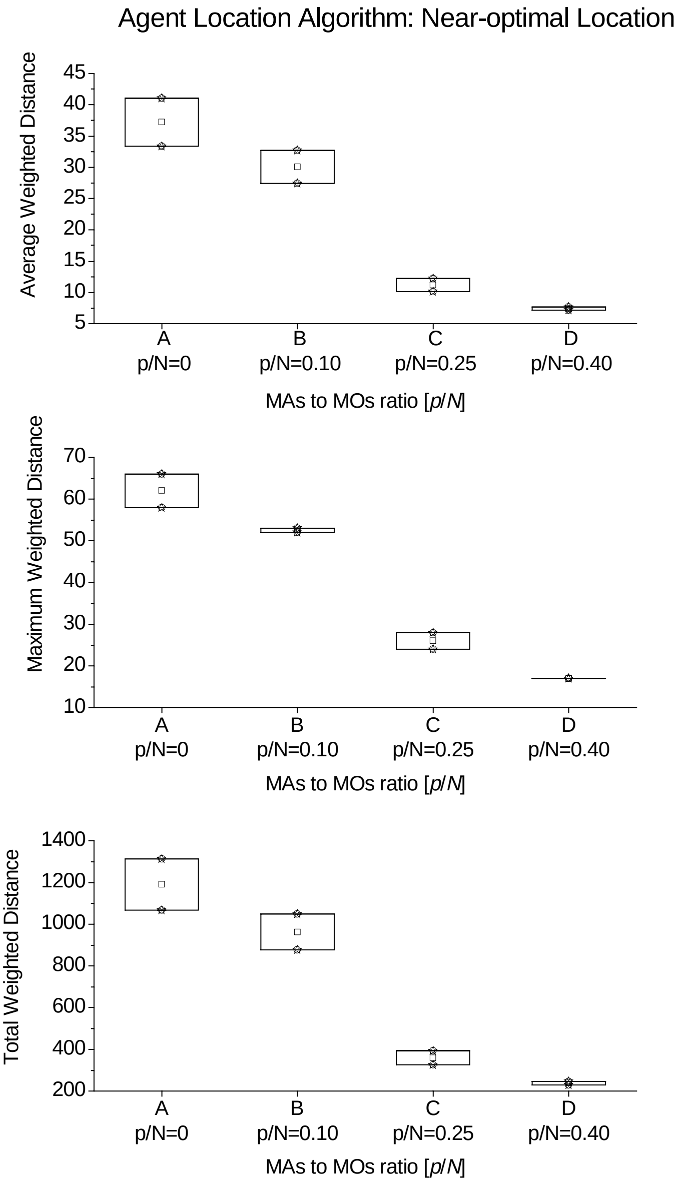


Figure 8-15. Best-fit hop-distance functions achieved with the near-optimal, lagrangian location algorithm.



TOPOLOGICAL FEATURES:
Total number of nodes, *N* = 32
Average node degree, *avgdeg* = 2.56 - 2.75
Length-diameter, *diam-hl* = 134 - 144
Average Length-depth, *avgdepth-hl* = 103.22 - 105.22
Biconnected components, *bicomp* = 10 - 19

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-16. Statistical box plots depicting weighted-distances achieved with the near-optimal, lagrangian location algorithm.

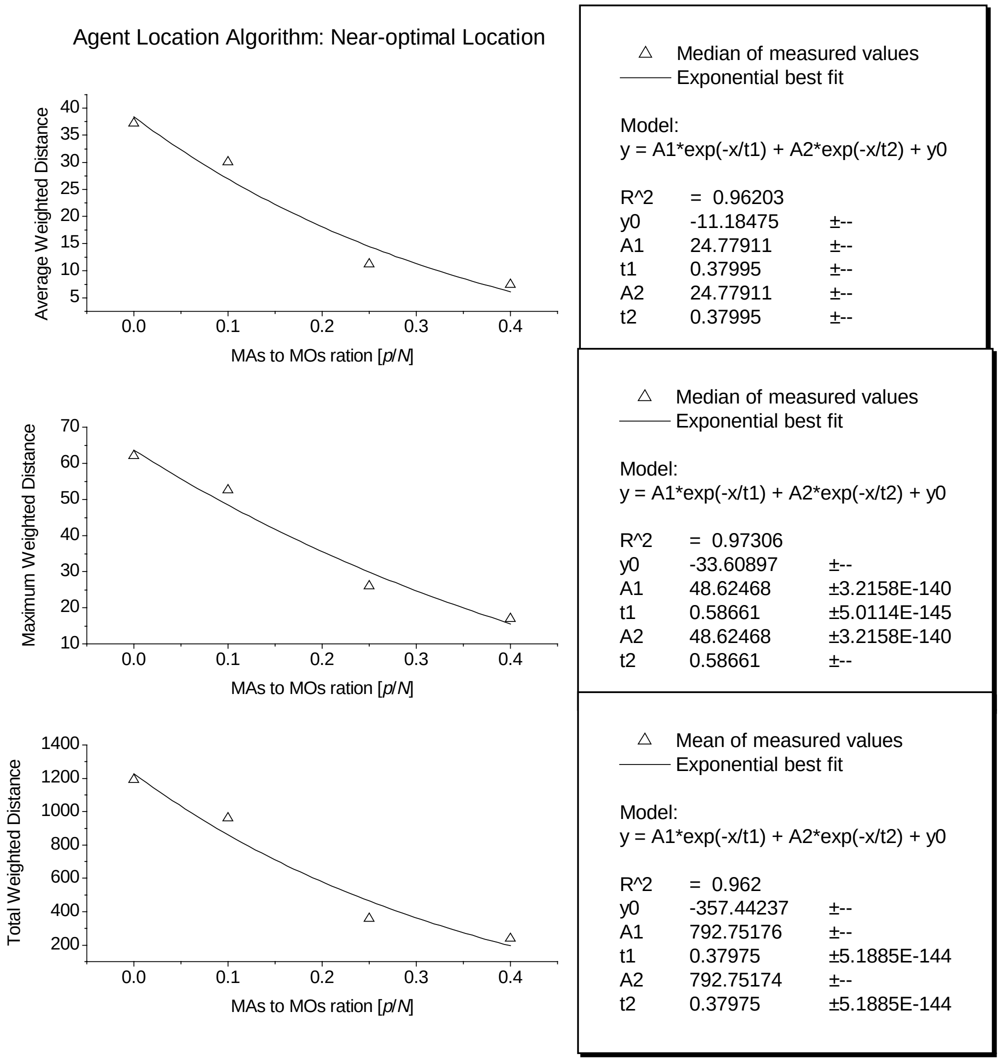
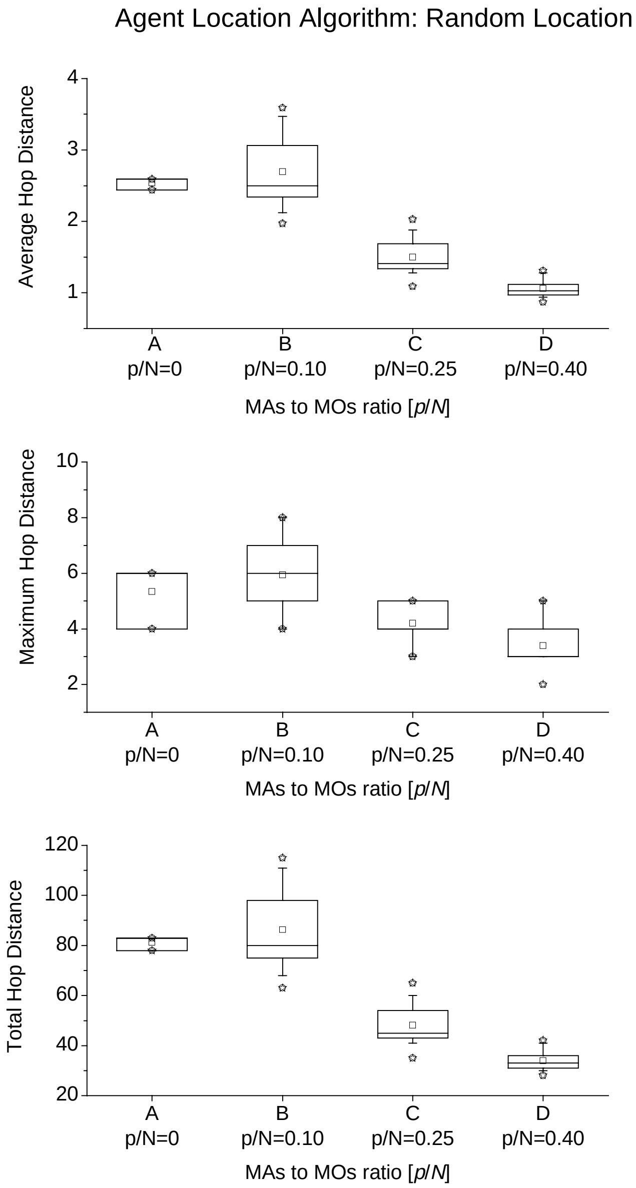


Figure 8-17. Best-fit weighted-distance functions achieved with the near-optimal, lagrangian location algorithm.

8.5.2 Random Agent Location

In the case of the random agent location algorithm a peculiarity can be observed. All indexes of central tendency are characterised by an initial peak that, for the example agent configuration, corresponds to *p/N*=0.10. This was initially been attributed to anomalies in the simulation software. After a closer analysis it was realised that this was not case. Those peaks are caused by the fact that a relatively small number of agents, if badly placed, could represent a worse situation than that of a centralised approach, whereby the polling station was placed in a central location. Obviously, this anomaly disappears when the number of agents becomes sufficiently large since, in this case, the improvements deriving from a higher level of distribution prevail.

Very accurate best-fit functions were achieved by interpolating on a linear combination of two exponential functions, as proved by the relatively high values of R^2.



Agent Location Algorithm: Random Location

TOPOLOGICAL FEATURES:
Total number of nodes, *N* = 32
Average node degree, *avgdeg* = 2.56 - 2.75
Hop-diameter, *diam-hh* = 8 - 10
Average hop-depth, *avgdepth-hh* = 6.12 - 7.97
Biconnected components, *bicomp* = 10 - 19

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-18. Statistical box plots depicting hop-distances achieved with the random location algorithm.

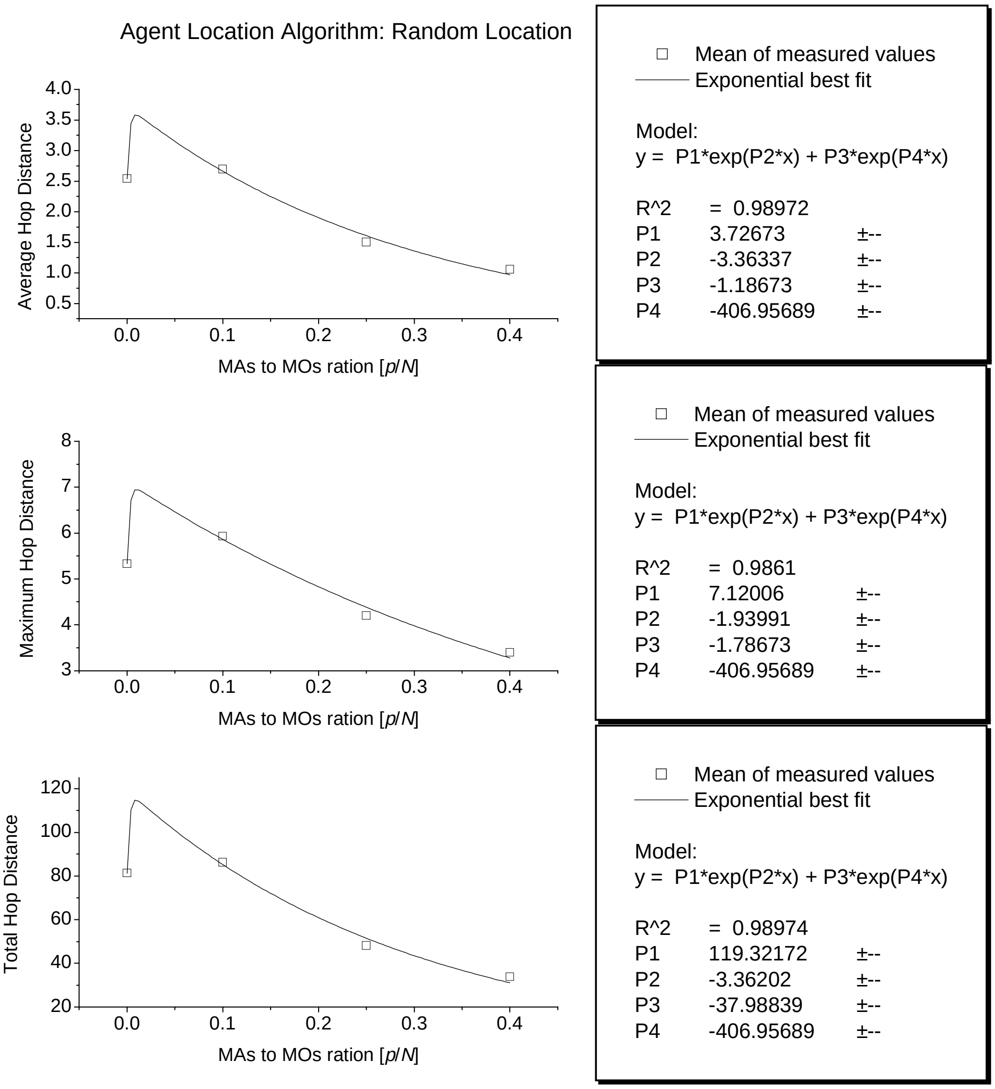
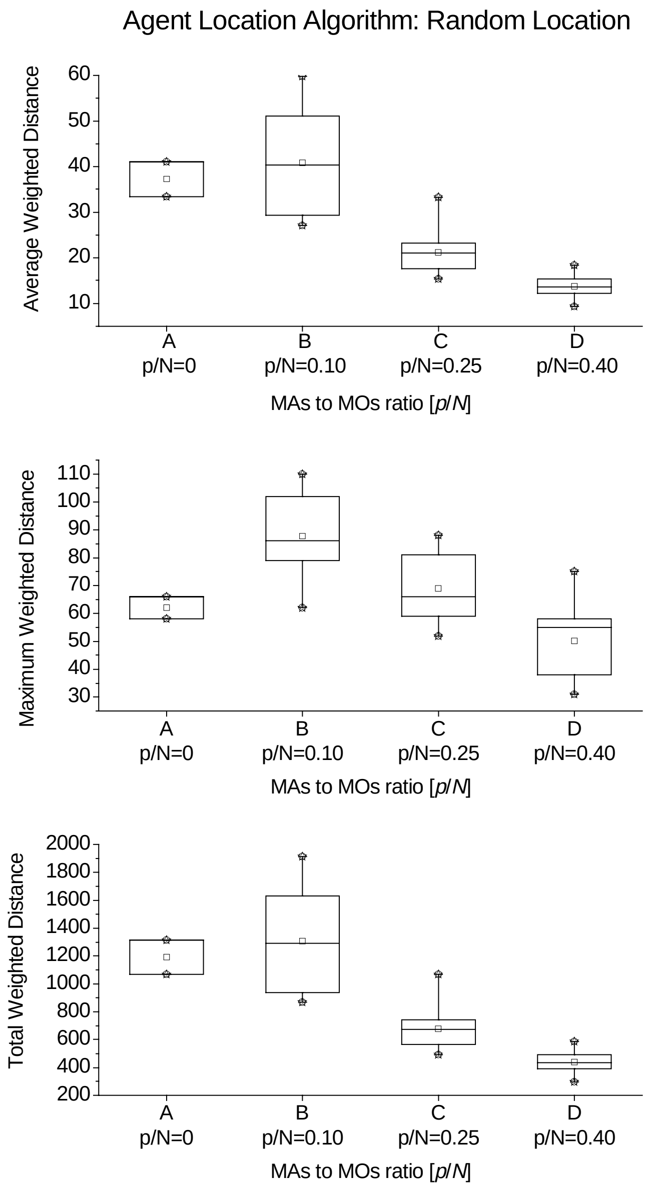


Figure 8-19. Best-fit hop-distance functions achieved with the random location algorithm.



TOPOLOGICAL FEATURES:
Total number of nodes, *N* = 32
Average node degree, *avgdeg* = 2.56 - 2.75
Length-diameter, *diam-hl* = 134 - 144
Average Length-depth, *avgdepth-hl* = 103.22 - 105.22
Biconnected components, *bicomp* = 10 - 19

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-20. Statistical box plots depicting weighted-distances achieved with the random location algorithm.

