CITY UNIVERSITY

London

Distributed Termination Detection For Multiagent Protocols

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Tshiamo Motshegwa

October 2009

To my mother . . .

who—among so many other things—

taught me the virtues of humility and sacrifice

To my father . . .

who—among so many other things—

taught me the virtue of discipline

To my late grandparents . . .

who—for all things—

provided a window to my past.

Table of Contents

1 Introduction **1**

1.1 Hypothesis 3

1.2 The Problem 4

1.3 Assumptions 6

1.4 Contribution and Originality 9

1.5 Exploitation 10

1.6 Thesis Roadmap 12

1.7 Summary 13

2 State of the art in Termination detection in distributed Systems **15**

2.1 Introduction and Background 15

2.1.1 Overview and terminology 17

2.2 Classical Algorithms and Taxonomy 19

2.2.1 Tracing Algorithms 20

2.2.2 Wave-Based Solutions 21

2.2.3 Other approaches to termination detection 22

2.3 A general computing model and termination detection 25

2.4 A contemporary taxonomy for distributed termination detection algorithms 29

2.5 Summary 34

3 Distributed Systems vs Multiagent Systems **37**

3.1 Introduction: On Agents and Multiagent Systems 37

3.2 Multiagent systems and Distributed systems 39

3.3 Summary 43

4 From computer protocols to agent interaction protocols 44

4.1 Introduction 44

4.2 On computer protocols 45

4.3 Specifying Protocols 46

4.3.1 A formal model of protocol systems 46

4.3.2 Formal Languages for protocol specification 52

4.4 Interaction protocols in Multi-Agent systems 54

4.4.1 Protocol engineering in multiagent systems 55

4.4.2 Discussion 62

4.5 Summary 64

5 Termination Detection for Protocols 66

5.1 Introduction 66

5.2 Overview 68

5.3 Definitions 69

5.4 Termination detection model 83

5.4.1 A model for agent conversations 83

5.5 Distributed protocol for termination detection 96

5.5.1 Evaluation of the distributed termination detection protocol . . 105

5.5.2 Proposed architecture for termination detection 108

5.6 Summary and Contributions . . . 110

6 Prototype implementation, experimental design and data analysis . . . 114

6.1 Introduction . . . 114

6.2 Overview . . . 114

6.3 Experimental setup . . . 115

6.4 Prototype implementation . . . 119

6.5 Scheduled experiments . . . 120

6.6 Data collection . . . 124

6.7 Data analysis . . . 127

6.8 Summary and contributions . . . 129

7 Partial Protocol Results . . . 131

7.1 Introduction . . . 131

7.2 Results: Exploratory Data Analysis . . . 133

7.3 Confidence intervals, standard errors and the Bootstrap . . . 144

7.3.1 Bootstrap results . . . 145

7.4 Summary . . . 156

8 Full Protocol Results . . . 157

8.1 Introduction . . . 157

8.2 Results: Exploratory Data Analysis . . . 158

8.2.1 Bootstrap results . . . 165

8.3 Summary and observations . . . 174

9 Comparisons . . . 175

9.1 Introduction . . . 175

9.2 Hypothesis testing . . . 178

9.3 Scalability Results . . . 182

9.4 Summary . . . 186

10 Distribution . . . 187

10.1 Introduction . . . 187

10.2 Experimental Results and Analysis . . . 190

10.2.1 A note on the presentation of results . . . 190

10.2.2 Results for network latency . . . 191

10.3 Results for detection delays in the distributed setting . . . 196

10.3.1 Results for scenario 1 . . . 196

10.3.2 Results for scenario 2 . . . 204

10.3.3 Scenario 3 . . . 208

10.3.4 Scenario 4 . . . 214

10.3.5 Scenario 5 . . . 218

10.4 Summary . . . 229

11 Conclusion . . . 230

11.1 Discussion of other issues . . . 230

11.2 Summary . . . 232

11.3 Contributions . . . 233

11.4 Critical Review . . . 234

11.5 Directions for future work 236

11.6 Related Work 236

References 239

A Termination Detection for protocols 261

A.1 Illustration for the diffusing conversations algorithm 261

A.2 Other possible implementations of a global c-matrix 263

A.3 Algorithms 266

A.4 Discussion and complexity analysis 268

B Prototype Implementation 273

B.1 Java Agent Development Framework 273

B.2 Implementation 275

C Experimental design and data analysis 280

C.1 Data collection setup 280

C.2 Data collection during experimental cycles 282

C.3 Hardware used in experiments 285

D Tutorial on resampling statistical methods 287

D.1 The Bootstrap 287

D.2 Bootstrap confidence intervals 293

D.3 The Jackknife 298

D.3.1 Jackknife-After-Bootstrap 300

D.4 Computational statistical tools support for resampling techniques . . . 302

D.5 Experiences with jacknife-after-bootstrap 303

E Statistical tests 308

E.1 Goodness of fit tests 308

E.2 Hypothesis Tests 317

E.3 Repeatability of experiments 318

F All summary statistics 321

F.1 Partial Protocol data sets 321

F.2 Full Protocol datasets 324

G Partial Protocol experimental runs results 330

G.1 Resampling, Bootstrap confidence intervals 336

H Network latency and Synchronisation 339

H.1 Clock synchronisation algorithms 340

H.2 Network latency experiments 342

I Distributed Termination detection 346

I.1 Tracing Algorithms 346

I.2 Some Selected Algorithms, some optimisations and robustness considerations 348

I.3 Static and dynamic termination algorithms 350

I.4 Contemporary taxonomy for algorithms 356

J Graphs, representation and complexity of algorithms 372

List of Algorithms

1 Shortest Termination paths algorithm 80

2 Reachability algorithm 81

3 Termination paths update Algorithm 82

4 Diffusing conversations algorithm 93

5 Algorithm for predicate $activeProtocol$ 94

6 Global termination on controllers $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$ 100

7 Local termination on controller $\mathcal{C}_n$ for agent i, handles $query\,(\mathcal{C}_n, i)$ messages from $\mathcal{C}_0$, returns $\mathcal{C}_n^i \mapsto F$ from this controller 101

8 Synchronisation of cache updates across controllers in $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$ 267

9 The Bootstrap algorithm 288

ACKNOWLEDGMENTS

First of all I would like to thank my supervisor Dr Eduardo Alonso for showing great patience when the drafts were taking longer than they ought to have been. I would also like to thank him for providing continuous support during the course of the PhD and making available the opportunities and provisions for me to do some undergraduate module teaching to support my studies.

I would like to thank Prof. M. Schroeder for origionally helping me secure a City University PhD studentship on completion of my undergraduate studies at City university, for arranging that I spend time at British Telecom Labs during the first year of the PhD and for the involvement in the AgentCities project.

I would also like to sincerely thank the examiners for all they have done at all the stages of the examination process, clearly at a great sacrifice of time on their part.

I would like to acknowledge Prof. Michael P. Wellman at Michigan university for responding to my query about his 1999 paper titled "*Distributed Quiescence Detection in Multiagent Negotiation*". I discovered the paper when I was in British Telecom , BTexact Laboratories in Mattlesham Ipswich on an extended visiting research student fellowship away from City University and still searching for a PhD topic.

In the paper he tabled the possibility of using Dijkstra-Scholten algorithm for quiescence detection in a multiagent protocol. The interaction with Prof Wellman prompted me to pursue the research topic I have chosen as on enquiring about any follow up to this work, mentioned there was none but that the area was well studied in distributed systems with early work in the 1980's but not so much within agents and that there is big scope for research.

I would also like to thank a number of people I have come across during the course of the PhD. I would like to thank Dr Simon Thompson at BT Labs for the 6 months I spent at the labs and the interactions we had. I would also like to thank Dr Christos Kloukinas, Dr Tillman Weyde, Prof. Bernie Cohen, Dr Kostas Statis and other current and past academic members of the City university Computing Department. I would also like to thank Dr Michael Rovatsos at the School of Informatics at University of Edinburgh.

I would also like to acknowledge other staff and PhD students over the years at the department of computing at City university, and other colleagues namely Dr Rodrigo Agerri, Dr Jacek Gomuluch, Dr Penny Noy, Dr Panos Dafas, Mr Alexander Kozlenkov, Mr George Lekeas, Mr George Katapodis, Mr Gilberto Cysneiros Filho, Mr Rafael Borges, Mr Chris Child, Mr Marcus Andrews, Dr Cornelius Ncube, Mr Peter Kristoffersson-Izmajlow, Dr Teddy Ngosi, Dr Tak-Shing Chan, Dr Eddy Parkinson and others for providing a lively research environment. I would also like to particularly thank Mark Firman and Edna Morris for all they have done in supporting me and other research students by unravelling and elucidating the processes within the department and the university. I would also like to thank the librarian Mrs Sheila Munton for always locating those hard to get documents. I would also like to thank Chris Marshall for making available more hardware when I needed it the most.

Finally I would like to also thank my dear family for all they have done from as long as I can remember. I would like to thank my parents for the sacrifices they have made for us. I would like to thank my dear sister, Ms Agnes Ntwaagae, my brothers, Dr Kefentse Motshegwa, Mr Pantanowitz Motshegwa and Mr Moduku Motshegwa for providing the context in which I grew up.

I would also like to convey my deepest gratitude to my grandfather who provided a window the past but passed on at a ripe old age of well over a hundred years, for his words of wisdom. In return, I was never able offer him a plausible explanation why I

could not study medicine instead.

On one of his quotes, he says;

> *Even the intrepid wayward of dragon slayers do return home, for the next dragon is the one that will cause his demise..*

Abstract of the Dissertation

Distributed Termination Detection For Multiagent Protocols

by

Tshiamo Motshegwa

Doctor of Philosophy in Computer Science

City University, London, October 2009

The research conducted in this thesis is on distributed termination detection in multiagent systems.

Agents engage in complex interactions by executing behaviour specifications in the form of protocols. This work presents and experiments with a framework for making termination in a multiagent system explicit. As a side effect, the mechanism can be exploited to aid management of agent interactions, by providing visibility of the interaction process and can be extended to drive multiagent system management tasks such as timely garbage collection.

Results from previous attempts to deploy agents systems when scaling up, e.g. Agentcities, have shown and exposed a big gap between theory and practice especially in the reliability and availability of deployed systems. In particular more work needs to be done in the area of supporting agent infrastructures as much as in theoretical agent foundations.

There are two aspects to this problem of termination detection in multiagent systems, firstly, the formal verification of behaviour at compile-time and secondly, monitoring and control at run-time. Regarding the former, there has been some work on the ver-

ification of agent communication languages. But overall verification is difficult and often requires knowledge of internal states of agents at compile time, and as yet has not been satisfactorily solved to be deployed in real systems. The second, the runtime approach is adopted in here.

The research is not about protocol engineering but assumes correct protocols, and protocol specifications to be finite state machine graphs. Given these correct verified protocols, the thesis proposes a number of definitions culminating in identification of minimal information in the form of sub-protocols that agents being autonomous, can make available for the termination detection. An off line procedure for deriving these sub-protocols is then presented.

The thesis then considers a termination detection model, and within this model, proposes an conversation model encompassing protocol executions, with hierarchical conversations modelled as diffusing computation trees and defines a number of predicates to derive termination in centralised and distributed environments. Algorithms that implement these predicates are sketched and some complexity analysis is performed. The thesis then considers a prototype implementation evaluated over some defined detection delays metric.

The evaluation approach is heavily empirical, with an experimental approach adopted to evaluate various configurations of the termination detection mechanism. The evaluation employs robust resampling and bootstrapping methods to analyse and obtain distributions and confidence intervals of the detection delays metric for the termination detection mechanism.

CHAPTER 1

Introduction

Like mathematicians, computer scientists use formal languages to denote ideas. Like engineers, they design things, assembling components into systems and evaluating tradeoffs among alternatives. Like scientists, they observe the behavior of complex systems, form hypotheses, and test predictions,

— Allen B. Downey.

Aspiring computer scientist aspire to do these things.

The research in this thesis is in the area of multiagent systems (MAS). Multiagent systems are related to distributed systems in that they are inherently distributed and distributed systems offer a platform for developing multiagent systems.

Differences are often cited to exist between the agent and object models and related communication models assumed in the two areas. Agents are deemed autonomous (in theory), are considered higher level entities which use a rich communication language and execute interaction protocols to engage in potentially complex goal oriented interactions in dynamic and uncertain environments. By contrast objects are by and large passive, with no real control over execution of their methods, for example. But as observed in [239], it appears the debate on agents and objects (processes) has moved on to converge to a consensus that agents and objects to occupy different realms and can co-exist.

It is worth observing though that, while distributed systems emphasise distribution of resources, MAS in addition, emphasise distribution of objectives, distribution of problem solving[1] (e.g. by the divide and conquer metaphor), coordination of actions and flexible interaction in open environments.

Given these assumptions, there are important issues addressed in the distributed system research and the corresponding results that can be adopted in developing multiagent systems but taking into account issues pertinent to the agent model of computation.

This thesis proposes to look at one particular area, distributed termination detection. We consider this in a multiagent system, identifying agent interaction protocols as the mechanism that enable coordination and flexible interaction between agents and using this as a starting point of our model.

Termination is an example of a stable global state of a distributed system. Components of a distributed system have only local views of a computation and lack a global perspective. So ascertaining that a distributed computation has terminated is not straight forward since it requires a global view.

The termination detection problem is related to the more general problem of detection of global predicates, a fundamental problem in debugging and monitoring.

It has also been shown [227], that the semantics of the garbage collection problem are contained in the semantics of the termination detection problem, in that, with appropriate transformations garbage collection schemes can be derived from termination detection schemes.

So termination detection has useful applications as the terminated status is among stable states (consider as another example global communication deadlock) that should be known for system administration. In summary consider the following applications of termination detection;

[1]Though this is also present in non-agent based distributed systems.

1. *Distributed workpool*, i.e. dynamic mapping of tasks onto processes for load balancing in which any task may potentially be performed by any process, and if the work is generated dynamically and a decentralized mapping is used, then a termination detection algorithm would be required so that all processes can actually detect the completion of the entire program and stop looking for more work.

2. *Deadlock detection*, a stable state where there are *wait-for* cycles ,i.e. when two or more processes permanently block each other by each process having a lock on a resource which the other process are trying to lock.

3. *Crash recovery*, recovering and rolling back from abnormal termination.

4. *Garbage collection*, because termination detection is related to garbage collection, it is possible with appropriate transformations to derive garbage collection schemes from termination detection schemes as [227] has shown.

We wish to explore research done in distributed systems in the area of termination detection to provide a basis for development of a class of mechanisms to make termination explicit in multiagent systems, (possibly at a tradeoff with some autonomy of agents, not an unrealistic assumption for practical systems). This mechanism can be exploited in the future for work on automatic garbage collection of multiagent systems in automated environments.

1.1 Hypothesis

The primary goal of this research is to study termination detection in multiagent systems and to design, implement and experiment with a mechanism for detecting termination of agent interactions in multiagent systems. A working hypothesis is that the mechanisms will allow for timely detection of termination in multiagent interactions.

1.2 The Problem

One of the underlying assumptions in the development of societies of interacting autonomous agents is that we can fully specify correct and predictable interaction protocols and mechanisms apriori. Given that this has been achieved, collaborating or self-interested agents can then engage in complex interactions such as negotiation to achieve their goals. Equipped with these capabilities and imbued with specific private strategies and resources, agents can be let loose in open environments to perform complex transactions on behalf of their owners. But given the nature of the interaction space (potentially large, open and unpredictable environments) and inherent uncertainty in open systems, it is extremely difficult to specify and predict fully apriori such interactions and their likely consequences without building overly complex monolithic agents.

Also experiments and experiences with an attempt to do deploy agents for services on a global network called AgentCities [66] are recounted in [242] and [67] lists concrete challenges for AgentCities service environment as follows, quote;

– i – Automation, i.e. management of autonomy:- Understanding how to effectively automate systems in an open environment, how to control and manage deployed automated systems. This must draw on work from mathematical control theory to distributed systems and agent technology,

– ii – Interoperability, i.e. communication:- How to enable on-line software systems to interact with one another in increasingly flexible ways: configurable interaction sequences, communication about arbitrary domains,

– iii – Coordination:- Putting in place frameworks that enable automatic creation, maintenance, execution and monitoring of contracts and agreements between automated systems to fulfil their business objectives,

– iv – Knowledge acquisition (interfaces between worlds):- Putting in place frameworks that enable automatic creation, maintenance, execution and monitoring of contracts and agreements between automated systems to fulfil their business objectives.

We claim and position our research to make a contribution to the first point above, observing that multiagent systems are distributed and are implemented on distributed systems infrastructures, and noting (as has been elsewhere [247]), that research efforts in agent infrastructure support should necessarily draw upon experiences and coordinate with the general distributed systems research.