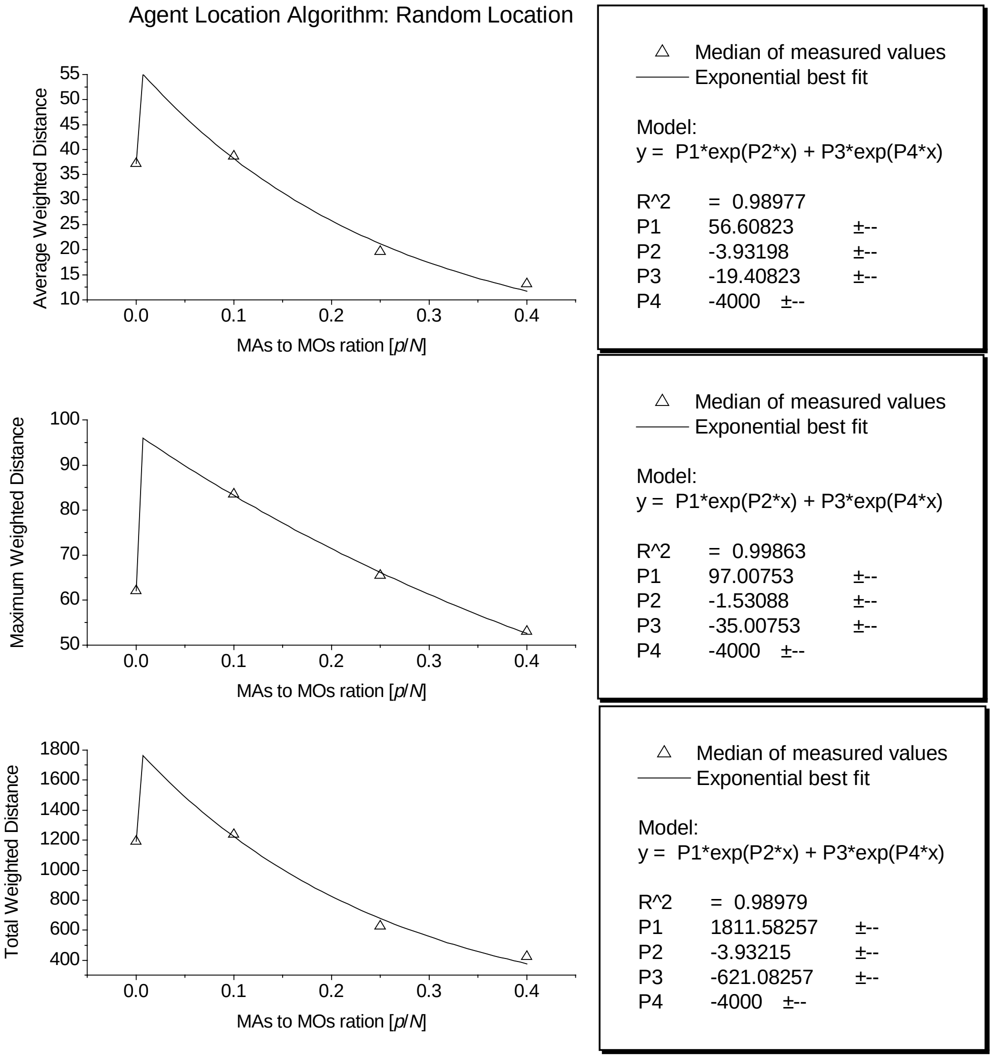
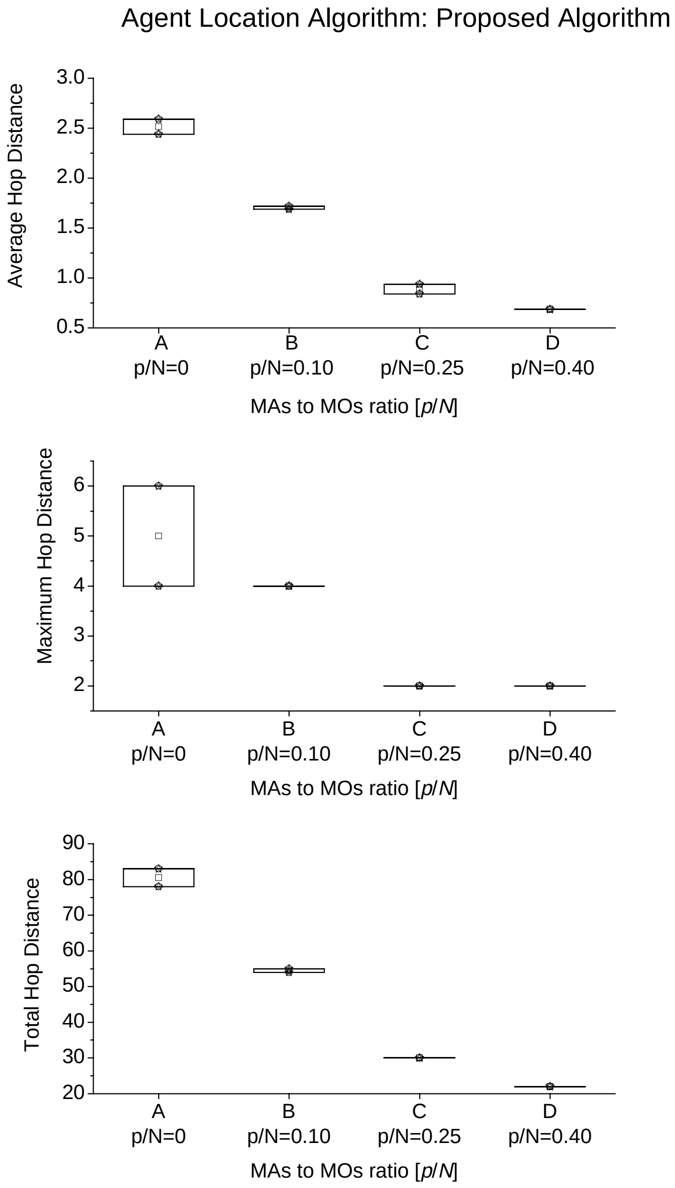


Figure 8-21. Best-fit weighted-distance functions achieved with the random location algorithm.

8.5.3 Proposed Agent Location

The diagrams obtained in the case of the proposed agent location algorithms are substantially analogous to those relative to the above near-optimal, lagrangian algorithm; therefore there is no need to add further comments.



TOPOLOGICAL FEATURES:
Total number of nodes, *N* = 32
Average node degree, *avgdeg* = 2.56 - 2.75
Hop-diameter, *diam-hh* = 8 - 10
Average hop-depth, *avgdepth-hh* = 6.12 - 7.97
Biconnected components, *bicomp* = 10 - 19

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-22. Statistical box plots depicting hop-distances achieved with the proposed location algorithms.

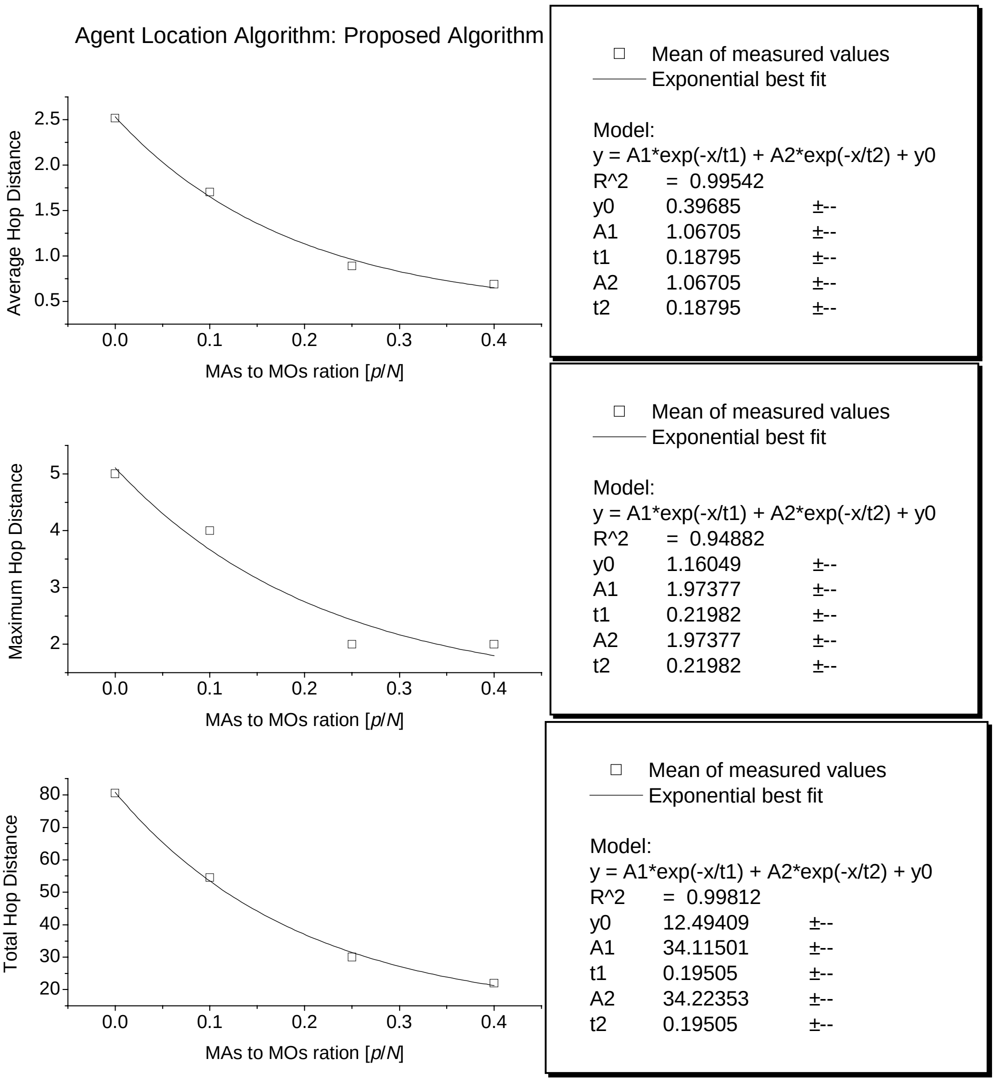
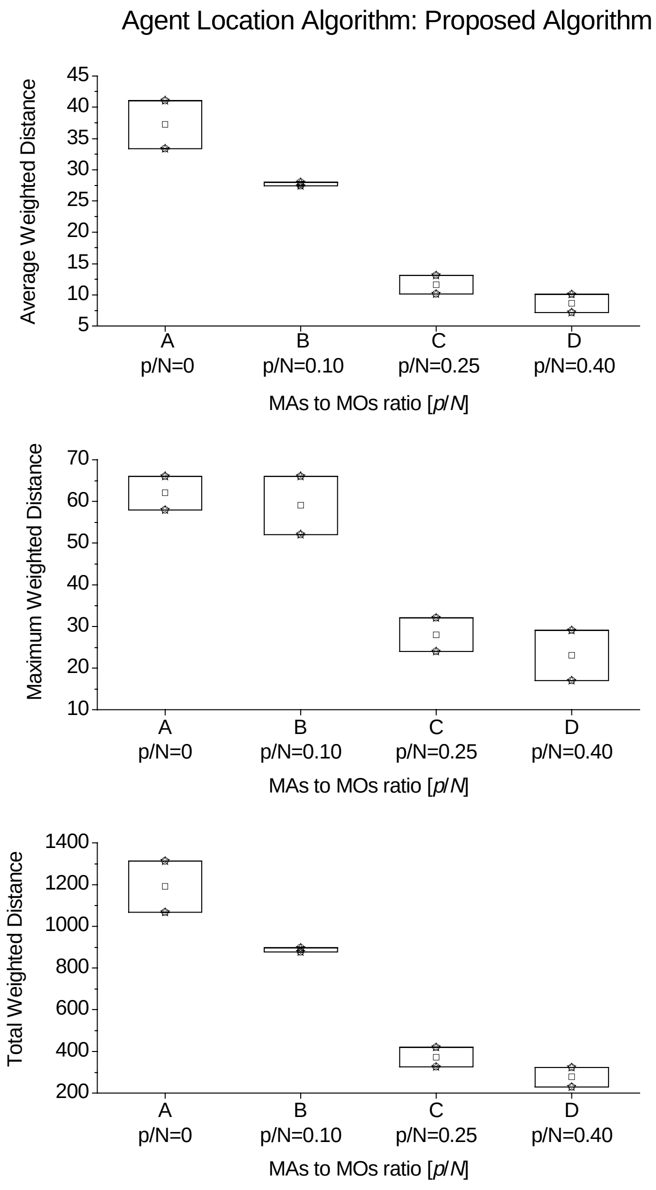


Figure 8-23. Best-fit hop-distance functions achieved with the proposed location algorithms.



TOPOLOGICAL FEATURES:
Total number of nodes, *N* = 32
Average node degree, *avgdeg* = 2.56 - 2.75
Length-diameter, *diam-hl* = 134 - 144
Average Length-depth, *avgdepth-hl* = 103.22 - 105.22
Biconnected components, *bicomp* = 10 - 19

LEGEND OF STATISTICAL BOX CHART:
Boxes delimit 25-75% boundaries
Squares represent mean values
Lines represent median values
Wiskers delimit 5-95% boundaries
Circles and Stars represent outliers

Figure 8-24. Statistical box plots depicting weighted-distances achieved with the proposed location algorithms.

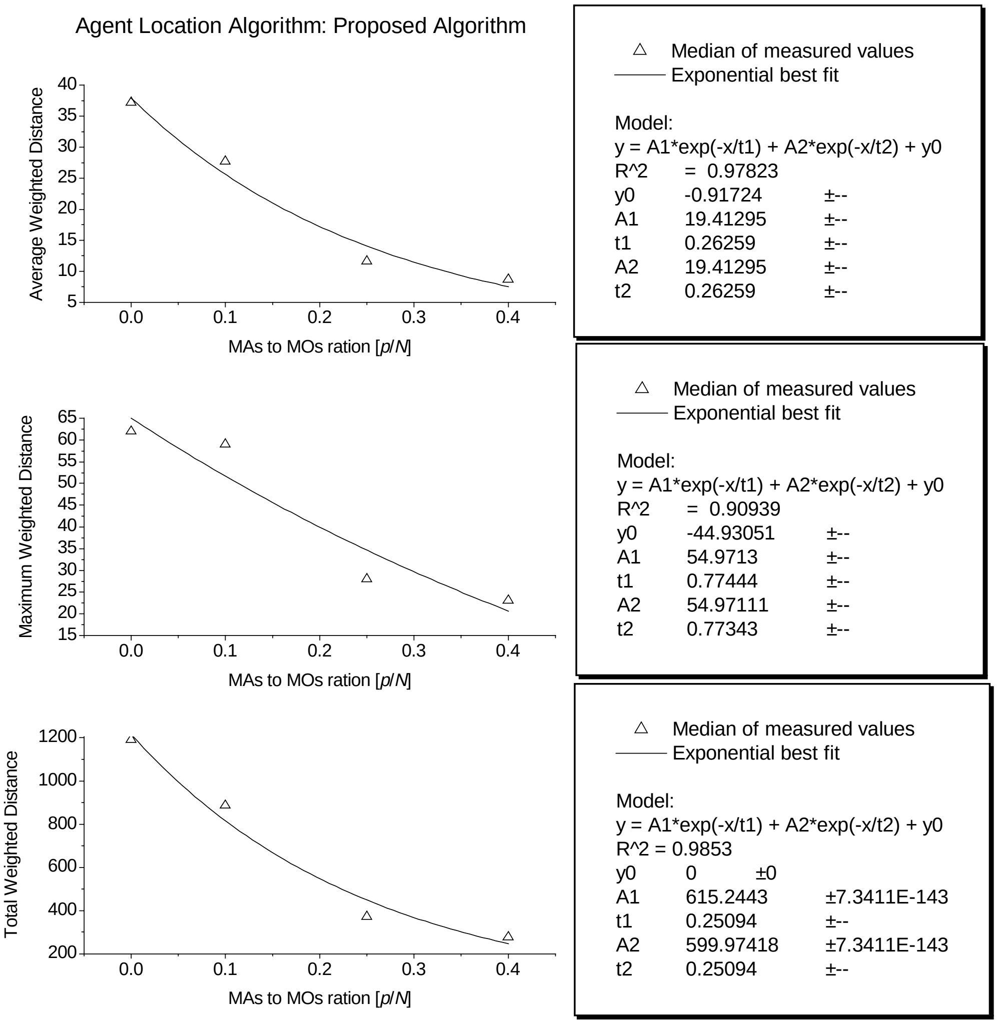


Figure 8-25. Best-fit weighted-distance functions achieved with the proposed location algorithms.