We observe that, while autonomy is a key feature in agent based systems, some level of control for the highlighted purposes should be acceptable when building non-trivial agent-based applications, such as the ones envisaged in global agent based service provision networks as exemplified by past initiatives such as AgentCities [66]. This is not an unreasonable assumption given that, while autonomous, agents in real applications participate in societies governed by some enforced rules of participation.

Chapter 3 from page 37 considers in detail multiagent systems and traditional distributed systems to motivate and highlight why some traditional problems in distributed systems like the termination detection problem may need further consideration within agent computational model assumed in multiagent systems. As a prelude to that discussion, consider that agents are deemed to generally have a high degree of autonomy about what they do, and regarding termination detection, agents may offer additional information about the execution of their protocols to facilitate the termination detection process for example.

Furthermore, consider the flexibility of interactions in multiagent systems, in particular consider a multiagent society with provisions for dynamism in protocols, i.e. in such environments;

– i – There could be support for dynamic execution of coordination protocols as proposed by [31], i.e. where the role an agent intends to hold within a protocol can be played without the need of prior knowledge.

– ii – There could be an infrastructure for dynamic protocol specification as discussed in [14], an infrastructure that accommodates revision of protocol specifications during execution in situations where there is such a strong requirement. This approach can be contrasted with the traditional one where specification of protocols has largely been considered as a design-time activity.

– iii – There could be infrastructure support for runtime protocol discovery in general.

Given these points, we propose that there is a plausible case for considering further how issues like termination detection can be addressed within multiagent systems environments.

1.3 Assumptions

From an Artificial Intelligence perspective, agents are communicative, intelligent, rational and possibly intentional entities. From the *computing* perspective, they are autonomous, asynchronous, communicative, distributed and possibly mobile processes [191] and multiagent systems or societies of agents are *modular* distributed systems and have *decentralized* data. Agents in a society have *incomplete information* or capabilities and interact to further their goals.

So regarding our research assumptions, we;

– i – We accept the computing perspective of agents as detailed above, and do not for example consider mentalistic notions in agents.

– ii – Furthermore, assume that an agent's behaviours are specified through public protocols for example available through public a library. We assume public protocols so that an external entity, e.g a resource manager, can know about terminating states and protocol paths.

– iii – Assume that all messages are observable in principle so that in general an observer can decide if a terminating state has been reached.

– iv – Assume a total ordering of messages, e.g. through a global clock. This is essential because temporal ordering of messages are used to identify current state of protocols. This global clock can be realised through clock synchronisation in a distributed system for example.

– v – And without loss of generality assume further that protocols are in the form of finite state machines, edge-labelled directed graphs and assume existence of transformations of other behaviour specifications to finite state machines (FSMs) for use with our mechanism. The basis for this assumption is discussed in Chapter 4.4, page 54, but as a prelude to that, we assume FSMs because;

  - Most of the models used in protocols specification are mainly extensions of finite state machines, i.e. FSMs underpin the current study of protocols.
  - FSMs are grounded in sound theoretical foundations and are well understood.
  - FSMs are relatively simple to implement.

- FSMs are accompanied by a variety of techniques and tools for formal analysis and design.
- FSMs have an intuitive graphical representation and graph theoretic approaches can be used with the resulting protocol structures

– vi – Regarding properties of agents, particularly the notion that agents can be persistent, we assume non-persistent agents, agents with a known lifecycle. In the case of persistent agents, assume existence of *copies* of these agents whose resources can be recouped once they have played their part in interactions. We assume that an agent or such a copy of an agent is terminated if all protocol executions in its set of interactions have reached terminal states. Regarding the notion of autonomy, we assume agents to be autonomous and this feature allows them to offer runtime information about their public protocol executions but are constrained by some society rules, e.g. obligation to register and provide this information.

Research Methodology Regarding the research methodology [129] observes that is no one standard way of conducting research in an evolving discipline like computer science and goes on to discuss models of argument, namely proof by demonstration, empiricism, mathematical proof and hermeneutics.

Our approach is to develop a model and a framework and conduct a simulation and perform detailed experimental evaluation (empiricism) to provide a demonstration and set benchmarks for future work. So the methodology can be thought of as using empiricism and coupled with proof of concept by demonstration.

1.4 Contribution and Originality

We propose a concrete and generic method for termination detection for multiagent systems discussed and evaluated in Chapters 5 to 10.

With this method, we propose to have made a number of contributions; On a *theoretical level*, we have considered the distributed termination detection research from distributed systems and considered it in the agent model and used this a basis for developing a class of agent control mechanisms.

1. To this end, in Chapter 5, we present definitions related to protocols, and define minimal protocol information agents can make available and propose a termination detection model.
2. We present an agent conversation model, and define some predicates and present algorithms for their implementation.
3. Combining all these we present a distributed protocol for termination detection and consider distribution possibilities.

On a *practical level*, we have offered a structured and systematic, methodical experimental framework, i.e.

1. In Chapter 6 we offer a prototype implementation for flat conversations and use it to evaluate the proposed mechanism and various configurations. The experimental prototype uses and tests a widely used agent development framework.
2. Again in Chapter 6 we define an extensive experimental and data analysis framework that uses robust resampling methods for quantitatively evaluating a prototype in this research. We claim that this experimental framework can also be used evaluating future contributions in this area as none to my knowledge exist.

3. Equally experimental work and results here can also set a benchmark for future work for comparison.

Aspects of these contributions have been previously published in [174] and are subject of papers in progress resulting from research discussed in this thesis.

1.5 Exploitation

In addition to the applications given on page 2, consider the following example scenarios for how termination detection maybe be exploited in agent applications.

Automated Auction-based marketplaces Consider an automated agent-based market place hosting an auction with numerous agents. Typically participants maintain varying valuations of goods and bid to those upper bounds according to adopted private strategies. Inevitably most participants will drop off early from the game. Typically in real applications, these entities would stay on longer than need be consuming system resources. In most applications this is not a concern. But where scalability and resource consumption is an issue, a deliberative mechanism for identifying and timely garbage collecting defunct agents is a necessity.

Multi-agent Negotiation Other uses of timely detection of termination in agent systems is in the area of multi-issue negotiation where an agent participates in numerous interactions to acquire resources forming a composite service for example. The overall negotiation is only complete when all deals are closed. So a mechanism for ascertaining this state would be useful.

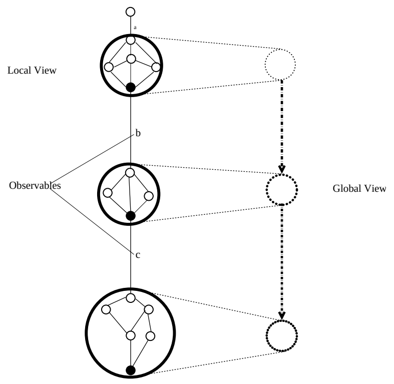


Figure 1.1: Mapping local view to global view for a process

Business Processes Consider a business process, with internal processes, transitions and stages. The outside observer, for example a manager, does not need to know the details of the internal processes, but would need to keep track of *deliverables*. This can be achieved by reporting or by maintaining a list of *checkpoints* or *observables*, actions marking *stage* transitions. The external entity would then keep track of particular terminal states marking end of a stage and transitions from them leading to the next phase in the process.

A cluster of local states and transitions separated by designated terminal states and checkpoints or observables can be viewed as an aggregate that can map to a *state* in the partial global view, and this view maybe what is required by an external entity to infer progress in the underlying process. Figure 1.1 illustrates this process.

In all these scenarios, we can envisage some protocol executions that can possibly be composed with a termination detection protocol or can overseen by a termination detection mechanism, and in line with the last assumption stated in page 7, termination of agents can then be eventually derived.

1.6 Thesis Roadmap

Part I presents background work, the material there is not my work apart from the analysis where given, the updated taxonomy and the survey of the related areas from the given references. In Part I ;

1. **Chapter** 2 provides a theoretical back drop presenting background research in termination detection, detailing models, algorithms and their taxonomy.
2. **Chapter** 3 Briefly discusses multiagent systems, their agent model of computation, distributed systems and the process or object model.

In Part II, **Chapter** 5 presents

1. Definitions related to protocols, and defines minimal protocol information agents can make available.
2. A termination detection model, comprising a conversation model, a set of predicates and algorithms for their implementation.
3. A distributed protocol for termination detection and distribution possibilities.

Part III presents experimental details and results, with

1. **Chapter** 6 detailing experimental setup, prototype implementation, experimental design, and detailing data collection and analysis,
2. **Chapters** 7 , 8 , 9, 10 presenting experimental results and data analysis,
3. **Chapter** 11 offers a summary and conclusions

4. Appendices A to I providing supplementary background material, illustrations, further analysis or repeated data analysis , data summaries for the experimental part of the work. These appendices can be consulted when referenced in the thesis for illustrations if necessary.

1.7 Summary

The research documented in this thesis is about termination detection in multiagent systems. The problem is encountered and widely researched in distributed systems. There are benefits of applications that can be accrued in considering this problem in multiagent systems, but this requires consideration of properties of agents such as flexible interaction and coordination through interaction protocols, autonomy, possible runtime protocol discovery, protocol specification revision and dynamic execution of coordination protocol with flexible roles. Therefore this highlights the need to reconsider research with respect to agents, observing that agents being autonomous, they can make available additional information about protocols they are executing.

This research proposes contributions at two levels, at a theoretical level, a consideration of the termination detection problem in the agent model, and at a practical level implementing and experimentally evaluating a mechanism for termination detection in a multiagent environment.

Next, **Part I** of the thesis provides background material with Chapter 2 providing a detailed survey of the research done in the area of termination detection in distributed systems to provide a theoretical backdrop for our work.

PART I
Background

CHAPTER 2

State of the art in Termination detection in distributed Systems

The problem we wish to discuss in this thesis was originally formulated in the area of distributed systems. This chapter serves to provide a self contained survey of the history and state of the art in the area of distributed termination detection. The purpose of this being to provide a context and the theoretical underpinnings for the proposed research work in this area within multiagent systems.

The chapter is structured as follows; A general introduction to the field is first given, illustrating briefly application areas, terminology, models, and then the problem formulation. A taxonomy and example classical algorithms are then discussed next. Then finally a selection of recent algorithms is presented to reflect current activity in the field.

2.1 Introduction and Background

There are times when there is a need to ascertain whether a condition is true for a distributed system and the condition cannot be judged *locally* but requires *global* knowledge of the state of the system. Distributed termination detection is one example that encapsulates this problem, other examples include distributed deadlock detection, distributed garbage collection and distributed debugging.

Distributed termination detection (from hereon referred to as DTD) is a fundamental problem in distributed computing. It is a classical problem of distributed control, and it is considered to be of practical, algorithmical, theoretical and methodical importance [227].

The termination detection problem is related to the more general problem of detection of global predicates, a fundamental problem in debugging and monitoring [16].

In general, a distributed system can be viewed as a set of autonomous processes which cooperate with each other to compute a task. To coordinate computation and exchange data, processes may communicate with each other by message-passing. Termination detection refers to the necessity of determining whether the system has entered a *silent* status where all processes are idle and no computation is possible to take place in the future [233].

The level of difficulty to detect such a status depends on the nature of the distributed system, but is usually non-trivial due to the variation of processor speeds and the unpredictable delays of the message delivery and the absence of global clocks. The distributed termination problem was first identified by [78], and has since inspired a lot of research interest as reflected in various literature, (e.g. [77, 231].

DTD is closely related to other important problems such as deadlock detection [178, 44], garbage collection [227, 228] and snapshot computation [46]. Indeed with garbage collection [227] has shown that the semantics of the termination detection problem are fully contained in the garbage collection problem and that with appropriate program transformations, solutions for the garbage collection problem can be applied to termination detection and vice versa.

Application of Distributed Termination Detection DTD has many applications: It serves an important role in the *diffusion computation* [78, 233] and the *distributed workpool* models which are commonly used in distributed and parallel computational models [7]. The work pool or the task pool model is characterized by a dynamic mapping of tasks onto processes for load balancing in which any task may potentially be performed by any process. There is no desired preassignment of tasks onto processes [7]. The mapping may be centralized or decentralized. [139] observes that, in a workpool, if the work is generated dynamically and a decentralized mapping is used, then a termination detection algorithm would then be required so that all processes can actually detect the completion of the entire program (i.e., exhaustion of all potential tasks) and stop looking for more work.

Furthermore, the terminated status of a distributed system is among the stable states (such as global communication deadlock, token loss) that should be known for system administration [233]. It has also been shown that termination detection schemes can be applied to solve other distributed computing problems such as deadlock detection, checkpointing [64], and crash recovery among others.

2.1.1 Overview and terminology

A distributed algorithm terminates when it reaches a *terminal state*, a configuration in which no further event is applicable.

Techniques have been developed to make termination explicit by distributively detecting that the program has reached a terminal configuration. These are the techniques we set out to explore in this section.

A very informal problem statement can be formulated as follows; Given a network of N nodes, implement a distributed termination detection algorithm. Each node can be either in ***active*** or in ***passive*** state. Only an active node can send messages to other nodes; each message sent is received after some period of time later. After having received a message, a passive node becomes active; the receipt of a message is the only mechanism that triggers for a passive node its transition to activity. For each node, the transition from the active to the passive state may occur *spontaneously*. The state in which all nodes are passive and no messages are on their way is ***stable***: the distributed computation is said to have ***terminated***. The purpose of the algorithm is to enable one of the nodes, say node 0, to detect that this stable state has been reached.

Definition of termination detection Consider this informal definition by [161],

> *A distributed computation is considered globally terminated if every process is locally terminated and no messages are in transit. Locally terminated can be understood to be a state in which the process has finished its computation and will not restart unless it receives a message.*

Consider a formalisation given by [233], summarised here, quote;
A distributed system consists of a set of processes $\mathcal{S} = \{P_1, P_2...., P_n\}$ which cooperate with each other to complete a job. Processes can communicate with each other by message-passing. Logically, from each P_i to each P_j there is a communication channel $C_{i,j}$. A process may switch between two states: *active* and *idle*. A process when performing some computation is said to be in the active state. An active process is free to send/recieve messages and may become idle spontaneously. On idle state, a process does not perform any computation, but can passively receive messages, on which event it becomes active immediately and starts computations. For distinction , computation carried out and messages transmitted by the system are called *basic computation* and basic *messages* respectively.

The distributed system is said to be terminated *iff* (i) P_i is idle and (ii) $C_{i,j}$ is empty for all $1 \leq i,\ j \leq n$ (condition (ii) is necessary because message delays are unpredictable and any *hidden* message will wake up the system later). When terminated , no distributed process can become active and perform any further computation. Extra messages, called control messages are sent , or extra information associated with basic messages to detect such a state. This is the distributed termination detection problem [233].

So the following definition follows;

Definition 1. *Termination detection*
Let $P_i(t)$ denote the state (active or idle) of process P_i at time t and $C_{i,j}(t)$ denote the number of messages of messages in transit in the channel at an instant t from process P_i to process P_j. A distributed computation is said to be terminated at time instant t_0 iff: $(\forall P_i(t_0) = idle) \wedge (\forall C_{i,j}(t_0) = 0)$.

2.2 Classical Algorithms and Taxonomy

In early research in the areas, (1980's) the termination detection algorithms were identified roughly fall into two categories, namely;

- *Tracing algorithms* (computation tree based). Algorithms of this type follow the computation flow by tracing active nodes along the message chains that activated them, and call termination when all traced activity has ceased.

- *Probe algorithms* (wave based) A probe is a distributed activity that visits all processes in the network (can be implemented by a token circulating on a ring or by an echo mechanism.

  Algorithms of this category rely on global (coordinated) scans of the network state and call termination when no activity is found. The distinction can be compared to that between reference counting and mark-sweep type garbage collectors [130].

The next section considers in more detail, the two classes of algorithms given in the taxonomy above and gives example algorithms.

2.2.1 Tracing Algorithms

A tracing algorithm relies on the knowledge of the set of initially active nodes, because all activity of the computation originates from these nodes by message chains. Solutions of this type are based on maintaining dynamically a directed graph, called a computation graph (spanning tree), of which the nodes include all active processes and all basic messages in transit [226].

Termination is detected when the computation graph becomes empty. One requirement is that the network be bi-directed , i.e. messages can be sent in two directions via each channel. The Dijkstra-Scholten algorithm [78] describes a solution for centralized basic computations, in which the computation graph is a tree with the initiator as the root (the only node initially active). The Shavit-Franchez Algorithm [206] generalises this solution to decentralised basic computations and uses a forest, in which each initiator of the basic computation is the root of a tree.

To illustrate in detail this class of algorithms, we consider the details of the Dijkstra-Scholten Algorithm below.

The Dijkstra-Scholten Algorithm The algorithm of Dijkstra and Scholten detects the termination of a *centralised* basic computation (called a diffusing computation [78]). The initiator of the algorithm (called the environment) also plays an important role in the detection algorithm.