8.5.4 Comparison

The above simulation results are merged in Figure 8-26 and Figure 8-27 for a better comparison. It can be observed that the proposed approach leads to traffic values that are always smaller than those that would be achieved with the lagrangian algorithm, which is provably near-optimal. Hence the agent solution is near-optimal too. In particular, in the example agent system configuration, a percentage improvement in the range of 0-3% was measured. It should be stressed once again that the lagrangian algorithm cannot be used to solve the agent location problem for the reasons already mentioned in Chapters 2 and 3.

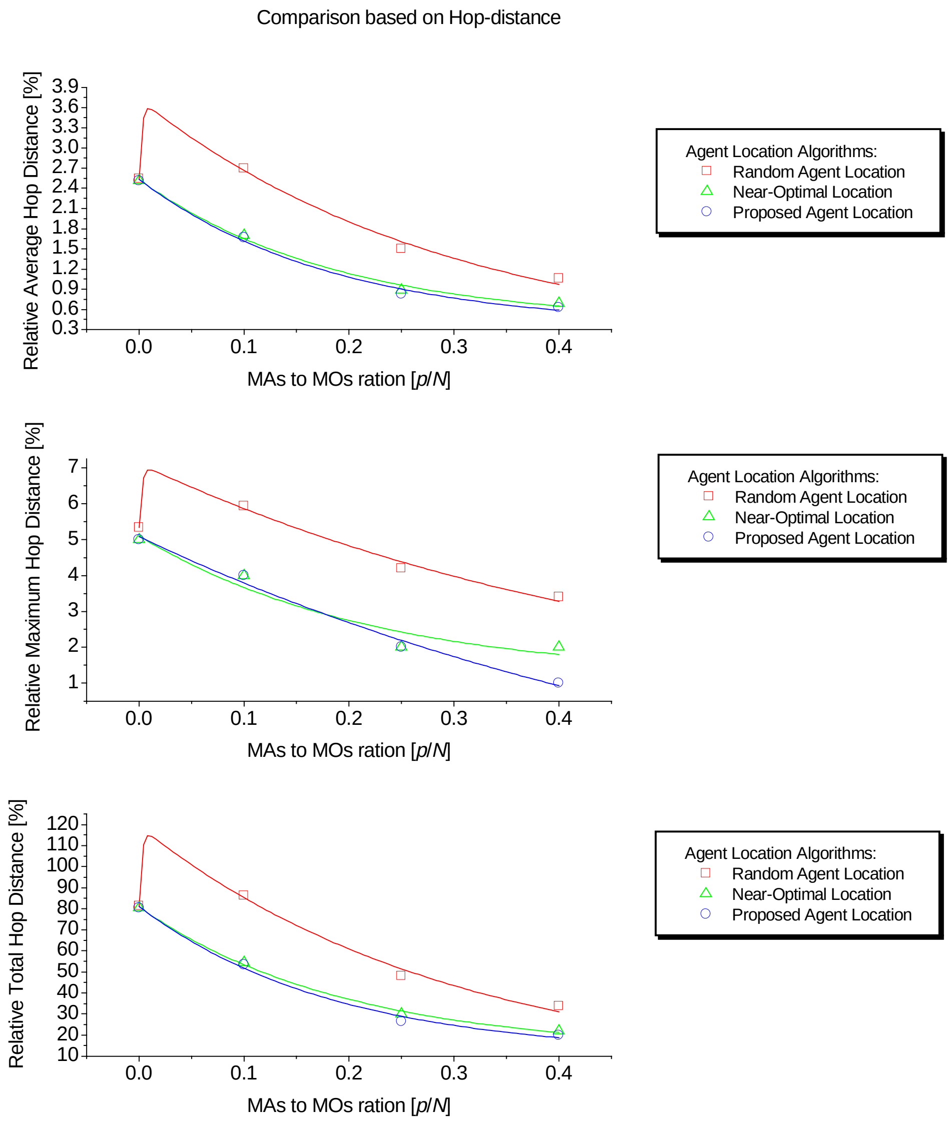


Figure 8-26. Comparison based on hop distance.

Furthermore, it should be noted that simulations characterised by up to a large number of agents (*p/N*=0.4) were performed for the sake of completeness. However, for a more efficient resource utilisation, typical agents-to-nodes ratios are envisioned to be much smaller (*p/N*<0.1).

The fact that the total hop-distance achieved by placing the agents in a random fashion is very far from the proposed solution (38-48% difference for *p/N*<1) provides another good justification for the adoption of the agent based approach. The percentage reduction in traffic with respect to centralised polling (*p/N*=0) is also significant. For instance, for *p/N*=0.1 the reduction in traffic was greater than 30% with monotonical increases with *p/N*. Finally, the fact that those three curves tend to converge for large values of *p/N* is not unexpected since when p/N=1 the number of agents equals the number of nodes. Hence each of the three algorithms equally succeed in placing the agents evenly.

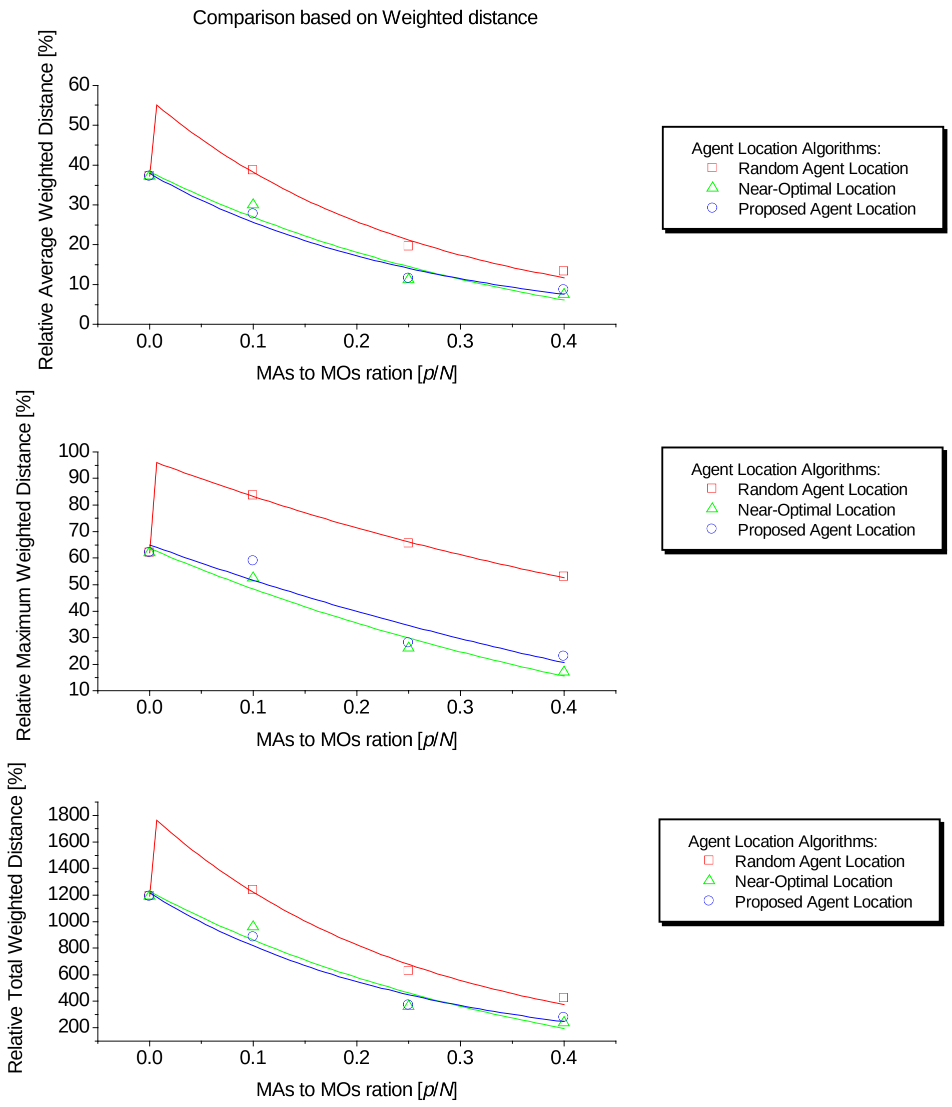


Figure 8-27. Comparison based on weighted distance.

Figure 8-27 allows a similar comparison based on maximum weighted distance, which is related to response time. However, in this case the agent location curve, though very close to the near-optimal one, does not exhibit any inferior value. In particular, the distance from near-optimality is 0-5% for *p/N*<1. This result was expected since the simulated agent system was optimised to minimise traffic, not response time. Further simulations, not reported here for brevity, indicated that near-optimality with respect to response time can be achieved with trivial modifications to the agent algorithm.

8.6 Adaptability

The ability of a monitoring system to adapt to network changes is a very attractive property, especially in view of the dynamic behaviour of current and future networks. Causes of that are, for instance, network congestion, device failure, terminal mobility, and mobile computing.

The conventional approach is to achieve adaptability by dynamically changing the routing tree rooted at the monitoring station. This is performed by the routing protocols. Consequently, because of congestion or failure, monitoring packets are re-routed through generally longer paths and both traffic and response times tend to deteriorate.

The proposed approach is an example of active monitoring; agents keep sensing the network during their operation and can periodically estimate the cost of alternative locations. Agent migration is triggered when the cost reduction justifies the migration overheads. Agents adopt the same logic used during deployment time to sense the network and estimate costs associated to candidate neighbour nodes. This adaptation strategy is based solely on local decision. Locality has the advantage of simplicity but has the drawback of not considering more global optimisation strategies.

A part of a preliminary study of this approach, a simple scenario, in which 2 links located in the vicinity of the central monitoring station fail, has been simulated. Traffic and response time were measured before the failure. After the failure the routing protocol readjusted the routing tables and full connectivity was achieved. In addition, the agent system reconfigured itself by relocating some of the agents. Steady-state traffic and response time were measured again.

Figure 8-28 shows the snapshot of those performance indicators taken before and after the link failure, respectively. With the centralised polling solution (*p*=0), both request and response packets get re-routed through longer paths. Consequently, both traffic and response time increase significantly – they almost doubled in our scenario. On the contrary, with the agent system the performance degradation at steady state was in the order of 5-10%.

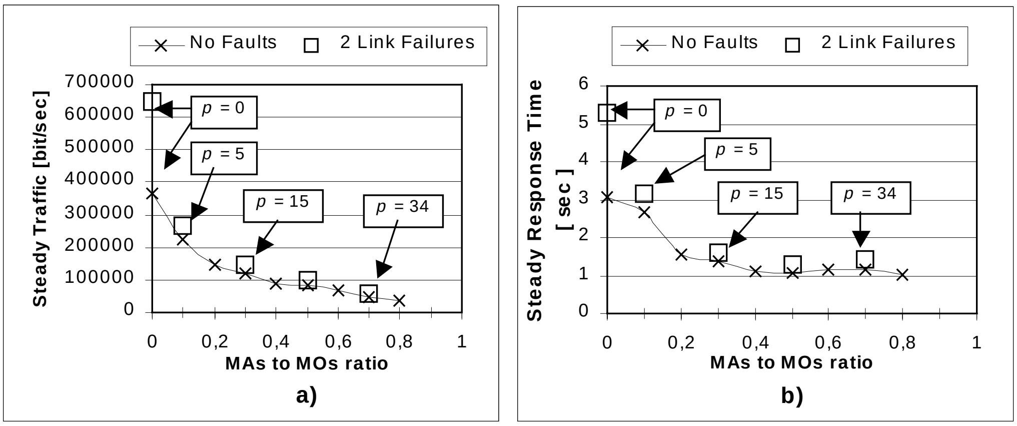


Figure 8-28. Adaptability. a) Traffic at steady state; b) Response time at steady state.

An important design choice concerns the number of agents initially deployed. This has an impact both at deployment time and run time. At deployment, the more agent we deploy, the higher the deployment overheads will be. A large number of agents also means a larger consumption of computational and memory resources at the hosting nodes. The benefit of a relatively large number of agents are felt at run time, since a larger number of agents means a higher level of distribution, which in turn will generally result in better steady-state performance.

Our simulations have identified another advantage related to increasing the number of agents. As this increases, the percentage of agents that need to migrate in face of changing network conditions tends to decrease more than linearly, as demonstrated by Figure 8-29.

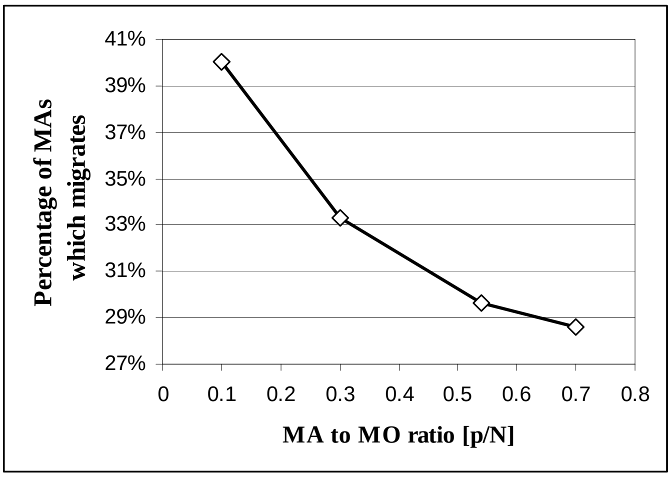


Figure 8-29. Impact of total number of agents on percentage of agent migration occurrences.

It should be mentioned that the above simulations on adaptation were intended as a preliminary study of “adaptation through agent migration” rather than pretending to be comprehensive. The idea was to see whether the same principles used to solve the location problem where still valid at run time – *i.e.*, during the execution of the distributed monitoring task.

The initial study suggests that this is indeed a promising direction, though more extensive simulations should be carried out before generalised conclusions can be drawn. In particular, agent migration, if not properly triggered, may lead to instability that can be disastrous. Furthermore, migration policies including more sophisticated heuristics need to be studied.

All these ideas are extremely interesting from a research point of view but it was decided not to pursue them further in the context of this PhD research since this topic deserves a more comprehensive and methodical experimentation. For instance, it would be interesting to evaluate the sensitivity of our agent system to variables such as number of faults, location of faults, fluctuating network congestion, etc. The effect of migration on robustness and system stability is another interesting topic.

8.7 Discussion and Conclusions

This chapter assesses the advantages that agents bring to distributed monitoring in terms of reduced traffic and shorter response times. A scalability study based on monitoring parameters (polling rate – Section 8.1) and topological parameters (number of monitored nodes – Section 8.2 – and network diameter – Section 8.3) has been presented. This allowed a quantification of the improvements achievable with respect to the more conventional centralised polling approach.

In essence, the simulation results of Sections 8.1 to 8.3 are consistent with the results achieved by mathematical analysis in Chapter 7. Performance indicators (traffic and response time) operate linearly with individual scalability indicators (P_r, N, $D(u)$) despite the fact that simulations have addressed the case of general network topologies, whereas the mathematical analysis of Chapter 7 assumed constraining conditions on network topology.

The simulations have also demonstrated that significant performance improvements are achieved at steady-state by the agent system in comparison with centralised monitoring. The amount of improvement can be quantified by setting up the various monitoring parameters and running the simulator. Crucial factors have been found to be the number of MAs, as well as the level of aggregation performed locally by the MAs.

The impact of the number of MAs on performance was studied in particular in Section 8.4. Again, results were consistent with those of Chapter 7 despite the different assumptions on network topology. An upper bound beyond which it was not worth increasing the number of agents was found. In essence, this behaviour depends on the fact that by increasing the number of agents, the agent-to-monitoring station communication load tends to increase as well. This effect became negligible when we increased the level of aggregation performed by the agents, an important factor which has a direct impact on the traffic generated by the agents. As a rule of thumb the agent-to-monitored nodes ratio should be large enough to impact significantly performance but should be smaller than 0.10. Various simulations showed in fact that if MAs are more than 10% of the number of monitored nodes, overheads related to agent deployment and agent-to-monitoring station communication tend to have a measurable negative impact. It should be said, however, that particular monitoring tasks may lead to slightly different conclusions on the best number of MAs to be adopted.

In the second part of the chapter (Section 8.5), the study of distance from optimality demonstrated that the proposed agent approach is near-optimal and significantly better than the performance achievable by random placement of agents. This result was unexpected when the agent location algorithm was designed and probably represents one of the most important outcomes of the thesis work. The initial aim was to solve the agent location algorithm efficiently and viably. Near-optimality was not one of the targets.