Intuitively, the algorithm works as follows:

– i – Every node [1] maintains a counter c. Sending a message increases c by one; the receipt of a message decreases c by one. The sum of all counters thus equals the number of messages pending in the network. When $node_0$ initiates a detection probe, it sends a token with a value 0 to $node_{N-1}$. Every $node_i$ keeps the token until it becomes passive; it then forwards the token to $node_{i-1}$ increasing the token value by c.

– ii – Every node and also the token has a colour (initially all white). When a node receives a message, the node turns black. When a node forwards the token, the node turns white. If a black machine forwards the token, the token turns black; otherwise the token keeps its colour.

– iii – When $node_0$ receives the token again, it can conclude termination, if

- $node_0$ is passive and white,
- the token is white, and
- the sum of the token value and c is 0.

Otherwise, $node_0$ may start a new probe.

A formalisation of this algorithm in given in ([226]), and is given in Appendix I, page 346.

2.2.2 Wave-Based Solutions

Applications of the algorithms discussed above require that communication channels are bidirectional; for each basic message sent from *p* to *q* a signal must be sent from *q* to *p*. The average message complexity equals the worst case complexity; each execution

[1]Here *node* is used in place of *process* to include other processing entities in general.

requires one signal message per basic message, and in the case of the Shavit-Francez algorithm, exactly one wave execution [226].

Wave based algorithms are based on the repeated execution of a *wave algorithm*; at the end of each wave, either termination is detected, or a new wave is started. Termination is detected if a local condition turns out to be satisfied in each process [226].

Dijkstra-Feijen-Van Gasteren [77] is an example of a wave based algorithm. It detects termination of a basic computation using *synchronous* message passing.

But the synchronous message passing assumed in that algorithm is a serious limitation on its general application, hence several generalisation of it to computations with asynchronous message have since been proposed , e.g. Safra's algorithm which introduces message counting, counting messages sent and received in order to establish that no messages are under way [160] has similarly introduced an algorithm based on vector counting but which maintains a separate count for each destination.

It is worth noting that an alternative to maintaining message counts is to use acknowledgements. [177] has proposed a variation of the Dijkstra-Feijen-Van Gasteren algorithm to use acknowledgements, though the resulting algorithm does not offer an improvement on the Shavit-Franchez algorithm.

2.2.3 Other approaches to termination detection

An alternative view of termination detection algorithms research is to consider a number of approaches in the existing literature for developing algorithms for the termination detection problem and identify a wider range of categories. These are discussed extensively in [4] and summarised here.

1. Using *distributed snapshots*. In this approach the fact that a consistent snapshot of a distributed system captures stable properties is used, coupled with the fact

that termination of a distributed computation is a stable property.

It follows therefore that if a consistent snapshot of a distributed computation is taken after the distributed computation has terminated, the snapshot will capture the termination of a computation. Algorithms using this approach often assume that there is a logical bidirectional communication channel between every pair of processes. Communication channels are assumed to be reliable but non-FIFO and message delay is assumed arbitrary but finite.[46] and [116] discuss using distributed snapshots for termination detection.

2. Using *weight throwing* . In this approach there is a controlling process. A communication channel exists between each process and the controlling process. All processes start off in the idle state and are assigned a weight of zero, whilst the controller process is assigned weight of one. The computation starts with the controlling process sending a basic message to one of the processes. That process becomes active and the computation starts. Weights are bounded between zero and one , i.e. weight W $(0 < W < 1)$. When a process sends a message it sends a part of its weight in the message. On receiving a message a process adds the weight received in the message to its weight. Thus the sum of weights on all the processes and on the message in transit is always one. On becoming passive a process sends its weight to the controlling agent in a control message. The controller add this to its weight. If its weight becomes one, the controller concludes termination. [161] and [117] discuss algorithms based on weight throwing.

3. Using spanning tree. Assuming N processes $P_i, 0 \leq i \leq N$, the processes are modelled as nodes i, $0 \leq i \leq N$ on a fixed connected unidirected graph. The edges of the graph represent the communication channels through which a process sends messages to neighboring processes in the graph. The algorithm uses a fixed spanning tree of the graph with process P_0 at its root which is responsible

for termination detection. This process communicates with other processes to determine their states. Messages used are called *signals*. All leaf nodes report to their parents if they are terminated. A parent node will similarly report to its parent when it has completed processing and all of its immediate children have terminated and so on. The root concludes that termination has occurred, if it has terminated and all of its immediate children have also terminated.

4. *Message optimal*. Algorithms using this approach attempt to optimise and reduce inefficiencies in message complexity when concluding termination, for example in spanning tree based algorithms. [143] discusses such a message optimal algorithm, using a network represented by $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges or communication links. The communication links are bidirectional and exhibit FIFO property.The algorithm assumes the existence of a leader and a spanning tree in a network

5. Using *atomic computation model*. In the atomic model a process may at any time take any message from its incoming communication channels immediately change its internal state *and* at the same instant send out zero or more messages. All local actions at a process are performed in zero time, therefore there is no need to consider process states when performing termination detection. In the atomic model a distributed computation has terminated at time t if at this instant all communication channel are empty. This is because execution of an internal action at a process is instantaneous. To find out if there are any messages in transit, various *message counting* methods are normally used. This include , naive counting, four counter methods, vector counters, channel counters [4] In this model a dedicated process, P_1, the initiator determines if the distributed computation has terminated. The initiator starts termination detection by sending control messages directly or indirectly to all other processes. [160] has devel-

oped a number of algorithms based on the atomic model.

6. *Fault tolerant* methods assumes processes may fail, particularly fail in a fail-stop manner. Algorithms here detect termination in this environment. For example based on the weight throwing scheme a scheme called flow detecting scheme is developed by [48] to derive a fault tolerant termination detection algorithm.

Some selected with some optimisations and robustness considerations are presented in Appendix I, page 348.

In addition to the above there are also attempts to develop a general computing model for termination detection. An example of such work is discussed in detail in [37]. The next section provides a brief summary of concepts discussed and introduced there.

2.3 A general computing model and termination detection

[37] introduces a general distributed computation model, termination definitions, some terminology and predicates relating to termination detection and finally algorithms for the given termination definitions.

So far the assumption has been that the reception of a single message is enough to activate a passive process. In the general model introduced by [37], a passive process does not necessarily become active on the receipt of a message, instead a *condition of activation* of a passive process is more general and a passive process requires a set of messages to become active. This requirement is defined over a set DS_i of processes from which a passive process P_i is expecting messages. The set DS_i associated with a passive process P_i is called a dependent set of P_i. A passive process becomes active only when its activation condition is fulfilled.

The Communication model: A distributed application program (whose execution is traditionally called the underlying computation) is composed of a finite set P of processes P_i, i = 1, .., n, interconnected by unidirectional transmission channels; the channel C_{ij} links the sender P_i to the receiver P_j. Processes communicate only by exchanging messages through channels; there is neither *common memory* nor a *global clock, [37]*.

Communication is *asynchronous* in the following sense:

1. A sender sends a message to a channel (which then has responsibility for its delivery) and then the sender immediately continues its own execution;

2. Channels do not necessarily obey the FIFO (first in first out) rule, but they are reliable (no loss, no corruption, no duplication, no spurious messages);

3. Channel transfers (carries) a message to its destination process, the receiver puts it in its local buffer: the message has then *arrived*. The arrived message can then be consumed provided that its receiver has been activated, i.e. when the request of receiver has been fulfilled;

4. The transfer delay (time elapsed between sending and arrival of a message) is finite but unpredictable.

The process model: in addition to the discussed process model, there is a further requirement expressed by an *activation condition* (see below) defined over the set DS_i of processes from which a passive process P_i is expecting messages [37]. The set DS_i associated with a passive process P_i is called dependent set of P_i. A passive process can only become active when its activation condition is fulfilled. If such an activation is realized as soon as the activation condition is fulfilled (i.e. without any additional delay w.r.t the activation condition fulfillment), this constitutes instantaneous activation.

A passive process that has terminated its computation is said to be *individually terminated*, its dependent set is empty and therefore it can never be activated [37].

Request models: Formulation of activation conditions strictly depends on the request model considered.

1. **AND model:-** In this model a passive process P$_i$ can be activated when a message from *every* process P$_j$ belonging to DS$_i$ has arrived. It models receive statement that is atomic on several messages [37].

2. **OR model:-** In the OR model, a passive process P$_i$ can be activated when a message from *any* process P$_j$ belonging to DS$_i$ has arrived. It models classical non-deterministic receive constructs [37].

3. Other more complex models such as *OR-AND*, *Basic k out of n* and *Disjunctive k out of n* models are presented in [37].

In order to abstract the activation condition of a passive process P$_i$, a predicate $fulfilled_A$ can be considered, where **A** is a subset of P, the set of all processes. Predicate $fulfilled_A$ is true if and only if messages arrived (not yet consumed) from all processes belonging to the set **A** are sufficient to activate process P$_j$. The following monotonicity property is valid: if $\mathbf{X} \subseteq \mathbf{Y}$ and $fulfilled_X$ is *true*, then $fulfilled_Y$ is also true [37].

Termination definitions: The following notations are introduced to formally define terminations of distributed computations. The notation used here is introduced in [37].

1. $passive_i$: true *iff* P$_i$ is passive;

2. *empty* (j, i) : true *iff* all messages sent by P$_j$ to P$_i$ have arrived at P$_i$; the messages not yet consumed by P$_i$ are in its local buffer;

3. arr_i (*j*) : true *iff* a message from P_j to P_i has arrived and has not yet been consumed by P_i;

4. arr_i = { processes P_j such that arr_i (*j*) };

5. ne_i = {processes P_j such that ¬ *empty* (*j*, *i*) }.

Dynamic termination: The set P of processes is said to be *dynamically terminated* at some time if and only if the predicate *Dterm* is true at this moment, where:

$Dterm \equiv \forall \mathrm{P}_i \in \mathrm{P}$: $passive_i \wedge \neg fulfilled_i$ ($\mathrm{arr}_i \cup \mathrm{ne}_i$) [37].

This notion of termination means that no more activity is possible from processes, though messages of the underlying computation can still be in transit (represented by possibly non empty sets ne_i in the predicate). This definition is interesting for *early* detection of termination as it allows to conclude a computation is terminated even if some of its messages have not yet arrived [37]. It can be shown that once true, the predicate *Dterm* remains true, thus dynamic termination is a stable property [37].

Static Termination The set P of processes is said to be *statically terminated* at some time if and only if the following predicate is true at this moment:

$Sterm \equiv \forall \mathrm{P}_i \in \mathrm{P}$: $passive_i \wedge (\mathrm{ne}_i = \emptyset) \wedge \neg fulfilled_i$ (arr_i) [37].

For this predicate to be true, channels must be empty and processes cannot be activated. Thus this definition is based on the state of both channels and processes. When compared to *Dterm*, the predicate *Sterm* correspond to "late" detection as, additionally, channels must be empty.

[37] discusses a number of theorems related to static termination and outlines their proofs, for example *Dterm* $\mapsto$ *Sterm* (leads-to relation over the predicates).

Given this model, static and dynamic termination detection can be discussed, this discussion is given in Appendix I, page 350 with some illustrations.

2.4 A contemporary taxonomy for distributed termination detection algorithms

[159] provides a more complete and detailed taxonomy for distributed termination detection algorithms, partitioning the algorithms according to the following eight classification categories;

1. The *algorithm type* . This considers the general action of the algorithm. For example most common method for constructing DTD algorithms is to consider creation of a wave algorithm.

2. The required *network topology*. Many DTD algorithms assume a particular network topology for the nodes to allow correct and efficient definition of the algorithm. Hamiltonian cycles, trees,spanning trees, rings etc. are often assumed.

3. The *algorithm symmetry*. If each process executes an identical algorithm, and no process is distinguished from others for any purpose, then the DTD algorithm is considered symmetric.

4. The required *process knowledge*. Some DTD algorithms can assume that the process have knowledge of the system initially. An assumption can be made for example about the static size of the network. It can be observed that given that this knowledge is required at compile time, this makes the particular algorithm less general and restricts the network from changing.

5. The *communication protocol*. Protocols can be assumed to be synchronous or asynchronous. Early DTD algorithms were based on communicating sequential

processes [109]. CSP is an asynchronous protocol and the resulting protocol were elegant, e.g. [92].

6. The communication *channel behavior*. The communication channel can be considered to be first-in first-out (FIFO) or non-FIFO. Algorithms assuming FIFO are easier to construct e.g. [168]. FIFO channel can be achieved for example with a network protocol, which can guarantee that eventually messages reach an application in FIFO order. On the other hand a non-FIFO protocol is more general as it can work with both type of channels, e.g. [78, 160].

7. The *message optimality*. It can be shown that there is a worst case lower bound on the number of control messages used by a DTD algorithm, e.g. [45]. This bound means that for each message sent in the basic computation, there is a constant number of control messages to determine when termination has occurred.

8. *Fault tolerance*. This is a non-functional requirement that the algorithm is robust to failures of the network and individual nodes, important in distributed systems.

This taxonomy, [2] and its elements are depicted in Figure 2.1. We have also extended and updated this taxonomy to reflect and incorporate algorithms that have since been subsequently developed.

[2]Or more precisely, the set of properties by which a taxonomy can be developed.

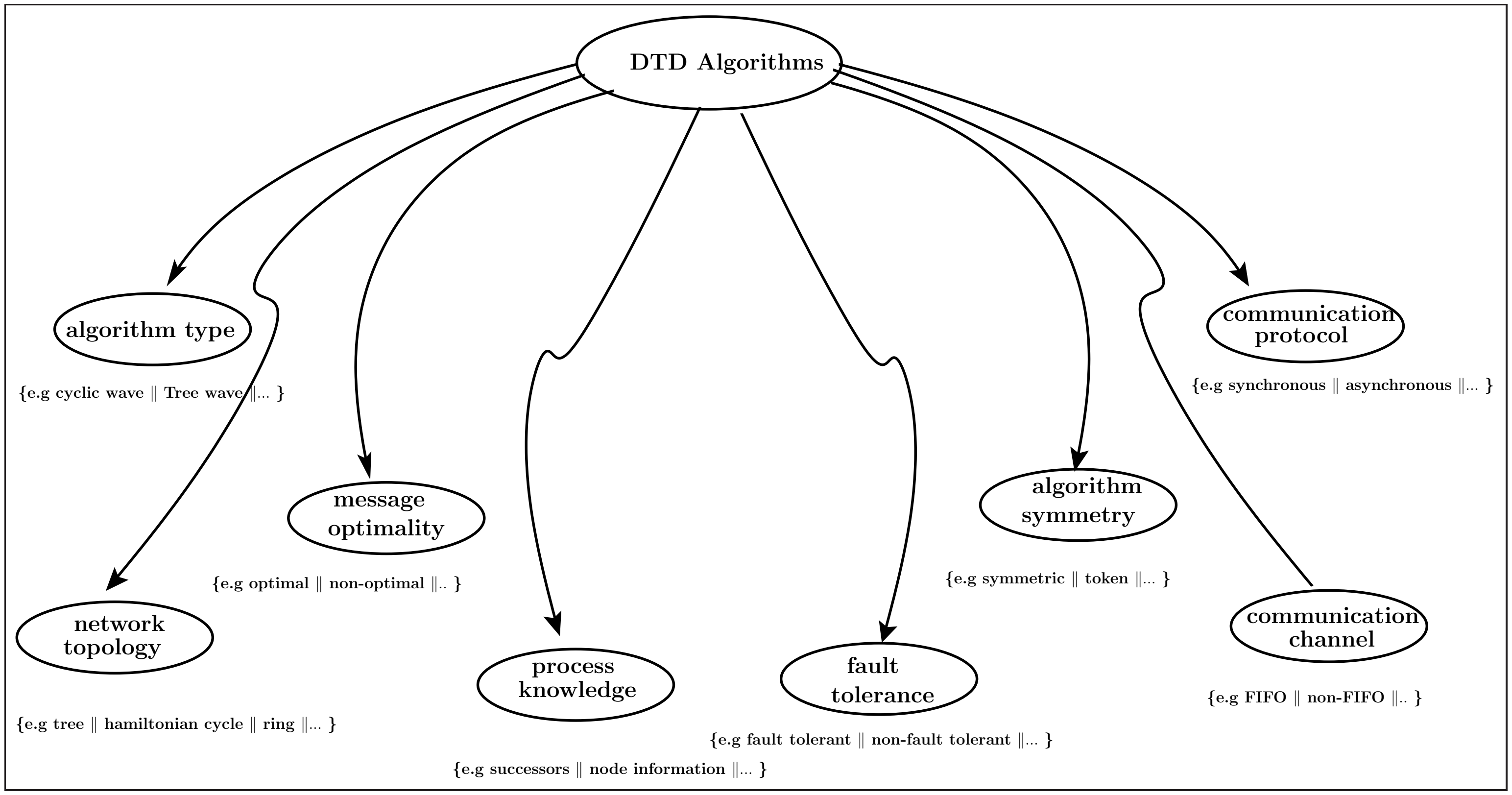

DTD Algorithms
algorithm type
{e.g cyclic wave ‖ Tree wave ‖... }
network topology
{e.g tree ‖ hamiltonian cycle ‖ ring ‖... }
message optimality
{e.g optimal ‖ non-optimal ‖.. }
process knowledge
{e.g successors ‖ node information ‖... }
fault tolerance
{e.g fault tolerant ‖ non-fault tolerant ‖... }
algorithm symmetry
{e.g symmetric ‖ token ‖... }
communication protocol
{e.g synchronous ‖ asynchronous ‖... }
communication channel
{e.g FIFO ‖ non-FIFO ‖.. }

Figure 2.1: A set of properties for Matocha's taxonomy of distributed termination detection algorithms

Table 2.1 gives an example taxonomy with algorithms arranged by *algorithm type*.