Figure 8-30 depicts the main results of Section 8.5. The total hop-distance is directly related to the total steady-state monitoring traffic. It can be observed that the proposed location algorithm leads to traffic values that are always smaller than those that would be achieved with the lagrangian algorithm, which is provably near-optimal. Hence, our agent-based algorithm is near-optimal too. In particular, a percentage improvement in the range of 0-3% was measured. It should be stressed once again that the lagrangian algorithm could not be used to solve the agent location algorithm for the reasons already mentioned in Chapter 2 and 3.

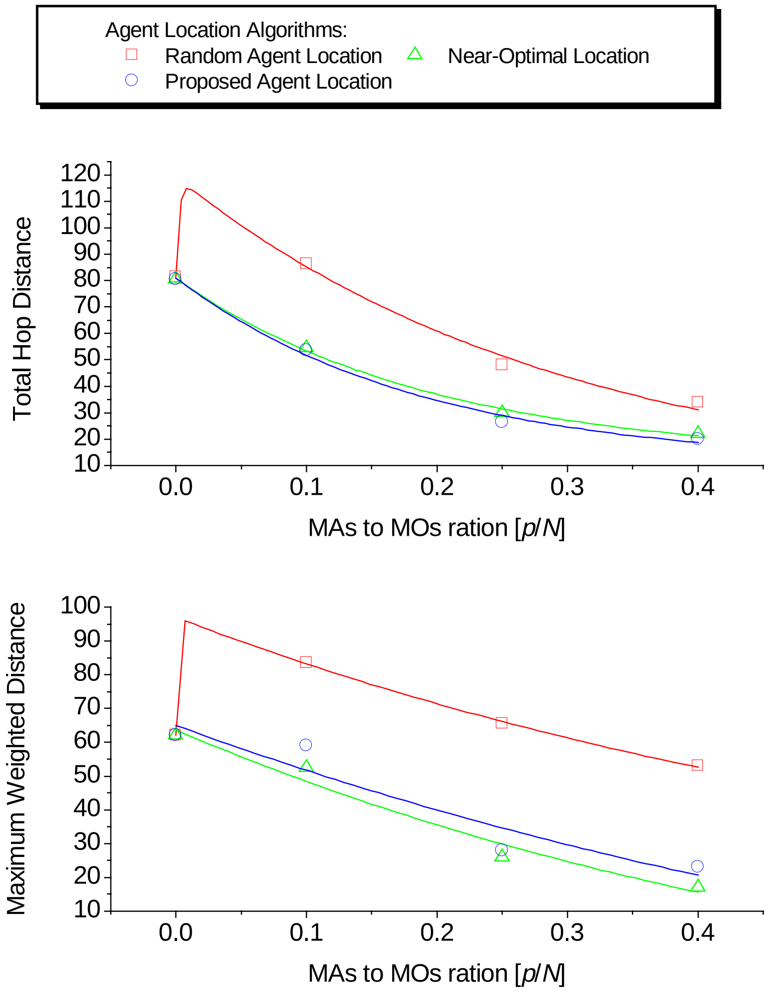


Figure 8-30. Agent system near-optimality.

It should be noted that, for the sake of completeness, we simulated situations characterised by up to a large number of agents (*p/N*=0.4). However for a more efficient resource utilization, typical "agents to nodes" ratios are envisioned to be much smaller (*p/N*<0.1). The fact that the total hop-distance achieved by placing the agents in a random fashion is very far from our near-optimal solution (38-48% difference for *p/N*<0.1) provides another good justification for the adoption of the agent-based approach. The percentage reduction in traffic with respect to centralised polling (*p/N*=0) is also significant. For instance, for *p/N*=0.1 the reduction in traffic will be greater than 30% and will increase monotonically with *p/N*.

Finally, the fact that the three curves tend to converge for large values of *p/N* is not unexpected since when *p/N*=1 the number of agents equals the number of nodes. Hence, under those conditions each of the three location algorithm will equally succeed in placing the agent evenly.

The plot which reports the maximum weighted distance (directly related to response time) for the three location algorithms is qualitatively analogous to the previous one. However, in this case the agent location curve, though very close to the near-optimal one, does not exhibit any inferior value. In particular, the distance from near-optimality is 0-5% for p/N<0.1. This result was expected since the simulated agent location algorithm was optimised to minimise traffic, not response time. Near-optimality with respect to response time can be achieved with proper modifications to the agent algorithm.

In the final part of this chapter (Section 8.6) we have presented an initial study of the adaptability of the agent system to network dynamics. Only a limited amount of simulations were carried out with the aim of having a 'feel' of the system capability to cope with link failures. Results were very promising and encourage furthering the study of agent adaptation through migration.

The feasibility and the advantages of shifting to the 'active decentralised' monitoring paradigm can be seen from the above simulation-based analysis of an example system based on autonomous mobile agents. In this system, the decision as to where best to locate the agents makes use of 'local' information in a 'distributed' fashion. Each agent identifies a better location and decides whether to 'clone' further agents or not - the decision is based solely upon local routing information. Each agent acts autonomously to solve a portion of the global configuration problem, which is solved in a distributed fashion.

Our study has shown that a system based on mobile agents will out-perform a conventional system in many typical situations. It has also helped in identifying conditions in which agents are not recommended. Mobile agents were particularly effective in reducing the communication and processing bottleneck located at the monitoring station. Despite the simplicity of the agent configuration algorithm, agents end-up evenly distributed for network topologies resembling inter-networks and hierarchical networks, respectively. Consequently, the achieved improved scalability and performance are not entirely surprising. However, our analysis provides a quantitative comparison of performance and scalability between centralised monitoring and agent-based distributed monitoring.

The main factor limiting the use of agents is their migration overhead - clearly, migration traffic and migration time are not incurred in the centralised approach. The former depends on agent size which, in turn, depends on the complexity of agent functionality. Simple autonomous agents able to perform local aggregation of monitored data, generate statistics, and trigger alarms can be implemented in the Java language in modules of size in the order of 10 Kbytes [Knight 99].

Agent migration time is dominated by the agent serialisation/de-serialisation process, required to prepare an agent for transmission and execution, respectively. Depending on various implementation aspects, this process requires times in the order of hundreds of milliseconds, well above the tens of milliseconds required to transport the code over the wire. For this reason, the time to complete the deployment of all the agents will be of the order of seconds or tens of seconds, depending on the number of agents, and on the network size and average node degree. Traffic bursts incurred during the initial deployment or long deployment times may not be acceptable in some situations. For instance, monitoring tasks whose total duration is of the order of agent deployment time cannot be implemented using this approach.

Our analysis highlights a key advantage of using ‘full’ code mobility instead of degenerate cases or stationary agents *i.e.* adaptability. We have presented an example showing how agent migration can be used to implement a monitoring system adaptable to network changes, a particularly attractive property for dynamic networks.

Again, the agent migration overheads identified in our analysis give a clue as to the time-scales over which adaptation might be effective. Since agent migration time is in the order of a second, agents are suitable to compensate to changes within time-scales larger than a second.

In conclusion, the proposed agent-based solution is feasible for distributed monitoring and generally results in increased scalability and steady-state performance. However, there are conditions in which agents cannot compete with the simplicity and efficiency of the conventional centralised approach – *e.g.* in very short or very simple monitoring operations. Therefore, the ideal solution is to integrate code mobility into existing systems and benefit from the relevant advantages rather than rely completely on mobile software agent approaches. A seamless integration between existing and mobile agent based approaches should be the topic of future research.

Chapter 9

Conclusions

This final chapter summarises and discusses the thesis contributions, draws the main conclusions, and elaborates on possible avenues of research for further development of the thesis work.

9.1 Thesis Summary

This thesis is focussed on the investigation of code mobility and its application to distributed monitoring. A novel approach termed *active, distributed monitoring* is proposed and evaluated. Both 'activeness' and 'distribution' are offered by the MA design paradigm.

The proposed agent-based monitoring system can efficiently populate a generic networked system with small, lightweight MAs that act as 'area monitors'. The agent system is capable of partitioning the monitored system into sub-systems and assigns MAs to sub-systems. However, the process is not realised in a centralised fashion. Agents progressively inspect the network, create initial partitions and clone agents for the new partitions. The agent deployment process is then 'distributed', a key factor for scalability.

It is also 'active' as opposed to 'static' because the 'area monitors' location is computed on-the-fly, depending on the network conditions detected by the agents themselves, rather than being computed off-line. In addition, because they can sense their network environment, agents can also dynamically re-locate themselves at run time, providing adaptation towards changing network conditions.

The proposed agent system was designed with 'scalability' and 'adaptability' in mind, and was meant to be able to cope with frequently changing conditions of large-scale, dynamic networked systems. The agent system was used as a testbed for the examination of the following research hypothesis:

> *The application of the 'weak agent mobility' paradigm to distributed monitoring represents an effective complement to more conventional 'centralised' and 'static distributed' monitoring approaches. In particular, the agent approach:*
>
> 1. *can lead to significant improvements in performance and scalability;*
> 2. *can be used to realise near-optimal distributed monitoring systems;*
> 3. *can be used to realise distributed monitoring systems which can adapt effectively to changes in the network state;*

A hybrid methodology was adopted to evaluate the research hypothesis. Mathematical modelling was used to study the more theoretical aspects of the work, including the asymptotic complexity of the proposed agent deployment algorithm and its behaviour at transient time; the sufficient conditions for which the algorithm is near-optimal; and a complete (transient time and steady-state) study of the agent system under those near-optimal conditions.

Complementary to the mathematical approach, simulations were carried out to study the agent-based monitoring system under general conditions and for realistic network topologies. The system was quantitatively compared with the more conventional static monitoring approach. Its ability to compute good agent locations for a range of topologies and conditions was also studied. Furthermore, the study of system adaptability was preliminary assessed. The aim was to get a 'feel' of adaptability to justify further work, rather than carrying out a comprehensive, methodical adaptability assessment.

9.2 Discussion of Thesis Contributions

The main contribution of this thesis is the examination of the research hypothesis described above. However, this has involved and required investigation and implementation work that can be seen, *per se,* as an additional contribution to the thesis. Individual contributions resulting from the thesis work are discussed in this section. These contributions may be seen in relation to the research gaps highlighted in Chapter 3.

9.2.1 Active, Distributed Monitoring

Monitoring, information gathering, and information filtering are often regarded as some of the most potential and promising applications of MAs. Moreover, ‘strongly distributed’ management paradigms are seen as the ones that should be referred to when designing future management applications. In particular, future monitoring systems need to be able to cope with scale and dynamics. However, the literature survey reported in Chapter 3 highlights a pathological tendency for ‘centralised’ or ‘weakly distributed’ paradigms.

The active, distributed monitoring approach proposed in this thesis follows the ‘strongly distributed’ paradigm. As such, it can regarded as an initial step towards a better understanding of the benefits of that paradigm, through the investigation of its related issues and the realisation of prototype applications.

9.2.2 Employment of Agent Mobility in Management

The literature survey of Chapter 3 has also highlighted that very few examples of MA-based management (exploiting what we called the ‘real essence’ of agent mobility) have been reported so far. This thesis contributes towards that gap by proposing a system which goes beyond the simpler MbD concept and is developed around agent weak mobility, cloning, autonomy, and reactivity. Pro-activity – *i.e.* the ability of the agent system to anticipate problems thus reacting prior to their detection – is not directly assessed, although its important role for improving performance is discussed. Agent pro-activity could be readily studied with the provided MA simulation infrastructure.

It should be mentioned, though, that the evaluation methodology of this thesis does not cover the whole range of functionality required for a management system. The thesis experimental work is, in fact, focused on monitoring that is only one of the functions of management systems. It should be reminded, however, that monitoring is a fundamental part of management. As such, by improving the effectiveness of the monitoring system, the whole management system will be positively affected.

9.2.3 Quantitative Comparative Performance Evaluation of MA-based Monitoring

Another literature gap concerns methodological and quantitative studies of agent-based management systems. The assessment of management systems is, *per se*, a difficult task. In fact, contrary to more traditional areas such as ‘computer architecture’, the area of management does not have established methodologies and benchmarking applications. The same problem applies to the agent community. Therefore, the thesis, by bringing together the field of management with that of agents, faces the challenge of proposing and following a new assessment method (Chapter 5).

An important contribution is the comparative performance evaluation between ‘static centralised’ monitoring and ‘active distributed’ monitoring. The thesis quantifies performance and scalability improvements for a range of input variables and network conditions.

It may be argued that the significant improvements achieved with the proposed agent system, in comparison with centralised monitoring, were to be expected because of the distributed nature of the proposed system. Moreover, it may be added that alternative ‘static distributed monitoring’ solutions may be adopted, leading to comparable results to the ones of the agent system. However, we should bear in mind that an important requirement of the proposed monitoring system is that it should be able to cope with highly dynamic as well as large-scale networked systems. This, in turn, means that we need a distributed monitoring system in which the location of ‘area monitors’ can be computed in real-time rather than statically. Static distributed monitoring solutions are based on a different assumption: that the location of ‘area managers’ is computed only initially, based on off-line computation of the network topology.

Clearly, network dynamics timescales involved in this case are significantly larger than those applicable to MAs. The location of area managers realised with MAs is computed by the agents themselves during the agent deployment process. It has been shown that this process can be concluded in at most O($R(u)$) seconds, where $R(u)$ is the network radius (Chapter 6). It was also proved that agent locations are at least near-optimal. Consequently, the agent system succeeds in placing area monitors near-optimally provided that the network status and topology vary more slowly than O($R(u)$) seconds. At run time, agents can cope with even more rapid network fluctuations because only individual agent migration is involved in the adaptation process.

Conversely, in the case of ‘static distributed monitoring’ the whole network topology needs to be discovered before the agent location is computed, a process which is impractical for large-scale systems and becomes not viable as network dynamics increases. In practice, the location of static area managers is computed on the bases of an ‘estimated’ rather than ‘measured’ network topology and it is not attempted to keep optimality as the networked system evolves. This approach was meant for situations in which systems rely on relatively static, fixed networks.

9.2.4 Preliminary Study of Adaptable, Self-reconfigurable MA-based Monitoring

Our preliminary study of adaptation and self-reconfiguration is another important contribution. The great interest sparked by topic pressed in the direction of attempting to perform a number of simulations to get at least an indication of the possibilities offered by the agent system.

Adaptability is a unique feature offered by MAs. In static management, adaptation tends to be a sub product of network- and transport- layer protocols. For instance, dynamic routing offers indirect compensation mechanisms in face of link failure or congestion. This is a very limited form of adaptation from the management point of view. Statically distributed area managers are fixed entities, both in location and functionality. Conversely, active monitoring offers the capability of dynamically changing the location of those managers (the monitoring entities in our case). Our initial study suggests that this is a promising approach to deal with continuously changing conditions of dynamic networked systems.

9.2.5 Novel Near-optimal Solution to P-median Problem

The requirements of active distributed monitoring led us into the investigation of the *p-median* problem. This is a classic NP-complete problem when striving for optimality. Upon surveying existing approximate solutions to that problem it was found that none of them suited our requirements. Most solutions were meant for a centralised system; others suffered from high computational complexity; and other did not guarantee near-optimality. We, then, embarked on the study of an efficient solution that could be easily computed in a distributed fashion, would be characterised by low polynomial complexity, and would guarantee near-optimality. One of the products of this thesis is such an algorithm along with its assessment with regard to computational complexity and distance from optimality. We have demonstrated the viability of this algorithm for the solution of the agent location problem.

9.2.6 Extensions to NS simulator for Code Mobility

A practical contribution of this thesis is the extension of NS (a widely used network simulator) with support for code mobility. We have also followed a methodological simulation work which sets an example on possible ways of using the simulation environment for evaluating distributed algorithms, with particular orientation towards systems based on code mobility.

Therefore, the methodology followed for the evaluation and the number of tools and scripts produced to support it may serve as a starting point for other simulation-based work in this area.

9.3 Conclusions

Conclusions that can be drawn from the thesis work about the above research hypothesis are discussed in this section. Detailed conclusions are also included at the end of Chapters 6, 7, and 8.

9.3.1 Performance and Scalability

9.3.1.1 Steady-state System

Performance improvement at steady state represents the key feature of distributed monitoring. This is also the less surprising result which was already expected at the beginning of the work and provided motivations for it. What was not available was a quantification of those benefits. By studying this aspect both theoretically and by simulation we can conclude that the proposed active, distributed monitoring system is a near-optimal solution to the problem of minimising traffic and response time.

Having designed such a system we have proved the first item of our hypothesis: significant improvements are achievable with respect to static monitoring. This is an immediate result for the case of ‘static centralised’ monitoring, as can be observed from the plots of Chapter 8. The actual improvements depend on the particular monitoring task and agent configuration. The parameterisation of the simulation environment allows one to easily change the task and the agent configuration, run the simulator, and achieve comparative figures between centralised and distributed monitoring. This could be used as a tool for the designer who wants to quantify the potential benefits for a particular monitoring system configuration.

Static distributed monitoring will also benefit significantly from this approach because of the increased ability of the agent system to cope with dynamic, large-scale networked systems.

Performance improvements can be directly correlated to the use of agent weak mobility and cloning. We have demonstrated how these are important ingredients for the efficient, dynamic solution of the agent location problem. Clearly, alternative solutions may exist but, to the best of the author’s knowledge, no such alternatives are readily available in the literature.

9.3.1.2 Transient-time System

While the steady-state system will generally outperform the centralised approach and the statically distributed one, there is a price that needs to be paid. This is the cost involved during transient time – *i.e.* during the agent deployment process. At transient time the system is not stable and cannot yet operate. We have demonstrated that agent deployment time is generally completed in O(*R(u)*) seconds. That span of time gives the boundary of applicability of the agent system. Clearly, deployment traffic and delay are not incurred in the centralised approach.

Deployment traffic gives us a second boundary of applicability. It depends on agent size that, in turn, depends on the complexity of the agent functionality. Simple, autonomous agents able to perform local aggregation of monitored data, generate statistics, and trigger alarms, can be implemented in the Java language in modules of size in the order of 10 Kbytes [Knight 99].

In general, monitoring tasks whose overall duration is shorter than the agent deployment time will be more conveniently realised with conventional approaches based on static objects. This is valid if we consider real-time-oriented types of constraints. If the priority is on traffic minimisation then the agent approach may be still be convenient, particularly in the case of data-intensive tasks. Tasks characterised by very high polling rate tend to inject significant traffic, which can be dramatically reduced with distributed area managers.

Another observation about agent deployment overheads should be added. If we compare the agent approach with a distributed approach based on static objects we may say that the latter will involve a lower deployment overhead. In fact, objects tend to be more lightweight than agents. However, the scope of applicability of the agent solution is wider since agent locations are computed at run time; whereas object location needs to be predetermined, which means that is based on ‘estimated’ topologies rather than the real one. This, in turn, translates in locations, which are closer to optimality in the case of agents, with subsequent steady-state traffic and response time reduction.

We can conclude that agent deployment is the major drawback of the agent approach which determines the boundaries of applicability of this approach. However, the deployment overheads become increasingly negligible if compared with the steady-state advantages, as the duration of the monitoring task or the polling rate tend to grow. Deployment overheads are cost-effective also in the case of data-intensive monitoring tasks. In that case, the agent’s particular versatility towards data processing and filtering may lead to significant reduction in both traffic and response time.

9.3.1.3 Overheads

Deployment traffic and time are only two of the overheads of the agent approach. These are also the overheads that could be studied more closely through the adopted evaluation methodology. Other overheads are related to computing resources – *i.e.*, memory and processing overheads. We have not focussed on the precise assessment of such overheads for a number of reasons. This thesis is focused on the evaluation of the benefits achievable from the network point of view. Network resources are assumed to be the critical ones if compared to computational resources. Another reason is that to assess computational overheads we should have integrated a third method, that is the experimental one. We should have realised the agent system on a real MA platform, over a wide area network and we should have measured the resource consumption. This would have shifted the focus towards the assessment of the overheads of MA platforms which is not central to this work. This kind of analysis is typically carried out by those research teams which develop new MA platforms who need to evaluate their platform in comparison with other existing ones.

Therefore, while we can draw quantitative conclusions on agent deployment traffic and delay, we have mainly referred to data available in the literature as far as the other overheads are concerned. Agents will require an MA infrastructure which will, *per se*, consume memory and processing resources. MAs, as well, will add load which will depend critically on the characteristics of the MA environment.

An important conclusion regards the identification of the major limiting factor of the proposed agent system. From our preliminary measurements carried out on existing MA platforms, from data available in the literature, from the theoretical studies of Chapter 6 and 7, and from the simulations presented in Chapter 8, we can conclude that the MA migration overheads are found to be the limiting factor. When agent mobility is utilised, agents end up spending most of their time for serialisation and de-serialisation. The actual agent transmission over the wire tends to be significantly faster than those processes required for preparing the agent for transmission and for execution, respectively. Depending on various implementation aspects, agent serialisation/de-serialisation involves times in the order of hundreds of milliseconds, well above the tens of milliseconds required to transport the code over the wire [Knight 99, Bohoris 00c].

9.3.2 Near-optimality

Near-optimality in correlation to weak mobility is the second point of the hypothesis under examination. The study and assessment of the agent-based monitoring system has served to prove this point. We have seen how near-optimality in terms of traffic or response time depends on agent location near-optimality. By surveying the literature we have identified a gap in efficient, suitable, and viable solutions to the agent location problem. We have then provided a novel near-optimal solution which exploits the fundamental property of weak agent mobility. We have also seen the benefits in terms of efficiency of exploiting agent cloning as well.

We should add that the proposed solution is proved near-optimal with respect to traffic minimisation. The proposed work could be readily modified to prove near-optimality with respect to response time. Solutions that minimise both traffic and response time promise to be significantly more complex and, thus, inefficient.

The simulations of Chapter 8, however, show a significant improvement in response time in the case where the algorithm tries to minimise overall traffic. This is reasonable because response time tends to increase as a result of network congestion. However, this result may not be true in general since traffic and response time minimisation generally require different agent locations. From the simulations of Chapter 8 we can conclude, though, that for the class of networks considered herein – that is for networks resembling the Internet topology – traffic minimisation resulted in significant response time minimisation.

9.3.3 Adaptability

The third point of the research hypothesis regards adaptability and its relation to agent weak mobility. Despite having examined this point only preliminarily rather than through a comprehensive set of simulations, it is already possible to draw some conclusions.

Our initial study indicates that, in addition to location near-optimality achieved at deployment time, it is possible to maintain near-optimality through simple agent migration strategies. Weak mobility adds a new dimension to static distributed monitoring by allowing adaptation to continuously changing network conditions. The degree of freedom introduced by agent mobility is hardly conceivable in well-standardised management architectures and methodologies refined over the years.

The initial results demonstrate the potential of agent mobility in terms of performance and scalability as well as flexibility. The simple ‘link failure’ scenario suggests that the real essence of MAs may have a significant impact on adaptability. Weak mobility, autonomy, reactiveness, pro-activeness, and cloning can be employed to design self-regulating monitoring systems targeted to large-scale, dynamic networked systems.

The relatively high costs associated to agent migration supported by general-purpose MA platforms give also an indication of the timescales over which adaptation might be effective.

When agent migration times are in the order of a second, the agent system is able to compensate to changes within timescales larger than a second. On the other hand, steady-state performance and scalability will be comparable to those typical of systems based on static object technologies provided that effective methods are adopted to place those objects.

9.4 Applicability

The thesis work has considered 'active distributed monitoring' in a general fashion, not binding it to any particular application. It will be applicable to network monitoring, system monitoring, or more generally for monitoring large-scale, dynamic networked systems of any nature. The key feature of scalability makes this approach suitable also for real-time monitoring. By populating the system with a suitable number of agents, it is in fact possible to provide an upper bound on the maximum response time. Since agents are distributed evenly in the networked system, the more agents that are deployed, the closer they will be to their portion of monitored resources, resulting in reduced response time.

Active distributed monitoring can be seen as a fundamental part of more complex management or control systems. It can be used for efficient information gathering, for collecting and processing raw information, or to create different views of data-intensive processes. More generally, all of the four fundamental monitoring activities suggested by Sloman can benefit from the proposed system [Sloman 94]. These have been discussed in Chapter 2 and are: 1) generation; 2) processing; 3) dissemination; and 4) presentation.

The agent location algorithm can be used beyond monitoring problems. We have seen the importance and applications of location problems in Chapters 2 and 3. Our solution falls in the category of heuristic, provably near-optimal solutions with the additional advantage of being characterised by low computational complexity. It can be applied in problems involving the optimal location of p servers in a network of N nodes with information exchanges required between them and $p<<N$.

We also discussed another important advantage, which is the possibility of computing the algorithm in real-time and in a distributed fashion. Applications of this algorithm are then numerous, from classic location theoretic problems to typical server or service location problems. More generally, this algorithm can be modified or extended to suit particular requirement of MA systems.

At a more conceptual level, several of the ideas developed in the context of this thesis are already being conveyed to other research project in which the author is involved as a coordinator. Agents for dynamic Quality of Service (QoS) monitoring are being studied in the context of the IST *Virtual Home Environment for Service PErsonalization and Roaming Users* project [VESPER]. Finally, adaptability through agent migration is being investigated in the context of the *Virtual Centre of Excellence in Mobile & Personal Communications* [MVCE].

9.5 Future Directions

Active distributed monitoring opens several new avenues of research. In this section, we discuss some of them. We focus on additional work that would enhance and complement the thesis. The next subsections are not necessarily presented in order of importance.

9.5.1 Experimentation with Active Distributed Monitoring

One necessary step towards the realisation of active distributed monitoring is its implementation and experimentation on prototype networks or on a real networked system. Real measurements will enlighten the actual behaviour of the proposed approach with regard to overheads, stability, and complexity.

Measurements on the actual agent deployment time as a function of various scale factors will strengthen even further the motivation for integrating agents in monitoring systems. A better understanding of processing and memory overheads will provide essential information to the designer of such systems. Our preliminary comparative measurements among Grasshopper (a general purpose MA platform), CORBA (as a distributed static object infrastructure), and CodeShell (an simple prototype of an MA platform optimised for constrained mobility) indicate that general purpose MA platforms do add significant overheads. The CodeShell measurements, however, suggest that there is a large margin of improvement.

9.5.2 Simulation Work and Experimentation on Adaptation

An initial study of adaptation has been presented in the thesis. Preliminary results suggest that agent migration is an important mechanism towards adaptable monitoring systems. It would be very interesting to pursue that study in a methodical fashion along the lines of the method described in Chapter 5. Extensive simulations on adaptation will need to be carried out to evaluate the ability of the system to self-reconfigure itself in face of a variety of network conditions, for a range of network topologies, and for different input parameters.

It is expected that agent migration will be the major limiting factor. However, various intelligent migration techniques may be able to reduce this problem. For instance, the latency involved in agent migration may be reduced by adopting appropriate cloning strategies. An agent will in that case clone itself, send the clone and continue its operation until the clone is up and running at the destination. At which time the new clone will take over and kill its originator.

A number of new stimulating research problems are related to this one. One is the instability problem. For instance, migration if triggered improperly may lead to an oscillating agent system where agents keep migrating among a cluster of nodes. Pro-activity may even aggravate the problem. Instability problems are well known in other disciplines such as mechanics and various measures can be taken to reduce them. Hence, the application of theories developed in mechanical engineering to our agent system may uncover interesting findings.

Agent control mechanisms need further study as well. By adding inertia to the system we might achieve stability at the expense of delay. We also need to realise mechanisms to control agent population.

The author of this thesis finds the above research topics very promising as well as stimulating and is currently writing a project proposal to further this work under the sponsorship of the Engineering & Physical Science Research Council (EPSRC).

9.5.3 Exploration of MA-based Management

The initial aim of the thesis was to investigate MA-based management, though the scope of the evaluation was then constrained to the area of distributed monitoring. The natural evolution of the thesis work would, thus, be the evaluation of other management functional areas, which have the potential of benefiting from agent weak mobility.

9.5.4 Exploration of Location Algorithm for other Classes of Network

The agent location algorithm was simulated for the class of network topology resembling an Internet network. The assumption was that networked systems increasingly rely on the Internet. The investigation of the properties of the algorithm for other classes of network would strengthen its applicability. The author of this thesis initially planned to repeat all the simulation work for arbitrary network topologies. The relatively long simulation time suggested to avoid pursuing this road, which was not of central importance, given the assumptions made on networked systems.

9.5.5 Modification of the Location Algorithm aiming at Response Time Minimisation

The agent location algorithm has been optimised in order to minimise the overall incurred steady-state monitoring traffic. The simulations of Chapter 8 demonstrate that near-minimal traffic is achieved as well as significant improvement in response time. This is valid for Internet-like topologies whereas it might not be true for a different class of topology. It would be interesting, then, to extend the simulations by considering arbitrary topologies in order to generalise this study.

Another step could involve the modification of the algorithm aiming at minimising response time rather than traffic. One should, then, verify the near-optimality of the new algorithm and study its effects on the overall incurred monitoring traffic.

A more challenging work would be the development of an algorithm that minimises both traffic and response time. This would have more ‘theoretical’ than ‘practical’ implications if we can demonstrate that, for typical networks, the minimisation of one factor leads to significant improvements in the other.

9.5.6 Integration and Interoperability

Having demonstrated the feasibility and advantages of MAs for distributed monitoring as well as their limitations, we have highlighted that there are conditions in which agents cannot compete with the simplicity and efficiency of conventional centralised approaches – *e.g.* in very short, or very simple monitoring operations. Therefore, the ideal solution seems to be the integration of code mobility into existing systems (in order to bring its benefits) rather than relying completely on MA approaches. A seamless integration between existing and MA-based approaches is a topic which deserves further investigation.

Another important investigation may look at agent standardisation and interoperability with existing management framework, such as OSI and SNMP management.

9.5.7 Viability Study in Perspective

The proposed active distributed monitoring system addresses two important requirements of next generation networks, namely scalability and high-dynamics. However, several issues need to be addressed before the deployment of our system to real networks can be considered. The future generation of fixed and mobile integrated networks is following the path of openness, aimed at facilitating the separation between network operator, service provider, and value-added service provider. The level of openness of future networks, though, is not clear at the moment.

Openness cannot be realised without a sufficient level of security and, unfortunately, MAs are still perceived as a constant threat to it. The investigation of secure and safe agent environments is, then, of crucial importance [Vigna 98]. Without security mechanisms, the proposed agent system will only be applicable to systems relying on corporate worldwide intranets. Networked systems spanning different organisations will tend to prevent the execution of foreign MAs within their boundaries, which is a major impediment for our approach. In fact, we assume that MAs are free to roam the network and access routing information, which is not a safe assumption in the current state but is a reasonable possibility for the future.

Safety is another important element that needs further study. Even when the security concerns will be defeated we shall still need to be able to control the agent system in order to prevent it from abusing of crucial network and computing resources. A safe MA environment will make sure that MAs are executed only ‘where’ and ‘if’ a sufficient amount of resources is available. The ability to limit the resources consumed by MAs is fundamental for the scenario in which network or service providers will be willing to sustain the load of foreign MAs.

References

[Abdu 99] H. Abdu, H. Lutfiyya, M. Bauer, *A Model for Adaptive Monitoring Configurations*. In Proc. of IEEE IM'99, (1999).

[Anastasi 00] G. Anastasi, A. La Corte, A. Puliafito, and O. Tomarchio, *An agent-based approach for QoS provisioning to mobile users in the Internet.* Proc. of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI2000), Orlando Florida, USA, (July 2000).

[Appleby 94] S. Appleby, S. Steward, *Mobile Software agents for control in telecommunications networks*. BT Technology Journal, pp104-13, (April 1994).

[Aridor 98] Yariv Aridor and Danny B. Lange, *Agent Design Patterns: Elements of Agent Applications Design.* Second International Conference on Autonomous Agents (Agents '98), (May 1998).

[Baldi 97] M. Baldi, S. Gai, G. P. Picco, *Exploiting Code Mobility in Decentralized and Flexible Network Management*, Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, (April 1997).

[Baldi 98] M. Baldi, G. P. Picco, *Evaluating the Tradeoffs of Mobile Code Paradigms in Network Management Applications*, ACM Transactions on Software Engineering and Methodology, 20th International Conference on Software Engineering (ICSE '98), Kyoto, Japan, (April 1998).

[Barr 93] W.J. Barr, T. Boyd, Y. Inoue, *The TINA Initiative.* IEEE Communications Magazine, Vol 31(3), pp.70-76, (1993).

[Baumann 99] J. Baumann, *Control Algorithms for Mobile Agents*. PhD Thesis, University of Stuttgart, (1999).

[Bieszczad 98c] A. Bieszczad, B. Pagurek, T. White, *Mobile Agents for Network Management*, IEEE Communications Surveys, Fourth Quarter 1998, vol. 1, no. 1, p. 2-9, (1998).