Algorithm	Cyclic wave	Tree wave	General wave	Non-repetitive wave	Parental responsibility	Credit recovery	Other
(Francez,1980)		✓					
(Dijkstra & Scholten,1980)							
(Francez et. al, 1981)	✓						
(Misra & Chandy, 1982)					✓		
(Chandy &Misra, 1985)			✓				
(Szymaski et. al, 1985)		✓					
(Mattern 1987)	✓						
(Muller, 1987)	✓						
(Huang, 1988)				✓			
(Mattern, 1989)							✓
(Vankatesan, 1989)							✓
(Lai et al.,1992)							✓
(Wang and Mayo, 2004)						✓	

Table 2.1: DTD algorithms and their associated type, adapted from [159]

Appendix I from page 356 presents recent research activity and the updated taxonomy in the rest of the tables from tables (Table I.1 through to I.7) for the other categories of the taxonomy.

A note on evaluating the performance of DTD algorithms Regarding performance analysis and measurement of DTD algorithms, a set of metrics can be considered. Three metrics are often deemed adequate [169], namely;

1. *Detection latency*. This measures the time elapsed between when the underlying computation terminates and when the termination algorithm actually announces termination. When computing this latency some algorithms e.g. [169] assume that message delay is at most one *unit*, similar assumptions are made in [144] and [48] when analysing detection latency of their algorithms. In addition message processing time is often deemed negligible.

2. *Message complexity*. This refers to the number of control messages exchanged by the termination detection algorithm in order to detect termination. Some algorithms as discussed above claim to be message optimal.

3. *Message-size complexity*. This means the size of control data as payload on the message by the termination detection algorithm.

2.5 Summary

Distributed termination detection, DTD, constitutes one of the basic and important problems in distributed computing. It is not easy to detect termination of a distributed computation because of the difficulty in obtaining a consistent global state in the absence of global clocks.

DTD has been observed to be related to other distributed computing problems such as global snapshot detection and distributed garbage collection. Indeed there is an important link between termination detection and garbage collection as first described by [227].

Many distributed algorithms have since been proposed to solve the problem after it was first conceived of by Dijkstra and Scholten when discussing diffusing computations.

This chapter has provided a detailed survey of the classical distributed termination detection problem as formulated for the communicating process model and the numerous solutions that have been since put forward. First the survey considered the classical algorithms and a taxonomy that partitioned the algorithms into *wave* and *probe* type algorithms.

section 2.4 introduced a contemporary taxonomy, due to Matocha, that gave eight criteria for assessing DTD algorithms. In tables I.1 through to I.7 in Appendix I we updated Matocha's 1998 taxonomy of distributed algorithms with recent algorithm proposals. It can be observed there that recent algorithms are by and large asynchronous and mostly do not make assumptions about message arrivals and are claimed to be message optimal. Most of the recent research activity has been in mobile systems and wireless networks.

It is worth noting that the algorithms discussed in this chapter assume a process model (by extension an object model). So while algorithms discussed here lay a good foundation for the study of termination detection in distributed systems in general, they often abstract away from the underlying computation so as to be as general as possible and hence may not be applicable directly to multiagent systems where an agent model and differing assumptions are made. These assumptions include flexibility of agent interactions, potential runtime dynamism of multiagent environments including runtime protocol discovery and potentially protocol specification revision , the use of semantically rich interaction protocols and most importantly the high levels of autonomy assumed which can manifest in agents being capable of providing additional information about their protocol execution at runtime which may aid termination detection process

Therefore, following this discussion, **Chapter** 3 next discusses, compares and contrasts agents and objects models of computation. It considers the assumptions therein in detail to motivate the need for revisiting problems encountered in distributed system research, like the distributed termination detection problem covered here.

CHAPTER 3

Distributed Systems vs Multiagent Systems

3.1 Introduction: On Agents and Multiagent Systems

Multiagent systems [220] research is multi-disciplinary and diverse. The theoretical foundations of the field can be seen in diverse areas spanning computer science, artificial intelligence, logic, philosophy and linguistics, game theory, economics and sociology.

On a practical and implementation level, work on multiagent systems and agent oriented software engineering can be viewed as an evolution of software engineering and multiagent systems are built on and are an evolution of distributed systems with emphasis on coordination and flexible interaction and open systems. Coordination in multiagent systems is primarily cast into a communication problem and effected through interaction protocols.

Below is a incomplete list of some key issues of research in multiagent systems and some are expanded on in the next paragraphs.

–i– Agent communication languages [141] and interaction protocols [120].

–ii– Organisations [93, 27] electronic institutions and markets [75, 87].

–iii– Multiagent coordination [126, 80, 50, 25].

–iv– Multiagent learning [6, 203].

–v– Negotiation [22].

–vi– Agent foundations, theories, social semantics, commitments [252], social order [53], roles [132] and autonomy [179], norms [54, 151].

–vii– Agent oriented software engineering [239, 253] multiagent systems engineering methodologies [36].

–viii– Agent technologies , languages and platforms [30].

–ix– Applications [127].

Agenthood Agents and Agent-orient programming are discussed in [209, 249], where agents and multiagent systems are proposed as candidate tools for managing the complexity that is inherent in software systems. In evaluating agent based solutions, common pitfalls to be considered in agent oriented development are highlighted in [248]. Multiagent systems research has not evolved in isolation but is closely related to other areas such distributed artificial intelligence [29, 175].

Agents *should* ideally exhibit desirable characteristics such as autonomy but there are some reservations [240] as to whether this is adequately captured and translated to real systems in the current state of the art, leading to research in efforts in computational autonomy [179]. Related to this is research on agent roles [73].

Agent communication languages Agents are distinguishable by their use of rich agent communication languages [141, 135] with communicative actions (speech acts [202]) [51] and defined semantics [140]. These semantics ideally should be verifiable as discussed in [246]. Furthermore, regarding agent communication languages, there has been also some efforts to define social semantics [213].

Agent interactions Agents exist in societies called multiagent systems [247, 210] and engage in collaborative (cooperative) or competitive interaction in achieving their joint or individual goals. Agents can be given strategies and protocols to engage in negotiations [22] to achieve their goal. These negotiations can possibly involve multiple issues, sometimes under time constraints [88]. Agents also can participate in auctions [133, 12] in electronic marketplaces. In some agents theories, agents can engage in dialectic interactions such as argumentation [201] to resolve conflicts in there knowledge [224] or belief revision [154] for example.

Agent theories There are numerous parallel strands of research in agent theories [249]. In some theories agents can be ascribed high level mentalistic notions of beliefs, desires and intentions [196], and have social semantics like commitments defined.

In practice there are various ways of implementing agents, e.g. logic based agents, and it has been the case that agents can be realised with the object paradigm (though with limitations [42]), and there are some views of agents as active objects [99].

3.2 Multiagent systems and Distributed systems

The research areas of multiagent systems and distributed systems overlap. Multiagent systems are inherently distributed systems, and distributed systems are platforms for supporting multiagent systems.

The following represent the widely accepted notions about agents (multiagent systems) and distributed systems;

1. Agents are generally considered to be *autonomous* (i.e., independent, not-controllable, in theory at least), *reactive* (i.e., responding to events), *pro-active* (i.e., initiating actions of their own volition), and *social* (i.e., communicative). Sometimes a

stronger notion is added that of beliefs, desired, intentions for example. Agents vary in their abilities; e.g. they can be static or mobile, or may or may not be intelligent. Each agent may have its own task and/or role. Agents and multiagent systems are used as a metaphor to model complex distributed processes.

2. A distributed system is considered to be a collection of independent systems that appear to the users of a system as a single system, i.e. *transparency* is often a key element. Processes and/or data can (or cannot) move from host to host, share information, etc.

There are a number of areas relevant to both distributed systems and multiagent systems that can be discussed to draw parallels.

Table 3.1 gives examples and compares and contrast various aspects of distributed systems and multiagent systems, showing what is known and well understood in both areas.

Feature	Distributed System	Multiagent system
mobility	mainly no	yes
Reliability	mostly yes	mostly no
Availability	mostly yes	mostly yes
Communication	simple	complex
Protocols	syntax-based (e.g HTTP)	semantic-based (e.g.FIPA)
Automatic Garbage Collection	yes	manual
Termination Detection	yes	not well studied

Table 3.1: Comparing and contrasting various features of multiagent and distributed systems. Showing

The current state of the debate on agents is summarised in [239], and states that the debate seems to converge to the consensus below, quote;

–i– The concept of agents is significantly different from the concept of objects in that it allows for a qualitatively different perspective of complex systems and their development and,

–ii– there is room for both the agent concept and the object concept because they are concerned with different levels of computational abstraction.

In our research we concur with this view and seek to consider some pragmatic and practical concerns that emerge as we make a progression from developing distributed systems based on relatively simple passive objects to building infrastructures for agents.

In particular we propose the argument below to motivate and highlight why some traditional problems in distributed systems like the termination detection problem may need further consideration within agent computational model assumed in multiagent systems. We consider autonomy, flexibility of interactions and dynamism is multiagent environment as relevant properties, i.e.

– i – Because agents are autonomous, regarding termination detection, agents may offer additional information about the execution of their protocols to facilitate the termination detection process for example.

– ii – Regarding flexibility and dynamism, consider the flexibility of interactions in multiagent systems, in particular consider a multiagent society with provisions for dynamism in protocols, i.e. in such environments;

- There could be support for dynamic execution of coordination protocols as proposed by [31], i.e. where the role an agent intends to hold within a protocol can be played without the need of prior knowledge.
- There could be an infrastructure for dynamic protocol specification as discussed in [14], an infrastructure that accommodates revision of protocol specifications during execution in situations where there is such a strong

requirement. This approach can be contrasted with the traditional one where specification of protocols has largely been considered as a design-time activity.

- There could be infrastructure support for runtime protocol discovery in general.

We propose that given these points, there is a concrete case for considering further how issues like termination detection can be addressed within multiagent systems environments.

3.3 Summary

This chapter has given a brief overview and account of the areas of multiagent systems and distributed systems by considering the underlying models of agents and objects. The current consensus is to view the notion of agents and objects not as competing but occupying different spheres and represent different levels of abstraction and therefore can coexist. With this background, we propose that there is necessity to consider some aspects of distributed systems research in light of the multiagent requirements and the agent model.

The reason for this is that while a multiagent system is a distributed system, there is emphasis on coordination, flexible interaction and higher degree of autonomy of entities and dynamism in environments. We propose to consider one aspect, namely a mechanism that detects termination of agents, by considering interaction protocols that are used by agents to flexibly coordinate. We can exploit this research to build on an agent management infrastructure that can culminate in the future with a realisation of an automatic timely mechanisms for high level tasks like society wide garbage collection.

Having identified interaction protocols as a starting point, the next chapter considers evolution of research in protocols, leading to the current state of the art in the area of agents interaction protocols and to serve as basis for the assumptions we make regarding protocols for subsequent chapters. These assumptions were introduced in Chapter 1 page 7.

CHAPTER 4

From computer protocols to agent interaction protocols

This chapter discusses interaction protocols as used in multiagent systems. First it provides a background by considering evolution of protocols, how protocols are specified and implemented using current methodologies . This thesis is not about protocol engineering or formal methods used therein for the development and verification of protocols. The purpose of the chapter is to solely provide some context and a backdrop for the discussion of the use of protocols in this thesis and also to give a basis for the choice we made on the use of finite state machines in representing protocols. To that end the chapter can be skipped without consequence to the subsequent chapters apart from noting the assumptions we make about the model of protocols we adopt as first highlighted in Chapter 1 in page 7.

4.1 Introduction

A distributed systems centric view considers protocols as sets of rules that govern the interaction of concurrent processes in distributed systems. Protocol design is therefore closely related and often discussed in the context various established fields, such as operating systems, computer networks, data transmission, and data communications [111].

There are a number of challenges regarding protocol engineering i.e. requirements, specification, validation verification of protocols, software engineering challenges that also face other systems . For example, assuming a protocol designer is capable of capturing and understanding the full set of requirements, then in conceptualising and designing a protocol that meets its requirements, a language that has precise, unambiguous semantics is needed in order to capture the protocol. Such a language is referred to as a formal language [115].

For this, a large body of work exists in the formal languages area. The first task in engineering a protocol is then that of choosing an appropriate language to describe the protocol.

This chapter begins by giving a general overview of protocols, then introduces some formal models and examples of formal languages for the specification of protocols. The chapter then proceeds to discuss the state of the art in protocol engineering in multiagent systems where it is shown that by and large, the current work builds on the work done in the wider area of computer protocols. The chapter concludes by discussing what is regarded as current challenges in engineering protocols for multiagent systems.

4.2 On computer protocols

The wider subject of computer protocols is discussed in detail in [111], where a historical account is given, together with fundamental challenges facing protocol designers in designing and analyzing protocols that formalize interactions in distributed systems.

Regarding validation and verification of protocols, various formal methods have been proposed , for example verification of protocols using model checking , e.g. SPIN is covered in [112, 113] and [214, 32, 199].

4.3 Specifying Protocols

Protocols have been studied extensively in relation to concurrent systems, and the behavior of concurrent systems is usually modelled as a sequence of states or actions, or both. A specification of a protocol, i.e. what the protocol is supposed to do consists of the set of all possible behaviors, or sequences of states, considered to be correct. The problem at hand often cited, is to determine a language that is suitable for specifying a protocol in an implementation-independent way. In addition however, this language must allow to easily map the essential features of the protocol onto an implementation.

4.3.1 A formal model of protocol systems

One view is to consider a protocol as analogous to a language in that it consists of a *vocabulary* of messages, a precise *syntax* for encoding the messages, a *grammar* that defines the rules for composing and exchanging messages, and the *semantics* for interpreting the meaning of strings in the vocabulary. Just as a spoken language serves to convey an idea from one person to another, so a protocol provides some service based on exchange. Therefore a protocol specification can be considered a precise and unambiguous formulation of this language of exchange.

Furthermore if an assumption is made that the set of messages that can be exchanged is finite, the analogy between languages and protocols leads to very convenient, well-developed formalisms - formal languages and finite automata. A standard definition of a formal language is that of a set of strings of symbols from some one alphabet, where an alphabet is a finite set of symbols and is usually denoted as Σ [114]. Relating this to protocols, Σ is the set of messages that an entity can send or receive, including messages that come from say, the environment (such as the expiration of a timer, for example).

4.3.1.1 Finite State Machines

A Finite State Machine, also called a finite automaton [163, 111, 114], consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from Σ. For each input symbol there is exactly one transition out of each state, possibly a self-transition. The initial state, that can be denoted q_0, is the state at which the automaton starts, and there is a set of states called a final or accepting states.

Formally, an automaton is represented by a 5-tuple $(Q, \Sigma, \delta, q_0, T)$ where Q is a finite set of states, Σ is a finite input alphabet, q_0 is the initial state, T is the set of final (terminal) states, and δ is the transition function mapping. Given the current state q_n and an input σ, the transition relation $\delta\left(q_n, \sigma\right) \rightarrow q_{n+1}$ defines the next state.

Definitions 2 and 3 provide the standard formal definitions for a finite automaton and non-deterministic finite automaton.

Definition 2. *A finite automaton (FA) A is a tuple* $(Q, \Sigma, \delta, q_0, T)$, where

• Σ *is a* finite *input alphabet,*

• Q *is a finite set of states,*

• δ *is the (partial) next state function,* $\delta : Q \times \Sigma \rightarrow 2^Q$

• q_0 *represents the initial state and* T *defines the set of terminal states, i.e* $q_0 \in Q$, $T \subseteq Q$.

δ is usually described by a transition diagram.[111]. If q, $q^{'} \in Q$, $\sigma \in \Sigma$ and $q^{'} = \delta\left(q, \sigma\right)$, and σ is said to be an arc from q to q$^{'}$ and written $q \xrightarrow{\sigma} q^{'}$.