[Birrell 84] A. Birrell, B. J. Nelson, *Implementing Remote Procedure Calls*. ACM Transactions on Computer Systems, Vol.2, pp.39-59, (February 1984).

[Bivens 99] A. Bivens, L. Gao, M.F. Hulber, B.K. Szymanski, *Agent-based Network Monitoring*. In Proc. of Agent Based High Performance Computing and Autonomous Agents, Seattle, Washington, USA, (May 1999).

[Bohoris 00a] C. Bohoris, G. Pavlou, H. Cruickshank, *Using Mobile Agents for Network Performance Management*, Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS '00), Hawaii, USA, J. Hong, R. Weihmayer, eds., pp. 637-652, IEEE, (April 2000).

[Bohoris 00b] C. Bohoris, **A. Liotta**, G. Pavlou, *Software Agent Constrained Mobility for Network Performance Monitoring*, Proc. of the 6th IFIP Conference on Intelligence in Networks (SmartNet 2000), Vienna, Austria, ed. H.R. van As, pp. 367-387, Kluwer, (September 2000).

[Bohoris 00c] C. Bohoris, **A. Liotta**, G. Pavlou, *Evaluation of Constrained Mobility for Programmability in Network Management*, To appear in the proceedings of the 11th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2000), Austin, Texas, USA, (December 2000).

[Buckley 90] F. Buckley , F. Harary, *Distance in Graphs*. Addison-Wesley, (1990).

[Busuioc 94a] M. Busuioc, D. Griffiths, *Cooperating intelligent agents for service management in communications networks*. CKBS-SIG Proceedings 1993. 1993 Proceedings of the Special Interest Group on Cooperating Knowledge Based Systems. Selected Papers from the Workshop, pp213-26, (1994).

[Busuioc 94b] M. Busuioc, *Distributed cooperative agents for service management in communications networks*. EE Eleventh UK Teletraffic Symposium. Performance Engineering in Telecommunication Networks , pp24/1-7, (1994).

[Calvert 97] K.L. Calvert, M.B. Doar, E.W. Zegura, *Modeling Internet Topology*, IEEE Communications Magazine, (June 1997).

[Cardelli 95] L. Cardelli, *A Language with Distributed Scope*. Computing Systems, Vol.8(1), pp. 27-59, (1995)

[Carzaniga 97] A. Carzaniga, G. P. Picco, G. Vigna, *Designing Distributed Applications with Mobile Code Paradigms*. Proceedings of the 19th International Conference on Software Engineering (ICSE'97). p. 22-32, (May 1997).

[Casavant 94] Casavant, T.L., Singhal, M., *Readings in Distributed Computing Systems*. IEEE Computer Society Press, (1994).

[Cejtin 95] H. Cejtin, S. Jagannathan, R. Kelsey, *Higher-order Distributed Objects*. ACM Transactions on Programming Languages and Systems, Vol.17(5), (September 1995)

[Cheikhrouhou 98] M.M. Cheikhrouhou, P. Conti, J. Labetoulle, *Intelligent Agents in Network Management: A State of the Art*. Networking and Information Systems Journal, (June 1998).

[Chess 97] D. Chess, C. Harrison, A. Kershenbaum, *Mobile Agents: Are they a Good Idea?* Proc. of Mobile Object Systems, Towards the Programmable Internet. J. Vitek, C. Tschudin, editors, Springer, pp.25-47, (1997).

[Chiariglione 98] L. Chiariglione, *Foundations for Intelligent Physical Agents.* FIPA 98 Draft Specification, part 11, http://drogo.cselt.it/fipa/spec/fipa98/fipa98.html, (August 17, 1998).

[Cornuejols 77] G. Cornuejols, M.L. Fisher, G.L. Nemhauser, Location of Bank Accounts to Optimize Float: An Analytical Study of Exact and Approximate Algorithms. Management Science, Vol.23, 789-810, (1977).

[Cugola 96] G. Cugola, C. Ghezzi, G.P. Picco, and G. Vigna, *A Characterization of Mobility and State Distribution in Mobile Code Languages*. In Proceedings of the Second Workshop on Mobile Object Systems, Linz, Austria, (July 1996)

[Cugola 97] G. Cugola, C. Ghezzi, G.P. Vigna, *Analysing Mobile Code Languages*. in Mobile Object Systems: Towards the Programmable Internet, Springer-Verlag, LNCS, (April 1997).

[Cypser 91] R. J. Cypser, *Communications for Co-operating Systems*. Addison Wesley, pp.244-245, (1991).

[Daskin 95] M. S. Daskin, *Network and Discrete Location*. Wiley, (1995).

[deMeer 00] H. de Meer, A. La Corte, A. Puliafito, O. Tomarchio, *Programmable Agents for Flexible QoS Management in IP Networks*. IEEE Journal on Selected Areas in Communication, Vol.18, N.2, (February 2000).

[deMeer 98a] H. de Meer, A. Puliafito, O. Tomarchio, *Management of QoS with Software Agents*. Cybernetics and Systems: an International Journal, Vol.27, N.5, (1998).

[deMeer 98b] H. de Meer, A. Puliafito, J.P. Richter, O. Tomarchio, *Tunnel Agents for Enhanced Internet QoS*. IEEE Concurrency, Vol.6, N.2, pp.30-39, (April-June 1998).

[Di Caro 98] G. Di Caro, M. Dorigo, *AntNet: Distributed Stigmergetic Control for Communications Networks*. Journal of Artificial Intelligence Research (JAIR), Vol.9, pp.317-365, (1998).

[Di Marzo 95] G. Di Marzo, M. Muhugusa, C. Tschudin, J. Harms, *The Messenger Paradigm and its Implications on Distributed Systems*. In Proc. of the ICC’95 Workshop on Intelligent Computer Communications, (1995).

[Dikaiakos 00] M.D. Dikaiakos, G. Samaras, *Quantitative Performance Analysis of Mobile Agent Systems: a Hierarchical Approach.* Technical Report TR-00-2, Department of Computer Science, University of Cyprus, (June 2000).

[Dini 97] P. Dini, G. Bochmann, T. Koch, B. Kramer, *Agent based Management of Distributed Systems with Variable Polling Frequency Policies*. INM'97, (1997).

[El-Darieby 98] M. El-Darieby, *Intelligent Mobile Agents for Network Fault Management*, Technical Report SCE-98-13, System and Computer Engineering, Carleton University, (1998).

[El-Shaieb 73] A.M., El-Shaieb, *A New Algorithm for Locating Sources Among Destinations*. Management Science, Vol.20, pp.221-231, (1973).

[Evans 86] D. Evans, *Supervisory Management: Principles and Practice*. 2nd edition, Cassell Educational Ltd, London, UK, (1986).

[Evans 92] J.R. Evans, E. Minieka, *Optimization Algorithms for Networks and Graphs*. Marcel Dekker, Inc., (1992).

[Fall 99] K. Fall, K. Varadhan, *NS Notes and Documents*. UC Berkeley, October 1999 (http:// www.isi.edu/ ~salehi/ ns_doc/).

[FIPA 98] FIPA 97 Specification, Version 2.0, Part 2, Agent Communication Language, (October 1998).

[FIPA] Foundation for Intelligent Physical Agents, web page: http://www.fipa.org/

[Fisher 75] M.L. Fisher, W.D. Northup, J.F. Shapiro, *Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience*. Mathematical Programming Study, Vol.3, pp.56-94, (1975).

[Franklin 96] S. Franklin, A. Graesser, *Is it an Agent or just a Program? A Taxonomy for Autonomous Agents*, In J.P. Muller, M.J. Wooldridge, N.R. Jennings (Eds), *Intelligent Agents III*, Proc. ECAI'96 Workshop (ATAL), Budapest, Hungary, August 1996. LNAI 1193, pp.21-35, Springer-Verlag, Berlin, Germany, (1997).

[Fuggetta 97] A. Fuggetta, G. P. Picco, G. Vigna, *Understanding Code Mobility*, IEEE Transactions on Software Engineering, vol. 24, no. 5, pp. 342-361, (1998).

[Gagnon 93] F. Gagnon, J-CH Gregoire, Implementation of Delegation in Distributed Network Administration. CCECE 93, (1993)

[Garey 79] M. R. Garey, D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, (1979).

[Garfinkel 74] R.S. Garfinkel, A.W. Neebe, M.R. Rao, *An Algorithm for the m-median Plant Location Problem*. Transportation Science, Vol.8, pp.217-236, (1974).

[Gavalas 00a] D. Gavalas, D. Greenwood, M. Ghanbari, M. O’Mahony, *Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology*, Computer Communications Journal, special issue on Mobile Agents for Telecommunication Applications, Vol. 23, No.8, pp. 720-730, (April 2000).

[Gavalas 00b] D. Gavalas, D. Greenwood, M. Ghanbari, M. O’Mahony, *Deploying a Hierarchical Management Framework using Mobile Agent Technology*. Proc. of 7th International Conference on Intelligence in Services and Networks (IS&N’00), LNCS, Vol.1774, pp.333-348, Springer-Verlag, Athens, Greece, (23-25 February 2000).

[Gavalas 00c] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *Enabling Mobile Agent Technology for Intelligent Bulk Management Data Filtering*. Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS’00), pp. 623-636, Honolulu, USA, (10-14 April 2000).

[Gavalas 00d] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *Implementing a Highly Scalable and Adaptive Agent-based Management Framework*. Proc. of the IEEE Global Communications Conference (GLOBECOM’00), Vol.3, pp.1458-1462, San Francisco, USA, (27 Nov – 1 December 2000).

[Gavalas 01a] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *Mobile Software Agents for Decentralised Network & System Management*. In Elevier Microprocessors and Microsystems, special issue on “Mobile Agent Technology: from first proposals to current evolutions”. In press.

[Gavalas 01b] D. Gavalas, *Mobile Software Agents for Network Monitoring and Performance Management*. PhD Thesis, University of Essex, UK, (2001).

[Gavalas 99a] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *Complimentary Polling Modes for Network Performance Management Employing Mobile Agents*, Proceedings of the IEEE Global Communications Conference (Globecom'99), pp. 401-405, Rio de Janeiro, Brazil, (5-9 December 1999).

[Gavalas 99b] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *A Hybrid Centralised - Distributed Network Management Architecture*, Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99), pp. 434-441, (July 1999).

[Gavalas 99c] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology*", Proceedings of the IEEE International Conference on Communications (ICC'99), pp. 1362-1366, (June 1999).

[Gavalas 99d] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *Using Mobile Agents for Distributed Network Performance Management*. Proc. of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA’99), LNCS Vol.1699, pp.96-112, Springer-Verlag, Stockholm, Sweden, (9-11 August 1999).

[Geoffrion 74] A.M. Geoffrion, *Lagrangian Relaxation for Integer Programming*. Mathematical Programming Study, Vol.2, pp.82-114, (1974).

[Ghezzi 97] C. Ghezzi and G. Vigna, *Mobile Code Paradigms and Technologies: A Case Study*. In Proceeding of the First International Workshop on Mobile Agents ‘97, Berlin, Germany, (April 1997).

[Goldman 71] A.J. Goldman, *Optimal Center Location in Simple Networks,* in Transportation Science, Vol.5 (1971), pp. 212-221, (1971).

[Goldszmidt 93] G. Goldszmidt, *Distributed System Management via Elastic Servers*. In Proceedings of the IEEE First International Workshop on System Management, Los Angeles, California, (April 1993).

[Goldszmidt 95a] G. Goldszmidt, Y. Yemini, *Distributed Management by Delegation*. Proceedings of the 15th International Conference on Distributed Computing Systems, (June 1995).

[Goldszmidt 95b] G. Goldszmidt, Y. Yemini, *Evaluating Management Decisions via Delegation*. Integrated Network Management IV. New York: Chapman & Hall, (1995).

[Goldszmidt 96a] G. Goldszmidt, *Computing MIB Views via Delegated Agents*. In Proc. of the 6th CAS conf., Toronto, Canada, (November 1996).

[Goldszmidt 96b] G. Goldszmidt, *Distributed Management by Delegation*. PhD Thesis, Columbia University, New York, (1996).

[Goldszmidt 96c] G. Goldszmidt, Y. Yemini, *Delegated Agents for Distributed System Management*. IFIP/IEEE DSOM 1996 Workshop. L’Aquila, Italy, (October 28-30th 1996).

[Goldszmidt 98] G. Goldszmidt, Y. Yemini, *Delegated Agents for Network Management*. IEEE Communications Magazine, Vol.36 No.3, (March 1998).

[Gray 97] R.S. Gray, *Agent Tcl: A Flexible and Secure Mobile-agent System*. PhD Thesis, Dartmouth College, Hanover, New Hampshire, (June 1997).

[Gregoire 93a] J-CH Gregoire, Delegation: Uniformity in Heterogeneous Distributed Administration. USENIX, San Diego, CA, (January 25-29, 1993).

[Gregoire 93b] J-CH Gregoire, *Management Using Delegation*. Advanced Information Processing Techniques for LAN and MAN Management. Amsterdam: North Holland, (1993).

[Gregoire 95] J-CH Gregoire, *Models and Support Mechanisms for Distributed Management*. Integrated Network Management IV. New York: Chapman & Hall, (1995).

[Grimes 96] G. Grimes, *Intelligent Agents for Network Fault and Performance Management*. Thesis, University of Limerick, (1996).

[GT-ITM] Source code of GT-ITM, available as http:// www.cc.gatech.edu/ projects/ gtitm/.

[Guedes 98] L.A. Guedes, P.C. Oliveira, L.F. Faina, E. Cardozo, *An Agent-based Approach for Supporting Quality of Service in distributed Multimedia Systems*. Elsevier Computer Communications, Vol.21, pp.1269-1278, (1998).

[Guiagoussou 01] M. Guiagoussou, R. Boutaba, M. Kadoch, *Java Implementations of Systems under Test for Advanced Fault Management*. In Proc. of IEEE IM'01, Seattle, USA, (May 2001).

[Hakimi 64] S. L. Hakimi, *Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph*. Operations Research Vol. 12, pp. 450-459, (1964).

[Hakimi 65] S. L. Hakimi, Optimum Distribution of Switching Centers in a Communications Network and Some Related Graph Theoretic Problems. Operations Research Vol. 13, pp. 462-475, (1965).

[Halls 97] D. A. Halls, *Applying Mobile Code to Distributed Systems*. PhD thesis, University of Cambridge, (1997).

[Handler 79] G.Y. Handler, P.B. Mirchandani, *Location on Networks Theory and Algorithms*, MIT Press, (1979).

[Hanle 98] C.Hanle, M.Hofmann, *Performance Comparison of Reliable Multicast Protocols using the Network Simulator ns-2*. In Proc. of the Annual Conference on Local Computer Networks (LCN), Boston, MA, USA, (October 11-14, 1998).

[Harrison 95] C.G Harrison., D.M. Chess, A. Kershenbaum, *Mobile Agents: Are they a good idea?* Technical Report, IBM Research Division. Watson Research Center, (March 1995).

[Hegering 98] H.G. Hegering, S. Abeck, B. Neumair, *Integrated Network Management of Networked Systems*. Morgan Kaufmann Publishers, (1998).

[Held 74] M. Held, P. Wolfe, H.P. Crowder, *Validation of Subgradient Optimisation*. Mathematical Programming, Vol.6, pp.62-88, (1974).

[Hohol 97] F. Hohol, P. Klar, J. Baumann, *Efficient Code Migration for Modular Mobile Agents*. In 3rd ECOOP Workshop on Mobile Object Systems: Operating System Support for Mobile Object Systems (MOS'97), (1997).

[IBM 99] IBM Tokyo Research Laboratory, *Aglets Workbench: Programming Mobile Agents in Java*. URL: http://www.trl.ibm.co.jp/aglets (1999).

[Ismail 99] L. Ismail, D. Hagimont, *A Performance Evaluation of the Mobile Agent Paradigm*. In Proc. of OOPSLA'99, Int. Conf. on Object-Oriented Programming, Systems and Applications, Denver, (1-5 November 1999).

[ISO-10164 98] ISO, Information Technology – Open Systems Interconnection – Systems Management – Command Sequencer. International Standard ISO 10164-21, ISO (1998).

[ISO-WG4 95] ISO WG4 N1851, *Open Distributed Management Architecture*. Working Draft 3, ISO, (July 1995).

[Jain 91] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons Inc., (1991).

[Jarvinen 72] P. Jarvinen, I. Rajala, H. Sinervo, *A Branch and Bound Algorithm for Seeking the p-median*. Operations Research, Vol.20, pp.173-178, (1972).

[Java 95] Sun Microsystems, *The Java Language Specification*, http://hava.sun.com/docs/books/jls/index.html (October 1995).