A number of classes of FA can be distinguished, for example deterministic [1]FA shown in Figure 4.1, *where* Q = { S1, S2}, q_0 = S1, T = { S2} and Σ = {1, 0}

[1]Other classes are Nondeterministic Finite Automata (NFA) and Nondeterministic Finite Automata with ε transitions (FND-ε or ε-NFA).

Definition 3. *Deterministic finite automaton*

A finite automaton is called deterministic if:

● *δ maps each state/input pair into at most one state, i.e.*

$\delta : Q \times \Sigma \rightarrow Q$

The definition for a finite automaton above does not provide a way of explicitly representing or manipulating variables other than by explicitly manipulating the state of the automaton. A notational convenience for separating a named set of variables V that are implicitly part of the state encoding yields a structure known as an extended finite state machine (EFSM) [113]. Formally, if V is a set of variables, each of which can assume a finite number of values, then the EFSM is the automaton given by (Q, V, Σ, δ, q_0).

Finite State Machines and protocols A common way of modelling protocols is by using communicating processes [108, 109] where each process is a finite automaton and the network of processes is connected via error-free, full-duplex FIFO channels [34].

The formal model of a finite state machine has been applied extensively to the study of communication protocols, (particularly specification and verification), since the very first publications for example in [3] where a pair of finite-state automata were used to model the transmitter-receiver protocol in a data communications system. Further early work is published in [68] [236, 35].

The finite state machine approach has also long been the method of choice in almost all formal modelling and validation techniques [223]. An introduction the theory of communicating finite state machines can be found also be found in [35].

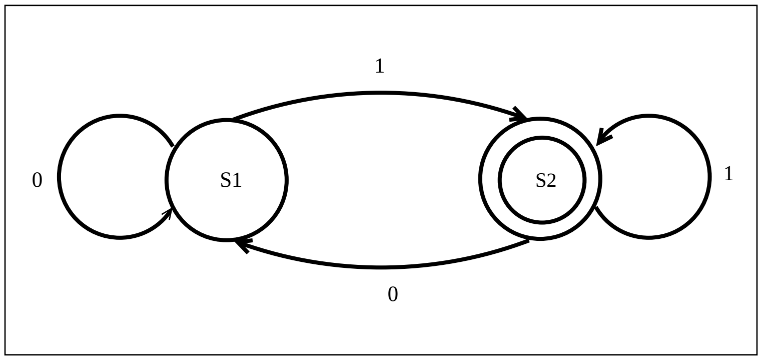


Figure 4.1: Example: Finite State Machine

4.3.1.2 Petri Nets

Many variations of the basic finite state machine model have been used for the analysis of protocol systems, both restrictions and extensions. It is observed that the restricted versions have the advantage, at least in principle, of a gain in analytical power. In the literature, it is cited that the extended versions have the advantage of a gain in modelling power [111]. Petri Nets are one such variant of finite state machine model. Petri nets were first described in [189], and surveys can be found in [188, 176, 33], and Petri Nets' modelling power and some extensions are discussed in [2]

Petri net structure Briefly, a petri net, PN is represented by a bipartite directed graph, with weighted arcs. In this graph, there are two kinds of nodes, namely *places* and *transitions*. The weighted arcs are either from a place to a transition or from a transition to a place. A place that has an outgoing arc to a transition t is called input place of t, a place that has an incoming arc from a transition t is called output place of t.

Formally, a Petri Net structure 5-tuple $PN = (P, T, F, W, M_o)$, a bipartite [2] graph

[2]A bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V i.e. , U and V are independent sets.

where:

Definition 4. *Petri net*

$P = \{p_1, p_1, ..., p_m\}$ *is a finite set of places.*

$T = \{t_1, t_1, ..., t_m\}$ *is a finite set of transitions,*

$F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs (i.e. a flow relation),*

$W : F \rightarrow \mathbb{N} - \{0\}$ *is a weight function which associates a nonzero natural value to each element of* F*. If no weight value is explicitly associated with a flow element, the default value 1 is assumed for the function,*

$M_o : P \rightarrow \mathbb{N} - \{0\}$ *is the initial marking,*

$P \cap T = \emptyset$ *(bipartite graph) and* $P \cup T \neq \emptyset$*.*

A petri-net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by N, and a petri net with a given initial marking is denoted (N, M_0)

Figure 4.2 gives an example of a petri net for a simplified communication network as discussed in [176] where also explanation of the notation is given.

Petri net dynamics The dynamics of a petri net is described by means of the concept of marking. A marking is a function that assigns to each place a nonnegative integer, called token; the initial state of the net is represented by the initial marking, denoted with M_0. From a graphic point of view, places are usually represented by circles, transitions by rectangles and marks by black dots into places. A place containing a token is said to be marked. Arcs are labelled with their weights and labels for unitary weight are usually omitted.

The dynamics of the net is described by moving tokens among places according to a particular *firing rule*:

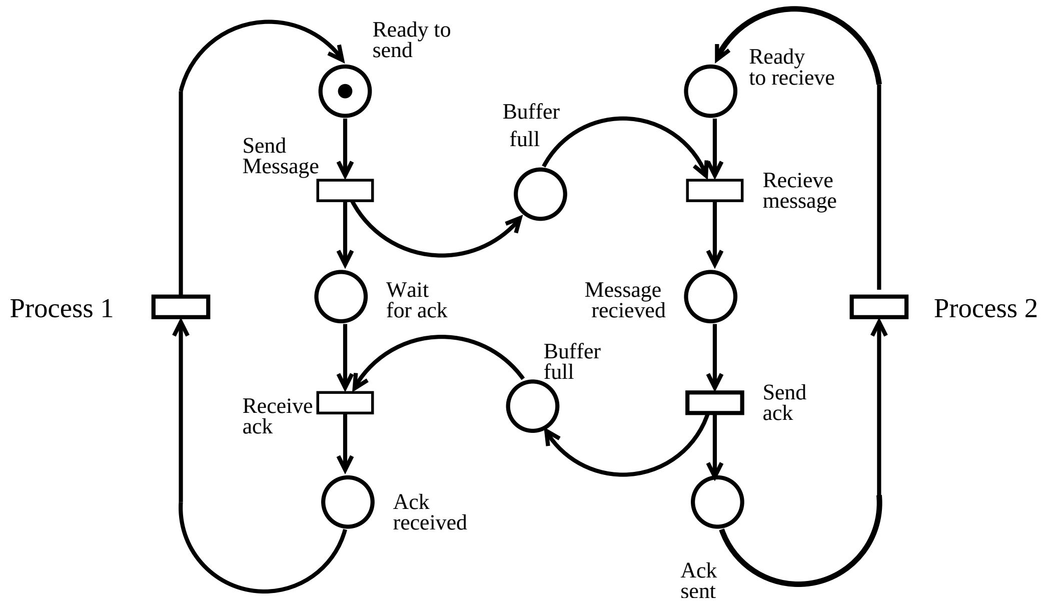


Figure 4.2: Petri net example: A simplified model of a communication protocol, adapted from [176]

1. a transition t is enabled to *fire* if each input place p of t is marked with at least w(p,t) tokens, where $w(p,t)$ is the weight of the arc from p to t.
2. a firing of an enabled transition t removes $w(p,t)$ tokens from each input place p of t, and adds $w(t,p)$ tokens to each output place of t.
3. the marking of the other places which are neither input nor output of t remains unchanged.

4.3.1.3 Petri Nets and protocols

[60] surveys the applicability of petri nets for protocol specification and validation and Figure 4.2 (adapted from [176]) shows graphically a very simple petri net model of a communication protocol.

4.3.2 Formal Languages for protocol specification

Since the models of computation usually considered are based on concurrent execution of sequential processes, the primary function of a protocol specification is to provide the legal execution sequences that each process can exhibit. Thus a very natural way to think about and specify protocols is by using a language that is based on concepts rooted in programming languages. In programming languages, as in the study of natural languages, syntax is separated from semantics. Language syntax is concerned with the structural aspects of the language, such as the symbols and the phrases used to relate symbols; syntactic analysis determines whether a program is legal. The semantics of a programming language, on the other hand, deals with the meaning of a program, i.e. what behavior is produced when the program statements are executed [115].

In order to create an unambiguous specification, one must use a language that has unambiguous semantics, so that a legal phrase in the language has a single interpretation. In a protocol context, this requires an underlying mathematical model of process execution, inter-process communication, and the system state space. A language having these properties is known as a formal description technique (FDT) [142].

A variety of languages have been proposed and developed for the purpose of describing protocols. Some of these languages were developed with the goal of augmenting informal descriptions in protocols published by standards committees, while others were developed as aids for the design and verification of protocols. These languages can be differentiated according to the model of computation, communication infrastructure, synchronization primitives, notion of time, and support for data types [232].

Several early attempts at developing a language formalizing a protocol description [15, 11, 10] gave birth to three parallel standardization efforts by the International Standards Organization, ISO, and others. The standardisation effort resulted in three primary languages, namely below:

1. **Estelle** [72] is a second generation FDT. The underlying model is that of extended finite state machines (EFSM) that communicate by exchanging messages and by restricted sharing of some variables.

2. **SDL**:- The Specification and Description Language (SDL) was developed by the standards body CCITT. SDL is also based on an extended finite state machine framework and was designed specifically for the specification and design of telecommunications systems.

3. **LOTOS**:- The Language of Temporal Ordering Specifications [28] was developed by the ISO standards body and was passed as an international standard in early 1989. LOTOS is strongly based on Milner's calculus of communicating system (CCS) [167], with additional influence by Hoare's CSP [108]. It falls into a class of languages known as a *process algebra*, which can be characterized by, firstly, a wide use of equations and inequalities among process expressions, and by secondly, an exclusive use of synchronized communication as the means of interaction among system components.

[120] gives a quick overview of these formal languages techniques and earlier work and general treatment of formal protocol representation, specification and verification is given in [26, 164, 164, 223].

A comprehensive bibliography of protocol synthesis literature, i.e. attempts to formalize and automate the process of designing communications protocols, is given in [200].

4.4 Interaction protocols in Multi-Agent systems

Overview The previous section considered aspects of the general area of communication protocols. Communication is also key aspect in multiagent systems, allowing agents to exchange information to cooperate and to coordinate tasks. Typically, communication in multiagent systems is represented as protocols, a set of rules that guide interaction between several agents [121]. For a given state of the protocol, only a finite set of messages may be sent or received. So this leads to a classical view of an interaction protocol as captured in [25], where the roles that agents play are considered, and the interaction is described as a finite state machine where:

1. states identify global states of the protocol,
2. transitions represent messages that are labelled with a role identifier and a performative. For any transition,agents playing an associated role can send a message that uses the associated performative.

And to quote [25], then as such, interaction protocols *are* a *coordination model*, the coordination medium being the agent communication language, the ACL, and the coordination laws are expressed through the finite state machine that describes the protocol. In our work we also take this view that agents coordinate using interaction protocols.

In multiagent research, it is widely accepted that protocols are public (compare this to agent strategies that generate agent utterances , which are private)[198]. A common protocol ensures that all participants following it will coordinate meaningfully and expect certain responses from others. There are debates about this notion of common protocols [187], in particular, concerns about difficulties in attaining common protocol knowledge.

It is noted though that there are various ways;-

1. protocol can be dynamically pre-arranged upon entering an interaction,
2. protocol maybe coded in the agents,
3. agents may obtain a protocol from a repository of published protocols,
4. an institution may dictate the protocol.

So in our work we assume common protocol knowledge, in particular through points 1, 2 and 3.

4.4.1 Protocol engineering in multiagent systems

There are a number of parallel strands of research on protocols and interaction behaviour specification in multiagent systems. [190, 86, 120] discuss protocols and protocol engineering in multiagent environments and propose conceptual frameworks. The standard approach to agent interactions has been message oriented, with interactions defined by interaction protocols that give permissible sequences of messages. Examples of research activity include work done on enhancing existing methods discussed above, i.e. finite state machines, petri-nets. Other attempts consider deeper issues of agent communication languages and the use of conversation policies [98] , and conversation oriented approach to agent interactions [20].

There are some arguments that the message centric approach is limited especially regarding robustness and flexibility [244]. It is with these concerns that there are strands of research that consider a shift away from message centric view to the introduction of social semantics and consideration of higher level notions such as social commitments and development of *commitment machines*, CMs. [252].

There is also further research that consider dialectical approaches , advocating the use of dialogue-games. [162] surveys commitment-based and dialogue-based protocols.

It is worth noting however, that some of these approaches often assume some underlying model of agents, e.g. logical frameworks as it is the case for dialogue-based protocols, and commitment machines can be mapped [244] to BDI frameworks [196].

The next sections considers briefly some of these approaches in turn.

4.4.1.1 Finite state machine based protocols

As discussed above there is historical precedent to using finite state in modelling protocols. This extends to multi-agent interaction protocols [251].

An interaction protocol as an fsm will show states and transitions labelled with allowed messages. As an example consider a finite state machine representation of the Contract Net Protocol[3] [215] shown in Figure 4.3. In that figure, a and b represent *roles*, i.e. in the role a, an agent can send messages from the set $\{cfp, accept\_proposal, cancel, reject\_proposal\}$ to a group of agents each playing the role b. In the role b and agent can send messages from the set $\{propose, refuse, inform, cancel\}$. Indexing can be used in the protocol message labels for both roles, e.g. to capture the fact that a cfp message is broadcast by an agent playing role a to multiple agents in a group. We can write $a : cfp : b(i)$ and more generally if multiple instances of role a exist write $a(i) : cfp : b(i)$. This setup can be generalised to multiple roles.

Finite state machine based protocols and conversation models are predominantly used in multiagent system research. A justification is given in [57] where it is observed that it is mainly because the finite state machines have an established underlying formal model that supports structured design techniques and formal analysis and facilitates development, composition and reuse. Furthermore finite state machines are simple, intuitive, provide visual flow of action or communication and are sufficient for many

[3]Contract Net Protocol is a task allocation protocol that facilitates negotiation between bidders and an auctioneer in a Multi-Agent System to form a contract.

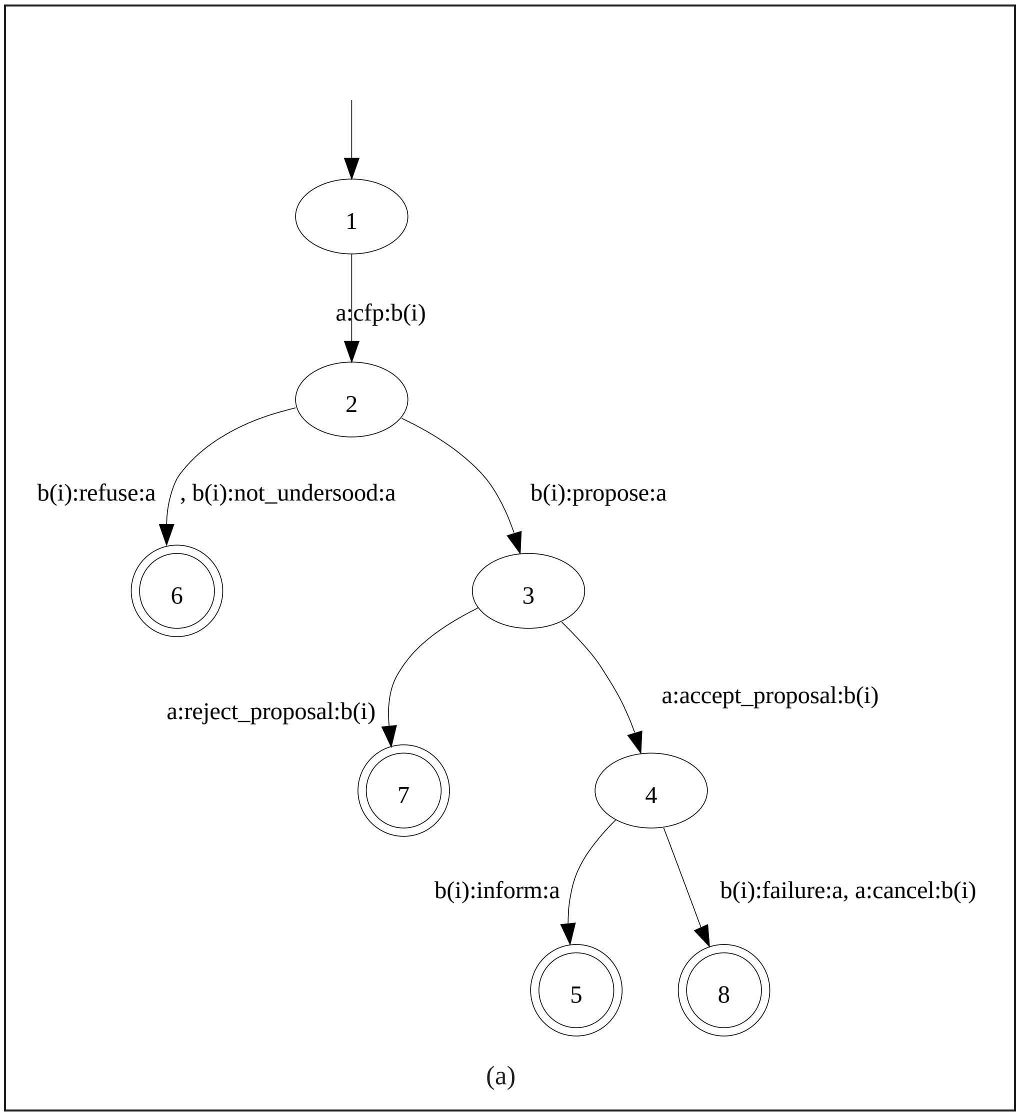


Figure 4.3: Contract Net protocol, showing messages and indexed agent roles for protocol participants.

sequential interaction.

4.4.1.2 Petri Nets and agent protocols

While finite state machine based protocols and conversation models are predominantly used, the main criticism has been that finite state machines are not adequately expressive to model interactions with degrees of concurrency.

Therefore there is some research directions that explore the use of petri-nets in modelling agent interactions, some work is reported in [102, 86].

Because the petri-net language is a generalisation of automata folmalism[4] then with appropriate transformations petri nets can be derived from finite transition systems [56] and reverse transformation also exists, called reachability analysis, is part of the definition of Petri Net. It generates a form of FSM labelled with Petri net transitions and called state graph.

Coloured Petri-nets [128] have recently been explored to represent agent interactions and related issues. [57] proposes the use of colored petri-nets as model underlying language for conversation specification. The motivation cited there for this is that;

1. While finite state machines are commonly used, they are not sufficient for complex agent interactions requiring concurrency.

2. Petri-net carry relative simplicity and graphical representation of Finite state Machines in addition supports greater expressive power to support concurrency.

Furthermore, a language, Protolingua [57] based on this model was investigated within the Jackal [58] agent development environment. For example [102] presents an analysis of existing Petri net representation approaches in terms of their scalability and appropriateness for different tasks.

[4]To express concurrency of events.

4.4.1.3 AUML

There is active research on the use of AUML [21, 150] in modelling agents interaction protocols, the rationale being that by aligning this work with the closest antecedent technology object oriented software development, there are benefits to be accrued, especially in the wide acceptance of agents [21], indeed there is a view in some research strands in Agent-oriented programming that multiagent systems can be considered extensions of object-oriented systems.

AUML is an extension of Unified Modelling Language, UML [5] [124] and there has been attempts to model agent protocols and interactions using it, examples can be seen in [136].

The motivation for the use of AUML has largely centered around the need for modelling methods and tools that supports a complete process lifecycle [136]. For example, the use of AUML specified interaction protocols in a prometheus [6] based designed tool [230] has been discussed in [185].

AUML has also been used by FIPA[7] to specify FIPA agent interaction protocols [1]. Regarding implementation, FIPA protocols implementations in multiagent tools has been largely as finite state machines. [8]

Figure 4.4 below shows an example of a FIPA contract-net protocol.

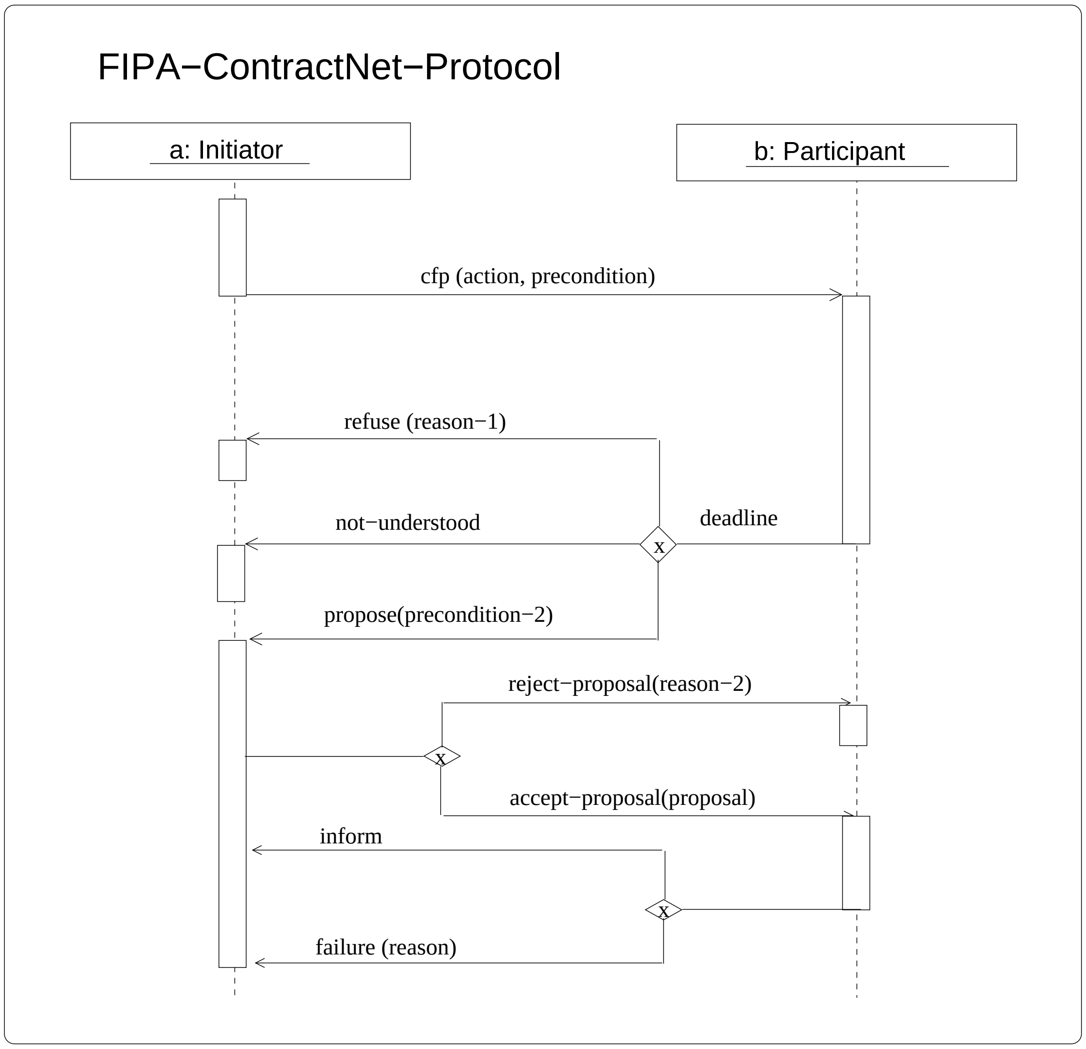
The AUML FIPA interaction protocol can be mapped to other formalisms, for example [39] provides a transformation of AUML diagrams to petri-nets to help define operational semantics of interaction protocols, i.e. the semantics of the AUML diagrams are defined through the semantics of petri-nets. Also [102] makes an argument for a

[5]Object Management Group.

[6]Prometheus is a software engineering methodology for designing agent systems.

[7]FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA has specified and defined semantics of FIPA-ACL, an agent communication language.

[8]Open source agent platform JADE uses finite state machine based "behaviours" for example.



(a)

Figure 4.4: FIPA contract-net protocol adapted from [1].

semi-automatic procedure for converting FIPA interaction protocol to their petri-net representations in search of a better representation of protocol features.

4.4.1.4 Multiagent conversation policies

Another view of agent interactions is to consider a conversational model [59] and to structure interactions as conversations [19] among agents and organise messages into appropriate contextual settings to provide a common guide to all agents. This is done by using conversation policies, *CPs*. A definition of a conversation policy is given in [98] and current research efforts discussed in [76].

Regarding implementation, while as observed in [162], the definition of conversation policies abstracts from any precise computational model, in practice CPs are modelled too as finite state machines and typically these models are called protocols.

Coloured petri nets [128] have been also used in conversation modelling as discussed in [59].

There have also been proposals for a conversational and coordination language as discussed in [18] and use of such a language in conversation oriented programming is described in [20].

It is worth noting though that conversation policies have limitations of their own. Particular challenges are identified by [162] to be in flexibility and specification of conversation policies.

4.4.1.5 Commitment-based and dialogue-based protocols

There is also active research directions in exploring the notion of commitments in modelling agent interaction protocols. Commitment are defined through commitment machines (CMs), where a commitment machine defines a range of possible interactions that start in *some state* (i.e. no initial state is designated as such, this to be contrasted with standard interaction protocols and finite state machines [244]).

Regarding implementation, a commitment machine is a *declarative* description of states and allowed transitions in a protocol, an example description for the a netbill protocol[9] [61], is given in [244].

However [251] has shown that given a commitment machine, a finite state machine equivalent representation can be synthesised automatically, and therefore one way to visualise the interactions that are possible with a given commitment machine, is to generate the finite state machine corresponding to the CM as demonstrated by [244].

Finally, there is also active research on the use of dialogue-game based protocols, and there have been proposals for dialogue-game based agent communication languages [41]. An extensive review of these new trends in ACLs and the use of commitment-based and dialogue-based protocols is given in [162]. Some recent ideas about designing and implementation of commitment-based interactions are given in [243] and [244].

4.4.2 Discussion

We observe that the various approaches discussed above though somewhat varied, can with appropriate transformations be converted to their under finite state machine equivalents without loss of information for the purposes of the work we want to do in this thesis on termination detection; Indications of possible existence of such transformations is due in part from the fact that some model are derivations of the finite state machine, and in this discussion for example;

1. [245] states that each commitment machine implicitly defines a corresponding

[9]NetBill is a system for micropayments for information goods on the Internet.

Finite state machine where the states of the FSM correspond to the states of the CM and the transitions are defined by the effects of the actions and [251] has shown that from a declarative description of states and allowed transitions in a protocol, i.e. a commitment machine, a finite state machine representation can be synthesised automatically.

2. AUML specified protocols can be converted to petri nets, and
3. Petri nets can be derived from transition systems [56, 186] and the reverse transformations are possible.
4. Conversation policies in practice are implemented as finite state machines [162]

Research Assumptions In our work we make the assumption that agents coordinate using interaction protocols, and that these interaction protocols are finite state machine based or can be reduced or transformed to finite state machine equivalent representations with preserving transformations. This is particularly useful in heterogenous environments where a common underlying representation is useful for interoperability. Considering the discussion above regarding various ways in which interactions are currently modelled, an assumption of a common underlying finite state machine representation and possibility of implementing transformations is not unreasonable.

To this end, our research makes assumptions that underlying protocols are based on the finite state machine model, and subsequent discussions e.g. in chapter 5, page 66, treats protocols as finite state machines, edge-labelled directed graphs.

4.5 Summary

This chapter has reviewed the evolution of computer protocols, their formal models and techniques for their specification. The chapter then discussed current practices in interaction protocols for multiagent systems, and has shown that current research in this area extends previous work on computer protocols and is progressing towards approaches like conversations policies and declarative descriptions such as commitment machines. Often though particular underlying models for agents are assumed in this approaches.

Various extensions to existing techniques has been proposed, for example the use of colored petri-nets and AUML in modelling interaction protocols. Petri-nets are considered to address the concurrency issues in communication, and AUML is considered to relate agent-oriented software engineering to the successful and widely accepted UML approach in object oriented systems so as to encourage the uptake of agent development.

But the finite state machine approach is widely used in modelling and implementing protocols in implemented multiagent systems. Furthermore, in the discussion of other approaches, it is apparent that with appropriate transformations, it is possible to derive finite state model equivalent representations, this observation we propose can help in accommodating heterogeneity in implemented systems.

So in this research we make an explicit assumption that coordination is achieved by the message-centric interaction protocols and that this protocols are based on finite state model as is predominantly the case.

With this background, **Chapter** 5 in **Part II** next considers a termination detection model for multiagent systems interactions.

PART II

TERMINATION DETECTION FOR AGENTS

CHAPTER 5

Termination Detection for Protocols

5.1 Introduction

We have discussed in the previous chapter, Chapter 4, section 4.4 that protocols represent the allowed interactions among communicating agents, and they regulate these interactions. They can also be viewed as specifications of these interactions as cited by [212]. Agents participate in different protocols by appropriately interacting with each other, for example, by responding to messages, performing actions in their domain, or updating their local states. Protocols can thus be taken as specifying policies that agents would follow with regard to their interactions with other agents. These policies would for example, determine the conditions under which a request will be acceded to or permissions issued or a statement believed [212].

We have also noted in the previous chapter that there are various approaches to specifying protocols and argued for adopting a finite state machine representation and considered some unified framework where with appropriate transformations a finite state machine representation may be derived from others.

Chapter 2 discussed extensive research in termination detection in distributed systems considering an underlying process model of computation. Under this computational model, the termination detection problem was discussed and various algorithms proposed over time were presented within some taxonomies.

Chapter 3 discussed an agent model of computation, contrasted it to the process or object model used in distributed systems research, and identified flexible interaction and coordination as some of the main considerations in the agent model, and also identified interaction protocols as a means to effect both.

The chapter also proposed that we could therefore consider looking closely at some research in distributed systems but within the agent model, for example termination detection, in light of the assumptions in the agent model. This in order to bring the benefits of the termination detection applications to multiagent systems infrastructures.

These applications were covered in chapter 2, page 16. There it was observed that as the terminated status is among stable states (consider as another example global communication deadlock) that should be known for system administration. Other applications are listed there.

Now also recall the example scenarios introduced briefly in section 1.5 cited where a termination detection mechanism can be exploited.

In all those scenarios, we can

–i– Consider agents executing a publicly visible behaviour specification in the form of public protocols, (while the agent strategies that generate agent responses themselves maybe private).

–ii– Furthermore we can assume that these behaviour specifications are in a form of finite state machines or if they can at least with appropriate transforms be translated to finite state machines in a unified framework proposed as discussed in the previous chapter.

–iii– As part of addressing the problem of termination, we can discuss the problem of determining when individual agents have reached *terminal configurations* in the protocols they are executing.

–iv– To study the problem of termination detection in multiagent system adequately will require consideration of representative cases in the space of interactions. e.g., one to one (client server like), many to one (auction like) and many to many (in general). We can consider a model for agent conversations, capture multiplicity of interactions and consider termination detection in such a model.

–v– Furthermore we can consider the issue of what additional information agents can avail to aid this process by observing that in multiagent systems, protocols are made public while individual strategies are maybe made private and that in practical implementations of multiagent systems, there is bound to be some restrictions on absolute agent autonomy, and provision of such additional information may be in line with conditions for participation in an agent society, as is the case with auctions for example.

5.2 Overview

This chapter is structured as follows;

Section 5.3 begins by giving basic definitions about *protocols*, *observables*, *termination paths*, *unique termination paths*, and defines and identifies the *shortest unique termination paths* as the minimal information that an agent can provide about its interaction protocol, (together with its interaction partners).

The section also sketches an off-line procedure to derive shortest unique paths given a protocol graph and presents an algorithm for this.

Section 5.4 presents a termination detection model, where we present a representation for the notion of a *conversation* , and model this as an interaction from an agent's perspective. The section also provides a model for branching conversation represented as *diffusing computation* tree, and provides a definition of a data structure, a conversation matrix, *c-matrix*, a structure that can be used by controllers, entities that oversee conversations. Given this model and definitions, the section then sketches a procedure for local termination of conversations and presents accompanying algorithms and some complexity analysis.

Section 5.5 considers possibilities for distribution, and discusses a distributed protocol for termination detection over a cluster of controllers, and sketches a procedure of such a protocol and gives some possible algorithms.

The section also provides preliminary evaluation of the protocol given the defined metrics for evaluating termination schemes. Detailed quantitative evaluation of one of these metrics, detection delays, is given in following chapters.

Then **Section** 5.5.2 in page 108, discusses how a termination detection mechanism may fit in within a generic multiagent systems management infrastructure.

Finally, **Section** 5.6provides a discussion and summary for this chapter.