[JDK-OS] Sun Microsystems, *Java Object Serilisation*, http://java.sun.com/products/jdk/1.1/docs/guide/serialization/

[JDK-RMI] Sun Microsystems, *Java Remote Method Invocation*, http://java.sun.com/products/jdk/rmi/

[JDMK 98] Sun Microsystem, *Java Dynamic Management Kit*, February (1998).

[JMAPI 96] Sunsoft, *Java Management API Architecture*. Revision A, (September 1996).

[Johansen 95a] D. Johansen, R. van Renesse, F. B. Schneider, *An Introduction to the TACOMA Distributed System – Version 1.0*, Technical Report 95-23, University of Tromso and Cornell University, (June 1995).

[Johansen 95b] D. Johansen, R. van Renesse, F. B. Schneider, *Operating System Support for Mobile Agents*. Proc. of the 5th IEEE Workshop on Hot Topics in Operating Systems, pp. 42-45, (1995).

[Joyce 87] Joyce, J., Lomow, G., Slind, K., Unger, B., *Monitoring Distributed Systems*. ACM Trans. Comput. Syst., 5(2), 121-50, (1987).

[Kahani 97] M. Kahani, H.W.P. Beadle, *Decentralised Approaches for Network Management*. Computer Communications Review, ACM SIGCOMM, Vol. 27 N.3, (July 1997).

[Kariv 79] O. Kariv, S.L. Hakimi, An Algorithmic Approach to Network Location Problems - Part 2: The P-medians, SIAM J. Appl. Math., Vol.37, pp. 539-560, (1979).

[Keller 96] A. Keller, *Service-based Systems Management: Using CORBA as a Middleware for Intelligent Agents*. In Proc. of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, (October 1996).

[Khumawala 72] B.M Khumawala, An Efficient Branch and Bound Algorithm for the Warehouse Location Algorithm. Management Science, Vol.18, pp.B718-B731, (1972).

[Knight 99] G. Knight, R. Hazemi, *Mobile Agent based management in the INSERT project*, Journal of Network and System Management (Mobile Agent-based Network and Service Management), Vol. 7 (3), (September 1999).

[Kooijman 95] R. Kooijman, Divide and Conquer in Network Management using Event-driven Network Area Agents. Technical University of Delft, The Netherlands. (May 1995).

[Kramer 99] K.H. Kramer, N. Minar, P. Maes, *Mobile Software Agents for Dynamic Routing*. Mobile Computing and Communication Review, Vol.3, N.2, (March 1999).

[Lange 98] D. B. Lange, *Mobile Objects and Mobile Agents: The Future of Distributed Computing?* , Proceedings of the European Conference on Object-Oriented Programming (ECOOP’98), (1998).

[Lange 99] D. B. Lange, M. Oshima, *Seven Good Reasons for Mobile Agents*. Communications of the ACM, Vol.42(3), pp.88-89, (March 1999).

[Lea 93] R. Lea, C. Jacquemont, E. Pillevesse, *Cool: System Support for Distributed Object-oriented Programming.* Communications of the ACM, Vol.36(9), pp.37-46, (November 93).

[Leckie 97] C. Leckie, R. Senjen, B. Ward, M. Zhao, *Communication and Coordination for Intelligent Fault Diagnosis Agents*. In Proc. of the IFIP/IEEE International Workshop on Distributed Systems Operations & Management (DSOM’97), Sydney, Australia, (October 21-23, 1997).

[Lee 98] L.C. Lee, H.S. Nwana, D.T. Ndumu, P. de Wilde, *The Stability, Scalability and Performance of Multi-Agent Systems*. BT Technology Journal, Vol. 16, N.3, pp.69-78, (July 1998).

[Leinwand 96] A. Leinwand, K. F. Conroy, *Network Management, a Practical Perspective*. Addison-Wesley, (1996).

[Levi 96] D.B. Levi, J. Schonwalder, *Script MIB. Definition of Managed Objects for the Delegation of Management Scripts*. IETF Internet Draft, 1st version, (November 1996).

[Levi 99] D. Levi, J. Schonwalder, RFC2592 – *Definitions of Managed Objects for the Delegation of Management Scripts*. The Internet Society, (May 1999).

[Lewis 97] L. Lewis, U. Datta, *Intelligent Agents for Distributed Configuration Management*. In Proc. of the IFIP/IEEE International Workshop on Distributed Systems Operations & Management (DSOM'97), Sydney, Australia, (October 21-23, 1997).

[Lewis 99] J.P. Lewis, A. Trail, *Statistics Explained*. Addison-Wesley, (1999).

[Liotta 01a] **A. Liotta**, G. Pavlou, G. Knight, *Active Distributed Monitoring for Dynamic Large-scale Networks*, Proceedings of the IEEE International Conference on Communications (ICC'01), Helsinki, Finland, IEEE, (June 2001).

[Liotta 01b] **A. Liotta**, G. Pavlou, G. Knight, *Reducing the Cost of Large-Scale Network Monitoring with Mobile Code*, submitted to IEEE Network.

[Liotta 01c] **A. Liotta**, G. Pavlou, G. Knight, *A Self-adaptable Agent System for Efficient Information Gathering*, Proceedings of the 3rd International Workshop on Mobile Agents for Telecommunication Applications (MATA'01), Montreal, Canada, Springer-Verlag (August 2001).

[Liotta 98a] **A. Liotta**, G. Knight, G. Pavlou, *Modelling Network and System Monitoring Over the Internet Using Mobile Agents*, Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS '98), New Orleans, USA, Vol. 2, pp. 300-312, (February 1998).

[Liotta 98b] **A. Liotta**, G. Knight, *Decomposition Patterns for Mobile Code-based Management*. In proc. of HP-OVUA, The Hewlett-Packard Openview University Association Plenary Workshop 1998, ENST de Bretagne, Rennes, France, (April 19-21, 1998).

[Liotta 99a] **A. Liotta**, G. Knight, G. Pavlou, *On the Efficiency of Decentralised Monitoring using Mobile Agents.* In proc. of HP-OVUA, The Hewlett-Packard Openview University Association Plenary Workshop 1999, Bologna, Italy, (June 13-15, 1999).

[Liotta 99b] **A. Liotta**, G. Knight, G. Pavlou, *On the Performance and Scalability of Decentralised Monitoring Using Mobile Agents*, Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations Management (DSOM'99), (October 1999).

[Liotta 99c] **A. Liotta**, G. Knight, G. Pavlou, *A Simulation-based Assessment of Information Gathering Systems based on Mobile Agents*. In proc. of Simulation'99, London, UK, (October 29, 1999).

[Lipperts 00] S. Lipperts, *How to Efficiently Deploy Mobile Agents for an Integrated Management*. In Proc. of 3rd IFIP International Conference on Trends Towards a Universal Service Market, Munich, Germany. In Lecture Notes in Computer Science, Springer-Verlag, (September 2000).

[Lopes 00] R. P. Lopes, J. L. Oliveira, *On the Use of Mobility in Distributed Network Management*. Proc. of the 33rd Hawaii International Conference on System Sciences (HICSS-33), (January 2000).

[M3010 91] CCITT Rec. M3010 1991, Principles for a Telecommunications Management Network (TMN), (1991).

[Magedanz 95] T. Magedanz, *On the impacts of Intelligent Agent Concepts on Future Telecommunication Environments*. In Proceedings of the 3rd International Conference on Intelligence in Broadband Services and Networks IS&N 1995, Heraklion, Crete, Greece, (October 16-20, 1995).

[Magedanz 96b] T. Magedanz *et al*, *Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?* In Proc. of IEEE INFOCOM, San Francisco, California, USA. (March 24-28, 1996).

[Martin-Flatin 00] Jean-Philippe Martin-Flatin, S. Znaty, Two Taxonomies of Distributed Network and System Management Paradigms. In Emerging Trends and Challenges in Network Management, S. Erfani and P. Ray (Eds.), Plenum Publishers, (2000).

[Martin-Flatin 97a] Jean-Philippe Martin-Flatin, *A Survey of Distributed Enterprise Network and Systems Management Paradigms*. Submitted to JNSM, Special Issue on Enterprise Network and Systems Management, (November 30, 1997).

[Martin-Flatin 97b] Jean-Philippe Martin-Flatin, S. Znaty, *Annotated Typology of Distributed Network Management Paradigms*. Proceedings of DSOM‘97, Sydney, Australia, (21-23 October 1997).

[MASIF 97] Object Management Group, Mobile Agent System Interoperability Facilities Specification, orbos/97-10-05, 1997, ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf (1997).

[Matthes 95] F. Matthes, G. Schroder, J.W. Schmidt, *Tycoon: a Scalable and Interoperable Persistent System Environment*. In M. P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag, p.25, (1995).

[Matula 76] D.W. Matula, R. Kolde, *Efficient Multi-Median Location in Acyclic Networks*, ORSA/TIMS Bulletin, No.2, (1976).

[MCB] Mobile Code Bibliography. http:// www. cnri. reston. va.us/ home/koe/bib/

[McCloghrie 91] K. McCloghrie, M. Rose, RFC1213 – Management Information Base for Network Management of TCP/IP-based internets: MIB-II, The Internet Society, (March 1991).

[McCloghrie 94] K. McCloghrie, F. Kastenholz, *RFC1573 – Evolution of the Interfaces Group of MIB-II*, The Internet Society, (January 1994).

[Meyer 95] K. Meyer, M. Erlinger, J. Betser, C. Sunshine, G. Goldszmidt, Y. Yemini, *Decentralising Control and Intelligence in Network Management*. Proceedings of the 4th International Symposium on Integrated Network Management, Santa Barbara, CA, (May 1995).

[Milojicic 99] D. Milojicic, *Mobile Agent Applications*. IEEE Concurrency, pp.80-90, (September 99).

[Mountzia 96] M.A. Mountzia, G. Dreo-Rodosek, *Delegation of Functionality: Aspects and Requirements on Management Architectures*. Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM’06), (October 1996).

[Mountzia 97a] M.A. Mountzia, G. Dreo-Rodosek, *Using the Concept of Intelligent Agents in Fault Management of Distributed Services*. Journal of Network and Systems Management, (1997).

[Mountzia 97b] M.A. Mountzia, D. Benech, *Communication Requirements and Technologies for Multi-Agent Management Systems*. In Proc. of the IFIP/IEEE International Workshop on Distributed Systems Operations & Management (DSOM'97), Sydney, Australia, (October 21-23, 1997).

[Mountzia 98] M.A. Mountzia, *A Distributed Management Approach Based on Flexible Agents*. Interoperable Communication Networks, Baltzer Science Publishers, Volume I/I, (January 1998).

[Mullins 89] L.J. Mullins, *Management and Organisational Behaviour*. 2nd edition, Pitman, London, UK, (1989).

[MVCE] *Virtual Centre of Excellence in Mobile & Personal Communications*, (http://www.mobilevce.co.uk/index2.htm)

[NAM] J. Mehringer, *The NAM network animator* http://www.isi.edu/nsnam/nam/index.html

[Narula 77] S.C. Narula, U.I. Ogbu, H.M. Samuelsson, *An Algorithm for the p-Median Problem*. Operations Research, Vol.25, pp.709-712, (1977).

[NS] UCB/LBNL/VINT, NS Network Simulator version 2. (http://www-mash.cs.berkeley.edu/ns).

[Oliveira 99] J.L. Oliveira, R. P. Lopes, *Distributed Management Based on Mobile Agents*. Proc. of MATA 99, (1999).

[Oppliger 99] R. Oppliger, *Security Issues Related to Mobile Code and Agent-based Systems*. Computer Communications, Vol.22, pp.1165-1170, Elsevier, (1999).

[Pagurek 98a] B. Pagurek, Y. Li, A. Bieszczad, G. Susilo, *Network Configuration Management In Heterogeneous ATM Environments*. In Proc. of the International Workshop on Agents in Telecommunications Applications IATA'98, AgentWorld'98, Paris, France, (4-7 July 1998).

[Pagurek 98b] B. Pagurek, Y. Li, A. Bieszczad, G. Susilo, *PVC Provisioning In Heterogeneous ATM Environments*. In Proc. of the 2nd Canadian Conference on Broadband Research (CCBR '98), Ottawa, Canada, (June 21-24, 1998)

[Papaioannou 00a] T. Papaioannou, *On the Structuring of Distributed Systems*. PhD Thesis, Loughborough University, (February 2000).

[Papaioannou 00b] T. Papaioannou, *Mobile Information Agents for Cyberspace – State of the Art and Visions*. Proc. of Co-operating Information Agents, (2000).

[Park 88] S.K. Park, R.W. Miller, *Random Number Generation: Good Ones are Hard to Find*. Communications of the ACM, Vol.31, N.10, pp.1192-1201, (October 1988).

[Pavlou 95] G. Pavlou, K. McCarthy, S. Bhatti, G. Knight, *The OSIMIS Platform: Making OSI Management Simple*. Integrated Network Management IV - New York: Chapman & Hall (1995).

[Pavlou 96] G. Pavlou, G. Mykoniatis, J. Sanchez, *Distributed Intelligent Monitoring and Reporting Facilities*, IEE Distributed Systems Engineering Journal (DSEJ), Special Issue on Management, Vol. 3, No. 2, pp. 124-135, IOP Publishing, (1996).

[Payer 97] Udo Payer, *Management by Delegation in ISDN-based Remote Access Environments*. In Proc. of the 8th Joint European Networking Conference. Edinburgh, (May 12-15, 1997).

[Peine 97] H. Peine, T. Stolpmann, *The Architecture of the Ara Platform for Mobile Agents*. In K. Rothernmel and R. Popescu-Zeletin editors, Proc. of the 1st International Workshop on Mobile Agents, N.1219 in Lecture Notes in Computer Science, Springer-Verlag, p.26 (April 1997).

[Pham 98] V. A. Pham, A. Karmouch, *Mobile Software Agents: and Overview*. IEEE Communications Magazine, pp.26-37, (July 1998).

[Puliafito 99] A. Puliafito, S. Riccobene, M. Scarpa, *An Analytical Comparison of the Clien-Server, Remote Evaluation and Mobile Agents Paradigms*. In Proc. of Joint Symposium: 1st International Symposium on Agent Systems and Applications; 3rd International Symposium on Mobile Agents, (September 99)

[Quittek 01] J. Quittek, M. Brunner, *Applying Active Technologies to Distributed Management*. Proc. of IM'01, Seattle, USA, (May 2001).

[Ranganathan 96] M. Ranganathan, A. Acharya, J. Saltz, *Distributed Resource Monitors for Mobile Objects*. Proc. of the 5th International Workshop on Object Orientation in Operating Systems, (1996).

[Ranganathan 97] M. Ranganathan, A. Acharya, S.D. Sharma, J. Saltz, *Network-aware Mobile Programs*. Proc. of the 1997 USENIX Technical Conference, pp. 91-104, (1997).

[Raza 98] K. S. Raza, *Implementation of Plug-and-Play Printer with Mobile Agents*, Technical Report SCE-98-04, System and Computer Engineering, Carleton University, (1998).

[Reid 91] K.B. Reid, *Centroids to Centers in Trees*. Networks, Vol.21, p.11-17, John Wiley & Sons, (1991).

[Rescigno 97] A. Rescigno, *Optimal Polling in Communication Networks*. IEEE Transactions on Parallel and Distributed Systems, Vol.8, N.5, (May 1997).

[Rothermel 97] K. Rothermel, F. Hohl, N. Radouniklis, *Mobile Agents: What is Missing?* Proc. of Distributed Applications and Interoperable Systems, DAIS'97, Chapman & Hall, pp.74-85, (1997).

[Rubinstein 00a] M.G. Rubinstein, O.C.M.B Duarte, G. Pujolle, *Improving Management Performance by Using Multiple Mobile Agents*. Forth International Conference on Autonomous Agents, ACM Agents 2000, pp. 165-166, Barcelone, Spain, (June 2000).

[Rubinstein 00b] M.G. Rubinstein, O.C.M.B Duarte, G. Pujolle, *Using Mobile Agent Strategies for Reducing the Response Time in Network Management*. 16th IFIP World Computer Congress, ICCT2000, pp. 278-281, Beijing, China, (August 2000).

[Rubinstein 00c] M.G. Rubinstein, O.C.M.B Duarte, G. Pujolle, *Reducing the Response Time in Network Management by Using Multiple Mobile Agents*. IEEE/IFIP Third International Conference on Management of Multimedia Networks and Services (MMNS’2000). In Managing QoS in Multimedia Networks and Services, Chapter 18, José Neuman de Souza and Raouf Boutaba (Ed.), pp. 253-265, Kluwer Academic Publishers, (September 2000).