5.3 Definitions

Consider following definitions about protocols;

Definition 5 (Protocol)**.** *A protocol is a tuple* $(S, \longmapsto, L, T)$*, where* S *is a set of states,* L *is a set of labels,* $\longmapsto \subseteq S \times L \times S$ *is a set of transitions and* $T \subseteq S$ *is set of terminal states, where* $T \neq \emptyset$ *and* $\forall t \in T \;\nexists s \in S, l \in L$ *such that* $s \neq t$ *and* $(t, l, s) \in \longmapsto$*. We will sometimes write* $s \overset{l}{\longmapsto} s'$ *instead of* $(s, l, s') \in \longmapsto$*. There is some state* $s \in S$ *designated as a start state.*

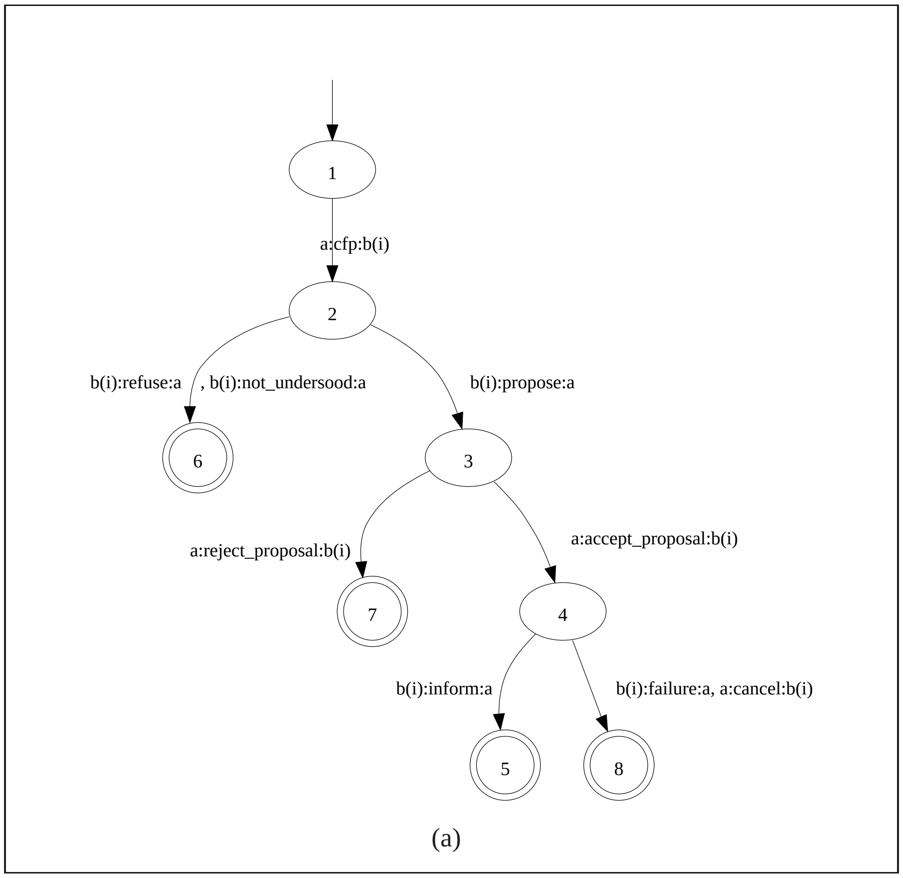


Figure 5.1: Contract Net protocol [215]. Showing roles in the protocol and with indexing is used to identify protocol participants.

Example 1 (CNP)**.** *Consider a Contract-Net protocol* [1]*, the protocol can be represented by the state transition system as shown in Figure 5.1 below. The protocol shown is being executed by two agents* [2]*. In this example*[3]
S = {1, 2, 3, 4, 5, 6, 7, 8}; T = {5, 6, 7, 8}; L = {cfp, propose...} and $\longmapsto$ *is as shown in the Figure 5.1.*

[1]An agent with a task to complete can solicit offers from other agents via a call for proposals, cfp, message.

[2]Messages are prefixed with agent identifiers

[3]Strictly speaking this is only a simple request protocol since it is defined as a one-to-one interaction. CNP degenerates to simple request protocol if there is only one bidder or task agent.

By executing a protocol,each agent participating in the protocol undergoes various internal state transitions. Each agent has a partial local view, i.e. interactions it engages in. The larger problem posed here is that of deriving a global view of system of interacting agents given individual agent partial local views, e.g. if say quiescence of the system is to be determined.

As part of addressing this problem, the discussion here centers on determining when agents have reached *terminal configurations*. The discussion also considers the issue of what additional information can agents avail to aid this process while preserving autonomy [4] and not introducing too much central control.

To detect termination of a protocol, we can define a *termination path*, i.e. a sequence of state transitions labelled by observable messages which end in a terminal state.

Definition 6 defines a *termination path*.

Definition 6 (Termination Path)**.** *Let $P = (S, \longmapsto, L, T)$ be a protocol, then a path p of length n is a sequence $(s_1, \ldots, s_n)$ where $s_i \in S$ for $1 \leq i \leq n$ and $s_{j-1} \overset{l_j}{\longmapsto} s_j$ for $1 < j \leq n$. The labels of path p are defined as a sequence $(l_2, \ldots, l_n)$. Furthermore, if $s_n \in T$, then p is a termination path.*

Example 2. *Given the protocol P in Figure 5.2 below, then for example, the path $p = (1, 2, 5)$ is a valid termination path with labels (b, c).*

Definition 7 (Observable States and Observables)**.** *Let $P = (S, \longmapsto, L, T)$ be a protocol, then an Observable state is a state $s_i \in S$ $s.t$ $\exists$ a unique path p, and $s_i \overset{*}{\longmapsto} s_n$ where $s_n \in T$ and $p \in TP$, where TP is a set of termination paths.*[5] *Observables are all the labels in path p.*

[4]In this preservation of autonomy, we mean within the rules, roles norms of the given society, and assume enforcement of compliance. It is generally accepted that to engineer MAS, autonomy may be constrained somewhat [184], our agents are constrained in that they do cannot make the decision not to provide this information.

[5]Termination paths are derived by a defined termination paths procedure.

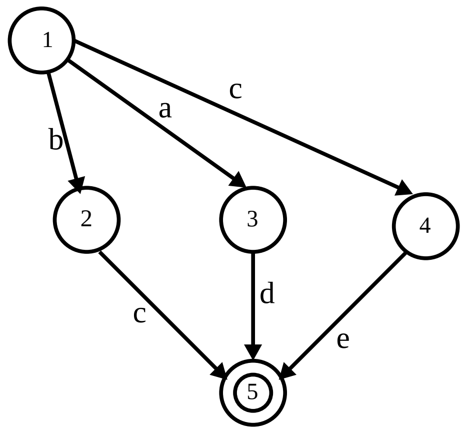


Figure 5.2: A protocol with shortest unique termination path.

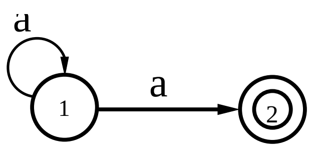


Figure 5.3: A protocol with no shortest unique termination path.

Definition 8 (Unique Termination Path)**.** *Let $P = (S, \longmapsto, L, T)$ be a protocol, then a termination path p with labels $(l_1, \ldots, l_n)$ is unique, if there is no path $p' \neq p$ with labels $(l_1, \ldots, l_n)$.*

Example 3. *Consider the protocol in Figure 5.2. The path $(2, 5)$ is a termination path, but as both paths $(1, 4)$ and $(2, 5)$ have labels (c) $(2, 5)$ is not a unique termination path. On the other hand $(1, 2, 5)$ is a unique termination path, as there is no other path with labels (b, c).*

Definition 9 (Shortest Unique Termination Paths, Observables)**.** *Let $P = (S, \longmapsto, L, T)$ be a protocol and TP the set of shortest unique termination paths, then the set of observables O is the union of all labels in any path $p \in TP$, i.e. $O = \bigcup O_i$, where $O_i = \{l_i | l_i \in (l_1, \ldots, l_n) \in TP\}$.*

Example 4. *In Figure 5.2 above, the set of observables is $O = \{$b, c, d, e$\}$. Note that*

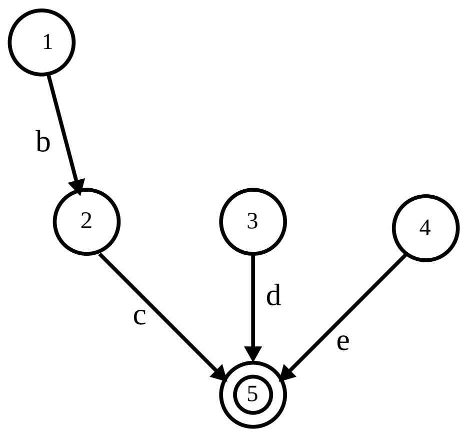


Figure 5.4: Shortest Unique Termination Paths

"a" is not element of the set O because the shortest unique path between states 1 and 5 is (3,5) with label "d".

Also Consider Figure 5.3. It is not always the case that there is a shortest unique termination path (e.g. $s \overset{a}{\longmapsto} s$*,* $s \overset{a}{\longmapsto} s'$*, where* s' *is a termination state, does not have a shortest unique termination path. The reason is that it contains a cycle. If we limit ourselves to directed acyclic graphs, then this problem does not occur.*

Minimal information Given the above definitions, the following statement can be made: Considering an arbitrary protocol (such as that given in Figure 5.2), the minimal information (sub-protocol) that an external entity (monitor) needs to keep to ascertain termination is the shortest unique termination paths of the protocol being executed, intuitively, if an observer is watching this protocol animation as messages are sent; shortest termination paths provide and answer to the question; *what are those messages or sequences of messages that if observed we know the protocol is as close to the terminal state as it can possibly be*?

For the example being considered here, these are depicted in Figure 5.4.

Procedure for deriving shortest termination paths A procedure for deriving termination paths given a protocol graph can be performed once and off-line on a given protocol or a set of protocols. Such a procedure can involve a graph traversal such as a modified *depth-first-search* to;

–i– Perform a reachability analysis[6],

–ii– Extract paths leading to terminal states,

–iii– and invoke a recursive mechanism to build up and check uniqueness of shortest paths.

One *such* procedure is sketched below and an algorithm presented in Algorithm 1 page 80. A concrete example for illustration is given in example 5 in page 78 for an arbitrary protocol graph depicted in Figure 5.6 in page 77.

Procedure for deriving shortest termination paths

1. From the start node $s \in S$ Perform reachability analysis and from every path p $\in P$ of valid paths leading to terminal state $t \in T$, extract all labels $l \in L$ and construct a set L_p of sequences $\{l_i\}$ of length *one* made out of the labels, i.e. $L_p = \{\{l_i\} \mid l_i \in L \wedge s_i \xmapsto{l_i} s_{i+1}, s_{i+1} \neq t\}$. Note that we insist on $s_{i+1} \neq t$ because labels l_i in the immediate neighborhood of the terminal states t will be used (see below) to construct another set L_t as input to the algorithm, and a test $L_p \bigcap L_t \neq \emptyset$ if true will mean that these $\{l_i\}$'s in L_t are not unique and not shortest termination paths and therefore need updating.

[6]Not in the strong sense of state exploration, but in the sense of picking a state and traversing paths to the final state.

2. Consider an index set $K = \{n \mid 1 < n < h\}$ where h is the height of the protocol graph, then let the set $A = \cup_{k \in K} A_k$ represent the set of sequences of all lengths representing transitions of any length. Starting from the root state $s \in S$, and from $\forall p \in P$, construct sets $A_k = \{\{l_1 \dots l_k\} \mid s_i \overset{l_1}{\longmapsto} s_2 \overset{l_K}{\dashrightarrow} s_n, s_n \neq t\ \}$ where each set A_k is a set of all sequences of length k

3. Starting from a terminal node $t \in T$, construct a set L_t of sequences $\{l_i\}$ of length *one* made out of the labels of the transitions in the immediate neighborhood of the terminal state t, i,e, L_t = $\{\{l_i\} \mid l_i \in L \wedge s_i \overset{l}{\longmapsto} t, t \in T\}$

4. Initialise the set of shortest termination paths TP to L_t, i.e. $TP = L_t$.

5. Check $L_p \bigcap L_t = \emptyset$

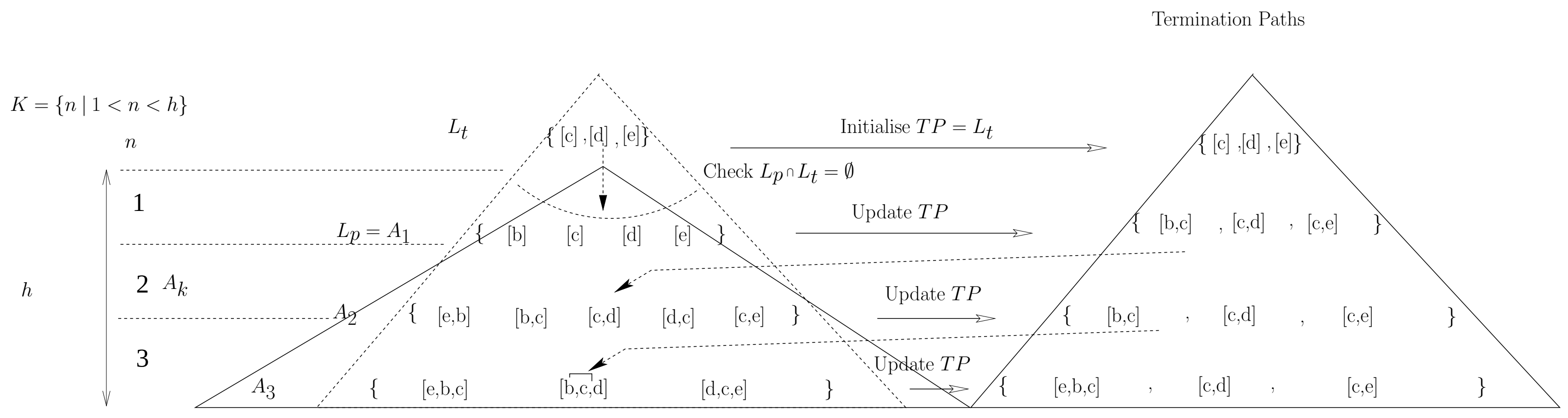
   - If true return TP as the set of minimal (shortest) termination paths.
   - Else $\forall \{l_i\} \in L_p \bigcap L_t$ update $\{l_i \in TP\}$ where $s_i \overset{l}{\longmapsto} t, s \in S, t \in T$ to include label l_{i-1} where $s_{i-1} \overset{l_{i-1}}{\longmapsto} s_i \overset{l_i}{\longmapsto} t$, i.e. update $\{l_i\}$ to $\{l_{i-1}, l_i\}$ in TP to include the next transition up that path $p \in P$.

6. Repeat for $(2 < n < h)$

   - $\forall \{l_{i-n-1} \dots l_i\} \in T_p$ if $\{l_{i-n-1} \dots l_j \subset \{l_k\} \in A_k\}$, where $\subset$ means *subsequence of*, then update $\{l_j\}$ by appending the next transition label up the path p to make this sequence unique, i.e. i.e. update $\{l_j\}$ to $\{l_{i-n}, l_{i-n-1} \dots l_i\}$ in TP.

7. The number of terminal states , T is $|T|$, therefore, the set of all termination paths for a protocol with multiple terminal states is, $\mathcal{TP} = \bigcup_{1 \leq n \leq |T|} TP_n$.

A graphical depiction of the trace of the shortest termination paths procedure is given in Figure 5.5 where the input as an arbitrary protocol given in Figure 5.6.



Termination Paths
$K = \{n \mid 1 < n < h\}$
n
h
1
2
3
A_k
L_t
{ [c] ,[d] , [e]}
Check $L_p \cap L_t = \emptyset$
Initialise $TP = L_t$
{ [c] ,[d] , [e]}
$L_p = A_1$
{ [b] [c] [d] [e] }
Update TP
{ [b,c] , [c,d] , [c,e] }
A_2
{ [e,b] [b,c] [c,d] [d,c] [c,e] }
Update TP
{ [b,c] , [c,d] , [c,e] }
A_3
{ [e,b,c] [b,c,d] [d,c,e] }
Update TP
{ [e,b,c] , [c,d] , [c,e] }

Figure 5.5: Showing an illustration of the algorithm for shortest termination paths running on a protocol given in example 5.6

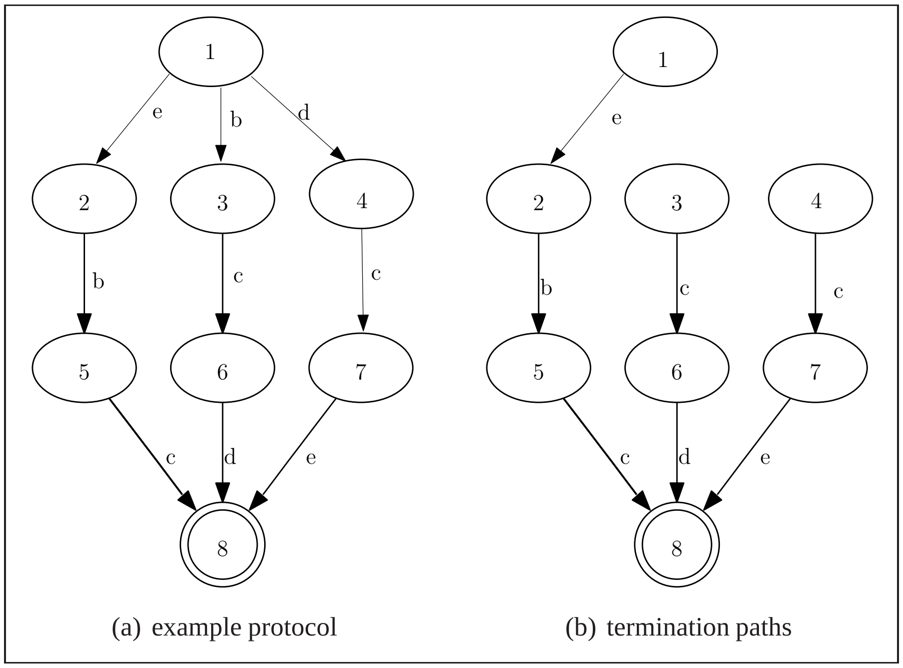


Figure 5.6: Showing and example protocol and termination paths extracted by Algorithm 1. The trace of the process is shown in Figure 5.5

Example 5. *For illustration, consider a protocol P shown in 5.6 (a) and the corresponding shortest termination paths shown in 5.6 (b) derived by the STP Algorithm 1 in page 80 whose trace is depicted in Figure 5.5. The following is an illustration of the steps.*

1. *step 1 From the protocol graph, extract paths $p \in P$ and construct set L_p of sequences.*

   - $P = \{p_1, p_2, p_3\}$ *where paths are :* $p_1 = (d, c, e)$, $p_2 = (b, c, d)$, $p_3 = (e, b, c)$.