[Rubinstein 00d] M.G. Rubinstein, O.C.M.B Duarte, G. Pujolle, *Evaluating the Network Peformance Management based on Mobile Agents*, Second International Workshop on Mobile Agents for Telecommunication Applications (MATA'00), Paris, France. In Lectures Notes in Computer Science 1931, Eric Horlait (Ed.), pp. 95-102, Springer-Verlag, (September 2000).

[Rubinstein 01] M.G. Rubinstein, *Mobile Agents in Network Management*. PhD thesis , Grupo de Teleinformática e Automação, Universidade Federal do Rio de Janeiro, (March 2001).

[Rubinstein 98] M.G. Rubinstein, O.C.M.B Duarte, *Service Location for Mobile Agent Systems*, IEEE/SBT International Telecommunications Symposium ITS'98, pp. 623-626, São Paulo, SP, Brazil, (August 1998).

[Rubinstein 99a] M.G. Rubinstein, O.C.M.B Duarte, *Evaluating Tradeoffs of Mobile Agents in Network Management*, Networking and Information Systems Journal, Hermes Science Publications, vol. 2, no. 2, pp. 237-252, (1999).

[Rubinstein 99b] M.G. Rubinstein, O.C.M.B Duarte, *Evaluating the Performance of Mobile Agents in Network Management*, IEEE Global Telecommunications Conference - Globecom'99, pp. 386-390, Rio de Janeiro, RJ, Brazil, (December 1999).

[Rubinstein 99c] M.G. Rubinstein, O.C.M.B Duarte, *Analyzing Mobile Agent Scalability in Network Management*, IEEE Latin American Network Operations and Management Symposium - LANOMS'99, pp. 64-74, Rio de Janeiro, RJ, Brazil, (December 1999).

[Sahai 97a] Akhil Sahai, Christine Morin, Stéphane Billiart, *Intelligent agents for a Mobile Network Manager (MNM)*. In Proceedings of the IFIP/IEEE International Conference on Intelligent Networks and Intelligence in Networks (2IN'97), Paris, France, (September 1997).

[Sahai 97b] Akhil Sahai, Stéphane Billiart, Christine Morin, *Astrolog: A Distributed and Dynamic Environment for Network and System Management*. In Proceedings of the 1st European Information Infrastructure User Conference, Stuttgart, Germany, (February 1997).

[Sahai 97c] Akhil Sahai, Stéphane Billiart, Christine Morin, *A Portable and Mobile manager for Distributed System Management*. In Proceedings of the Third Joint Conference on Information Sciences, Raleigh, North Carolina, USA, (March 1997).

[Sahai 98a] Akhil Sahai, Christine Morin, *Mobile Agents Enhanced Thin Client Approach to Network Management.* In Proceedings of IEEE Singapore International Conference on Networks (SICON' 98), Singapore, (June30-July 3 1998).

[Sahai 98b] Akhil Sahai, Christine Morin, *Mobile Agents for Enabling Mobile User Aware Applications*. In Proceedings of the Second International Conference ACM Autonomous Agents (Agents 98), Minneapolis/St.Paul, USA., (May 1998).

[Sahai 98c] Akhil Sahai, Christine Morin. *Towards Distributed and Dynamic Network Management*. In Proceedings of the IEEE/IFIP Network Operation and Management Symposium (NOMS), New Orleans, Lousiana, USA, (February 1998).

[Schonwalder 00] J. Schonwalder, J. Quittek, C. Kappler, *Building Distributed Management Applications with the IETF Script-MIB,* IEEE Journal on Selected Areas in Communications, Vol. 18, No. 5, p.702-714, (May 2000).

[Schonwalder 96] J. Schonwalder, *Using Multicast-SNMP to Coordinate Distributed Management Agents*. In Proc. 2nd Int'l. IEEE Workshop of Systems Management, Toronto, Ontario, pp. 136-41, (June 1996).

[Schonwalder 97] J. Schonwalder, *Network Management by Delegation – From Research Prototypes Towards Standards.* Proc. of the 8th Joint European Networking Conference (JENC8), Edinburgh (May 1997).

[Schonwalder 99] J. Schonwalder, J. Quittek, *RFC2593 – Script MIB Extensibility Protocol Version 1.0,* The Internet Society, (May 1999).

[Schoonderwoerd 96] R. Schoonderwoerd, J. Bruten, L. Ronthkrantz., *Ant-based load balancing in telecommunications networks*. Adaptive Behaviour, Vol.5, N.2, pp.169-207, (1996).

[Schrage 75] L. Schrage, *Implicit Representation of Variable Upper Bounds in Linear Programming*. Mathematical Programming Study, Vol. 4, pp.118-132, (1975).

[Schuringa 00] J. Schuringa, G. Remsak, *Packet Routing with Genetically Programmed Mobile Agents*. In R. Harmen (Editor), Telecommunication Network Intelligence. Proc. of the Sixth International Conference on Intelligence in Networks (SmartNet 2000), Kluwer Academic Publishers, pp.389-404, (September 2000).

[Shehory 98] O. Shehory, K. Sycara, P. Chalasani, S. Jha, *Agent Cloning: an Approach to Agent Mobility and Resource Allocation.* IEEE Communications Magazine, pp.58-67, (July 1998).

[Siegel 96] J. Siegel, *CORBA Fundamentals and Programming*. John Wiley & Sons, (1996).

[Siegl 95] M.R. Siegl, G. Trausmuth, *Hierarchical Network Management: a Concept and its prototype in SNMPv2*. Technical University of Wien, Austria. (May 1995).

[Silva 00] L. Silva, G. Soares, P. Martins, V. Batista, L. Santos, *Comparing the Performance of Mobile Agent Systems: A Study of Benchmarking*, Journal of Computer Communications, Special Issue on Mobile Agents for Telecommunication Applications, (January 2000).

[Sloman 94] M. Sloman, *Network and Distributed Systems Management*. Addison-Wesley Publishing Company, (1994).

[Soares 99] G. Soares, L.M. Silva, *Optimizing the Migration of Mobile Agents*. In Proc. of the 1st International Workshop oin Mobile Agents for Telecommunications Applications (MATA'99), pp.161-178, (October 1999).

[Somers 96] F. Somers, *HYBRID: Unifying Centralised and Distributed Network Management using Intelligent Agents*. Proc. IEEE Network Operations and Management Symposium (NOMS'96), Kyoto, Japan, April 1996. IEEE Press, New York, USA, (1996).

[Stallings 93] Stallings, W., SNMP, SNMPv2 and CMIP The Practical Guide to Network Management Standards. Addison-Wesley Publishing Company, (1993).

[Stallings 96] W.Stallings, SNMP, SNMPv2, and RMON Practical Network Management, Addison-Wesley, (1996).

[Stamos 90a] J.W. Stamos, D.K. Gifford, Implementing Remote Evaluation. IEEE Transactions on Software Engineering, Vol. 16, No.7, (July 1990).

[Stamos 90b] J. W. Stamos, D. K. Gifford, *Remote Evaluation*, ACM Transactions on Programming Languages and Systems, 12(4):537-565, (October 1990).

[Steenekamp 96] P. Steenekamp, J. Roos, *Implementation of Distributed Systems Management Policies: A Framework for the Application of Intelligent Agent Technology*. In Proc. of the 2nd Int'l. IEEE Workshop of Systems Management, Toronto, Ontario, pp. 127-35 (June 1996).

[Steward 94] S. Steward, S. Appleby, *Mobile software agents for control of distributed systems based on principles of social insect behaviour*. Singapore ICCS '94. Conference Proceedings. (Cat. No.94TH0691-6), pp549-53 vol.2, (1994).

[Straβer 97] M. Straβer, M. Schwehm, *A Performance Model for Mobile Agent Systesms*. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'97, Volume II, Editor H. R. Arabnia, Las Vegas, pp.1132-1140, (1997).

[Straβer 96] M. Straβer, J. Baumann, B. Hohl, *Mole – A Java Based Mobile Agent System*, Special Issues in Object-Oriented Programming: Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP'96, M. Muhlauser, Ed., pp.327-334, (July 1996).

[Suzuki 96] M. Suzuki, Y. Kiriha, S. Nakai, *Dynamic Script Binding for Delegation Agent*. DSOM '96, L'Aquila, Italy, (1996).

[Tansel 83a] B. C. Tansel, R. L. Francis, T. J. Lowe, *Location on Networks: a Survey. Part I: The p-center and p-median problems.* Management Science, Vol.29(4), pp.482-497, (April 1983).

[Tansel 83b] B. C. Tansel, R. L. Francis, T. J. Lowe, *Location on Networks: a Survey. Part II: Exploiting Tree Network Structure.* Management Science, Vol.29(4), pp.498-511, (April 1983).

[Theilmann 99] W. Theilmann, K. Rothermel, *Disseminating Mobile Agents for Distributed Information Filtering*. Joint Symposium ASA/MA'99, 1st Symposium on Agent Systems and Applications and 3rd International Symposium on Mobile Agents, pp.152-161, IEEE press, (1999).

[Thiel 91] G. Thiel, *Locus Operating System, a Transparent System*. Computer Communications, Vol.14(6), pp.336-346, (1991).

[Thompson 98] J.P. Thompson, *Web-based Enterprise Management Architecture.* IEEE Communications Magazine, 36(3), pp.80-86, (1998).

[Vassila 95] A. Vassila, G. Knight, *Introducing Active Managed Objects for Effective and Autonomous Distributed Management*. In Proceedings of the 3rd International Conference on Intelligence in Broadband Services and Networks IS&N 95, Heraklion, Crete, Greece, (October 16-20, 1995).

[Vassila 97] A. Vassila, G. Pavlou, G. Knight, *Active Objects in TMN*. In Proceedings of ISINM '97, (1997).

[VESPER] *Virtual Home Environment for Service PErsonalization and Roaming Users*. IST-1999-10825 (http://vesper.intranet.gr/).

[Vigna 98] G. Vigna, (editor), *Mobile Agents and Security*. Lecture Notes in Computer Science, LNCS 1419, Springer-Verlag, (1998).

[Waldbusser 91] S. Waldbusser, Remote Network Monitoring Management Information Base. RFC1271, IAB, (November 1991).

[Waldbusser 95] S. Waldbusser, Remote Network Monitoring Management Information Base. RFC 1757, (February 1995).

[Waldo 99] J. Waldo, *The Jini Architecture for Network-centric Computing*, Communications of the ACM Volume 42, No. 7, pp 76-82, (July 1999).

[Weir 97] C. Weir, *Architectural Styles for Distribution, Using macro-patterns for system design.* Second European Conference on Pattern Languages of Programming, EuroPLoP'97, (June 1997).

[White 94] J. E. White, Telescript Technology: The Foundation for the Electronic Marketplace. White paper, General Magic, Inc., (1994).

[White 95a] J. E. White, *Telescript Technology: An Introduction to the Language*. White paper, General Magic, Inc., (1995).

[White 95b] J. E. White, Telescript Technology: Scenes from the Electronic Marketplace. White paper, General Magic, Inc., (1995).

[Wies 97] R. Wies, M.A. Mountzia, P. Steenekamp, *A Practical Approach Towards a Distributed and Flexible Realization of Policies Using Intelligent Agents*. In Proc. of the IFIP/IEEE International Workshop on Distributed Systems Operations & Management (DSOM'97), Sydney, Australia, (October 21-23, 1997).

[Wijata 00] Y.I. Wijata, D. Niehaus, V.S. Frost, *A Scalable Agent-based Network Measurement Infrastructure*. IEEE Communications Magazine, (September 2000).

[Wooldridge 94] M. Wooldridge, N. Jennings. *Agent Theories, Architectures, and Languages: A Survey*. In Proc. of ECAI94 Workshop on Agent Theories, Architectures & Languages (eds M.J. Wooldridge & N.R. Jennings) Amsterdam The Netherlands, pp 1-32, (1994).

[Wooldridge 95] M. Wooldridge, N. Jennings. *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, Vol.10(2), pp.115-152, (1995).

[Wooldridge 96] M. Wooldridge. *INTELLIGENT AGENTS II: Agent Theories, Architectures, and Languages*. Springer-Verlag Lecture Notes in AI - Volume 1037, (1996).

[Wooldridge 98] M. Wooldridge, *Agent-based computing*. Interoperable Communication Networks, Baltzer Science Publishers, Volume I/I, (January 1998).

[X738 93] ITU-T Rec. X.738, Information Technology - Open Systems Interconnection - *Systems Management: Summarization Function,* (November 1993).

[X739 93] ITU-T Rec. X.739, Information Technology - Open Systems Interconnection - *Systems Management: Metric Objects and Attributes,* (November 1993).

[X753 97] ITU-T Rec. X.753, Information Technology - Open Systems Interconnection - *Systems Management: Command Sequencer for System Management.* ITU, Geneva, Switzerland, (October 1997).

[Yemini 91] Y. Yemini, G. G. Goldszmidt, S. Yemini, *Network Management by Delegation*. Integrated Network Management II, Amsterdam (1991).

[Yemini 93] Y. Yemini, *The OSI Network Management Model.* IEEE Communication Magazine, Pagg.20-29, (May 1993).

[Yucel 99] S. Yucel, T. Saydam, A. Mayorga, *A QoS-Driven Management Architecture using Mobile Agents*. Journal of Selected Areas in Communications, (1999).

[Zegura 96] E.W.Zegura, K.L.Calvert, S.Bhattacharjee, *How to Model an Internetwork*. IEEE INFOCOM 96, San Francisco, CA, (1996).

[Zegura 97] E.W. Zegura, K.L. Calvert, M.J. Donahoo, *A Quantitative Comparison of Graph-based Models for Internet Topology*. IEEE/ACM Transactions on Networking, (1997).

[Zhang 96] T. Zhang, S. Covaci, R. Popescu-Zeletin, *Intelligent Agents in Network and Service Management*. In Proc. IEEE Global Telecommunications Conference (GLOBECOM'96), London, UK, Novermber 1996. IEEE Press, New York, NY, USA, (1996).

[Zhang 97] Tianning Zhang, *Java-based Mobile Intelligent Agents as Network Management Solutions*. In Proc. of the 8th Joint European Networking Conference. Edinburgh, (May 12-15 1997).

[Znaty 94] S. Znaty, J. Sclavos, *Annotated Bibliography on Network Management*. Computer Communication Review, ACM-SIGCOMM, Vol. 24 N. 1, (January 1994).

Appendix: Mathematical Developments

The following mathematical series is used several times in Chapter 7. Its mathematical developments are, hence, showed in this appendix:

$$\sum_{l=0}^{(R(u)-1)} * \sum_{i=l}^{(R(u)-1)} n^i = \sum_{l=0}^{(R(u)-1)} * \left(\sum_{i=0}^{(R(u)-1)} n^i - \sum_{i=0}^{(l-1)} n^i \right)$$

If we develop the two inner geometric series (assuming *n*>1) we obtain

$$= \sum_{l=0}^{(R(u)-1)} \left(\frac{n^{R(u)}-1}{n-1} - \frac{n^l-1}{n-1} \right) = \sum_{l=0}^{(R(u)-1)} \left(\frac{n^{R(u)}-1}{n-1} - \frac{n^l}{n-1} + \frac{1}{n-1} \right)$$

A fundamental properties of series allows to express the series of sums as the sum of series; hence

$$= \sum_{l=0}^{(R(u)-1)} \left(\frac{n^{R(u)}-1}{n-1} \right) - \sum_{l=0}^{(R(u)-1)} \left(\frac{n^l}{n-1} \right) + \sum_{l=0}^{(R(u)-1)} \left(\frac{1}{n-1} \right)$$

In the first and third series the terms inside the series are independent from the summation variable *l* and can, thus, be carried out of the series (they need to be multiplied by *R(u)*). These terms have the same denominator and can then be directed added, leading to the following expression:

$$= R(u) * \frac{n^{R(u)}-1+1}{n-1} - \sum_{l=0}^{(R(u)-1)} \left(\frac{n^l}{n-1} \right)$$

If we now develop the remaining geometric series we obtain

$$= R(u) * \frac{n^{R(u)}}{n-1} - \frac{1}{n-1} * \frac{n^{R(u)}-1}{n-1} = \frac{R(u) * n^{R(u)} * (n-1) - n^{R(u)} + 1}{(n-1)^2}$$

If we consider the asymptotic behaviour of this expression for R(u) →∞ we achieve the following expression

$$\sum_{l=0}^{(R(u)-1)} * \sum_{i=l}^{(R(u)-1)} n^{i} \propto O\left(R(u) * n^{R(u)}\right)$$