   - $\therefore$ *the set of sequences* $L_p = \{[b], [c], [d], [e]\}$

2. *step 2 Construct $A_k s$, sets of sequences of length k.*

   - $A_1 = L_p = \{[b], [c], [d], [e]\}$, $A_2 = \{[e, b], [b, c], [c, d], [d, c], [c, e]\}$, $A_3 =$

$\{[e, b, c], [b, c, d], [d, c, e]\}$

3. *step 3 Construct L_t, the set of sequences of labels in the immediate neighborhood of terminal state.*

   - $L_t = \{[c], [d], [e]\}$

4. *step 4 The set of termination path TP initialised to L_t derived in step 3.*

   - $TP = L_t = \{[c], [d], [e]\}$

5. *step 5 Check for uniqueness of paths: $L_t = \{[c], [d], [e]\}$ and $L_p = \{[b], [c], [d], [e]\}$.*

   - $L_p \bigcap L_t \neq \emptyset = \{[c], [d], [e]\}$
   - $\therefore$ *update TP to $TP = \{[b, c], [c, d], [c, e]\}$*

6. *step 6 Height h of the protocol graph is 3, $\therefore$ Repeat for $(2 < n < 3)$*

   - *Current elements of TP are $[b, c], [c, d]$ and $[c, e]$, check if any is a subsequence of some sequence element of A_k (Recall $A_1 = \{[b], [c], [d], [e]\}$, $A_2 = \{[e, b], [b, c], [c, d], [d, c], [c, e]\}$, $A_3 = \{[e, b, c], [b, c, d], [d, c, e]\}$)*
     - *e.g. for the first iteration,$A_k = A_2 = \{[e, b], [b, c], [c, d], [d, c], [c, e]\}$ then for each of $[b, c], [c, d] or [c, e]$ check if any is a subsequence of some $[l_i] \in A_2$ and update with next transition where true, e.g. $[b, c] \subset [b, c] \in A_2$. Then $[b, c] \in TP$ is updated with the next transition, e in path p_3 to which it belongs, to become $[e, b, c]$ in the updated TP*

7. *step 7 If the protocol had multiple terminal states, then its set TP is the union of all $TP_i s$ where TP_i is set of termination paths for a particular terminal state derived as above.*

The algorithm for shortest termination paths, stp, is presented in Algorithm 1 next where the $reachability$ and $update$ procedures used within it are presented in Algorithms 2 and 3 respectively. The stp procedure declares global data structures, sets as defined in the preceding discussion, namely sets TP, A, L_p, L_t initialised to be empty. The variables N and K are also as defined, i.e N represents the set of indices for the total number of final states for a given protocol with multiple final states, and K the set of indices for indexing sets in the set $A = \cup_{k \in K} A_k$ as discussed previously.

Regarding the invoked procedures, $reachability$ and $update$, they have access to stp's global variables as initialised there, and their outputs are the updated global variables.

Algorithm 1 Shortest Termination paths algorithm

```
procedure $stp$ $(P, s, t)$

INPUT:          -Protocol $P$ = $(S, \longmapsto, L, T)$; $s \in S, t \in T$

OUTPUT:         -A set $TP$ of all shortest termination paths, sequences $\{l_i\}$ of.
                labels, i.e. $TP$ = $\{\{l_i\}|l_i \in L\}$ where $\{l_i\}s$ are unique ,
                sequences, i.e. for and indexed set $A = \cup_{k \in K} A_k$ , then
                $\forall A_k \in A$ $\nexists$ sequence $s_k \in A_k$ s.t $s_k = \{l_i\}$

GLOBAL DATA STRUCTURES:    Sets $TP$, $L_p$, $L_t$, $A$

INIT:           $N = \{n \mid 1 < n < |T|\}$
                $K = \{n \mid 1 < n < h\}$
                $h = height(P)$
                $TP = \emptyset$, $L_t = \emptyset$, $L_p = \emptyset$, $A = \emptyset$.

for all $(t \in T \wedge n \in N)$ do
  reachability($P, s, t$)

  set $TP_n = L_t$
  if $L_p \bigcap L_t = \emptyset$ then
    return $TP$
  else
    update($P, TP_n, A$)
  end if
end for
return $TP = \bigcup_{1 \leq n \leq |T|} TP_n$.
```

Algorithm 2 Reachability algorithm

```
procedure $reachability$ $(P, s, t)$

INPUT:              -Protocol $P$ = $(S, \longmapsto, L, T)$; $s \in S, t \in T$

OUTPUT:             -Updated set $L_p$ = $\{\{l_i\} \mid l_i \in L \wedge s_i \overset{l_i}{\longmapsto} s_{i+1}, s_{i+1} \neq t\}$,
                    -Updated set $L_t$ = $\{\{l_i\} \mid l_i \in L \wedge s_i \overset{l}{\longmapsto} t, t \in T\}$,
                    -Indexed set $A = \cup_{k \in K} A_k$

DATA STRUCTURES:    Access to $stp$'s global data structures $A, L_t, L_p$

for all $k \in K$ do
    traverse $P$ and construct $A_k s$ and derive $A$
    if $(k = h)$ then
        insert $l_i$ to $L_t$
    end if
end for

set $L_P = A_1$

return $L_p, L_t, A$
```

Algorithm 3 Termination paths update Algorithm

```
procedure $update$ ()

INPUT:              -Indexed set $A = \cup_{k \in K} A_k$
                    -Current set of shortest termination paths, $TP$

OUTPUT:             -Updated set of shortest termination paths, $TP$

DATA STRUCTURES:    Access to $stp$'s global data structures $A$, $TP$

repeat
  for all $\{l_i \dots l_n\} \in TP$ do
    for all $A_k \in A$ do
      for all $l_k \in A_k$ do
        if $\{l_i \dots l_n\} \subset l_k$ then
          $\{l_i \dots l_n\} \longrightarrow \{l_{i-1}, l_i \dots l_n\}$
        end if
      end for
    end for
    $A \longrightarrow A - A_K$
  end for
  $update()$
until $A = \emptyset$
return $TP$
```

5.4 Termination detection model

The discussion of termination detection algorithms in chapter 2 assumes and underlying process (object) model of computation and the algorithms discussed there use basic messages and rely primarily on low level constructs such as message counting and acknowledgements. Agents use high level agent communication languages, ACLs, and coordinate using structured interaction protocols that regulate their interactions, and agent messages have a context. Therefore an entity tasked with detecting global properties such as termination can use additional information about a protocol to carry out the task as discussed above in section 5.3.

Additionally a particular agent can engage in multiple interactions or conversations in pursuit of its goals. To this end we consider a conversational model for agent interactions.

5.4.1 A model for agent conversations

Consider a number of conversational scenarios;

1. *Scenario 1*: An agent engages in a conversation involving the execution of a single protocol in a single interaction with another party.
2. *Scenario 2*: An agent engages in multiple independent conversations involving execution of a single protocol.
3. *Scenario 3*: An agent engages in a single conversation that triggers additional conversations and the original conversation is not terminated until the sub-conversations are terminated, i.e. consider the recursive definition of a conversation.
4. *Scenario 4*: A generalisation of the point above, where an agent engages in multiple conversations that have sub-conversations.

First consider definition 10 below that defines a conversation. In this definition conversations can involve sub-conversations.

Definition 10. *A conversation is a tuple $\langle C_k, \langle C_{k,i}\rangle, P, F\rangle$ where $\langle C_{k,i}\rangle$ is a vector of its associated conversations (if it triggered any) and P is a protocol or sub-protocol (such as the set of termination paths TP) associated with this conversation. $F \in \{0, 1\}$, is a flag that is set if the root conversation C_k is completed, unset otherwise. Then for notational convenience we can write $C_k \mapsto F$ to refer to the boolean flag F associated with the root conversation C_k (or equivalently just write $C_k = 1$ or $C_k = 0$ or at a higher level, the notation $C_k \mapsto F$ can be represented by a predicate such as $computeF(C_k)$).*

Example 6. *If during a conversation C_1 between (i, j) further sub-conversations $C_{1,1}$ and $C_{1,2}$ are triggered, where $C_{1,1}$ and $C_{1,2}$ may as well be roots of further conversations, then represent this as $\langle C_1, \langle C_{1,1}, C_{1,2}\rangle, P, F\rangle$.*

Then consider definition 11 that defines a *conversation matrix* C-Matrix, a structure that can be used by an observer who oversees a conversation say in a termination detection of procedure.

Definition 11 (C-Matrix)**.** *Let M be a matrix of $m_{i,j}$ entries, $1 < i \leq n$ and $1 < j \leq n$. Let each $m_{i,j}$ entry be a tuple $\langle F, \langle C_k\rangle\rangle$, $F \in \{0, 1\}$ $1 < k \leq m$ for some m, where each C_k is an active conversation in definition 10 and $\langle C_k\rangle$ is a vector of root conversations. Write $M_{i,j} \mapsto F$ to reference F at $m_{i,j}$ (or equivalently just write $M_{i,j} = 1$ or $M_{i,j} = 0$, or at a higher level, the notation $M_{i,j} \mapsto F$ can be represented by a predicate such as $computeF(M_{i,j})$).*

Clearly regarding F, from definition 10 and 11, $M_{i,j} \mapsto F = \bigwedge_{1 \leq k \leq n} C_k \mapsto F$ (or equivalently $computeF(M_{i,j}) = \bigwedge_{1 \leq k \leq n} computeF(C_k)$) for the conversations in the vector $\langle C_k\rangle$, i.e. F set when all the conversations have been completed.

Example 7. *Figure 5.7 shows a representation of a c-matrix. $m_{i,j}$ in the matrix represents a registered interaction between i and j. A $\bigotimes$ at (i, j) represents flag F set, and existence a non-empty vector $\langle C_k \rangle$ of active conversations and a $\bigcirc$ at (i, j) represents flag F unset for terminated set of conversations in the vector $\langle C_k \rangle$. Conversations in the vector $\langle C_k \rangle$ are also flagged $\bigcirc$ and $\bigotimes$ as defined in definition 10 if the current state t in the protocol execution is a terminal state, i.e. $t \in T$.*

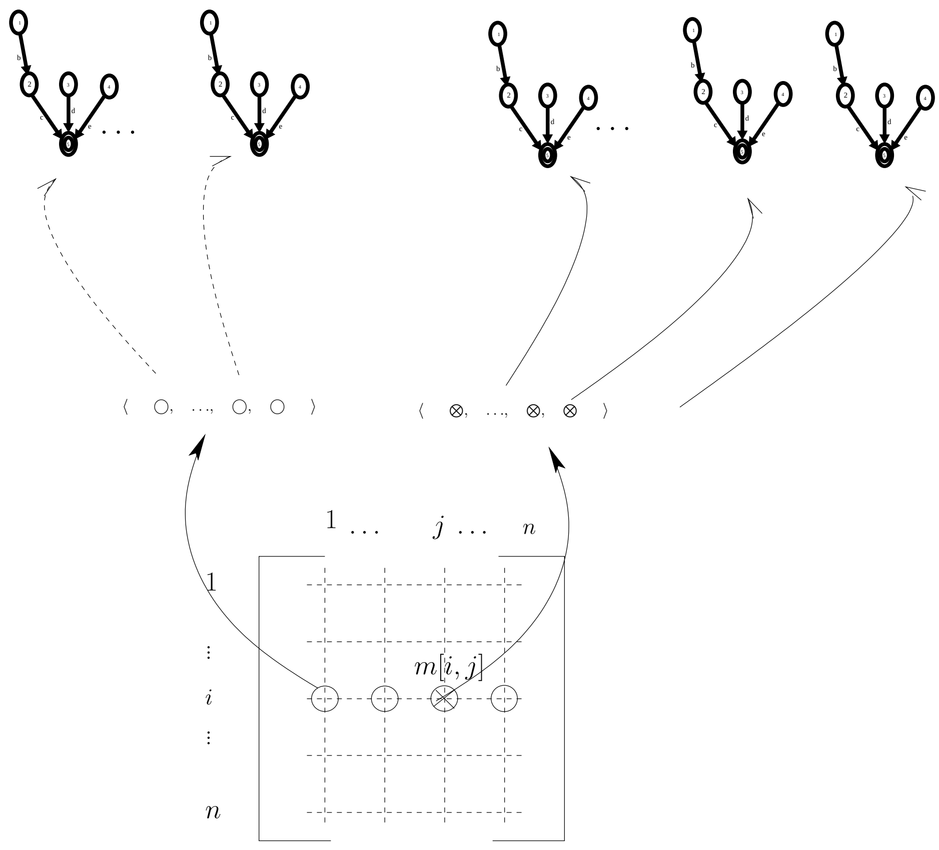


Figure 5.7: Showing a c-matrix structure and flat independent, each conversation in the vector C_k has no sub-conversations

Hierarchical conversations as diffusing computation trees As it is, example 7 on the use of a $c-matrix$ satisfies scenario 1 and scenario 2 above in page 83. To accommodate further scenarios, consider the use of the recursive definition of conversations in definition 10 to generalise the use of the conversation matrix ,$c-matrix$. We introduce a diffusing computation[7] tree of conversations, shown in Figure 5.8. Notice the recursive representation of the conversations.

Definition 12. *A diffusing computation tree for a conversation is a graph, a pair $G = (V, E)$ of sets satisfying $E \subseteq V \times V$, where the vertex set $V = \{C_i \mid \neg inactive(C_i)\}$, a set of active conversations, where the negation[8] of the predicate* inactive *tests existence of an active protocol execution associated with the conversation. E is the edge set, a binary relation, where an edge $e = (c_i, c_j)$ indicates that c_i triggered c_j, $E = \{(c_i, c_j) \mid (c_i, c_j) \in R \subset V \times V\}$ where V is the vertex set, the set of all possible conversations, $\therefore$ elements of E belong to $V \times V$.*

It then follows from the recursive definition of a conversation that a conversation graph G is made up of subgraphs $G^{'}(V_1, E_1), G^{''}(V_2, E_2) \ldots G^n(V_1, E_1)$ and $V = \cup_{i \in I} V_i$ and $E = \cup_{i \in I} E_i$, i.e. when a conversation C_i triggers a sub-conversation [9] C_j, a node $v = C_j$ and an edge $e = (C_i, C_j)$ are added to V and E respectively to grow G. The reverse happens when a sub-conversation terminates, i.e. a node and an edge are removed.

Figure 5.8 shows a diffusing computation tree representing a conversation, the root designates the conversation that spawned other conversations. The original conversation is terminated when the computation collapses to the root node and G reduces to a trivial graph of order[10] $|G| = 0$ and $||G|| = 0$, an *empty graph* $(\emptyset, \emptyset)$.

Next, consider extending [11] $E \subseteq V \times V$ relation to be *reflexive* and *transitive* and not

[7]A variant of Dijkstra diffusing computation.

[8]Defined using negation this way to simplify our subsequent discussions.

[9]Strictly we should write $C_{i,j}$ as defined, but we write C_j here for simplicity

[10]Order of a graph is the number of vertices, denoted $|G|$, equally number of edges is denoted $||G||$.

[11]We need to extend E to $E^{'}$ for use as a basis of a procedure used for concluding termination of conversations discussed later. These properties are useful because if we take the nodes to be computational

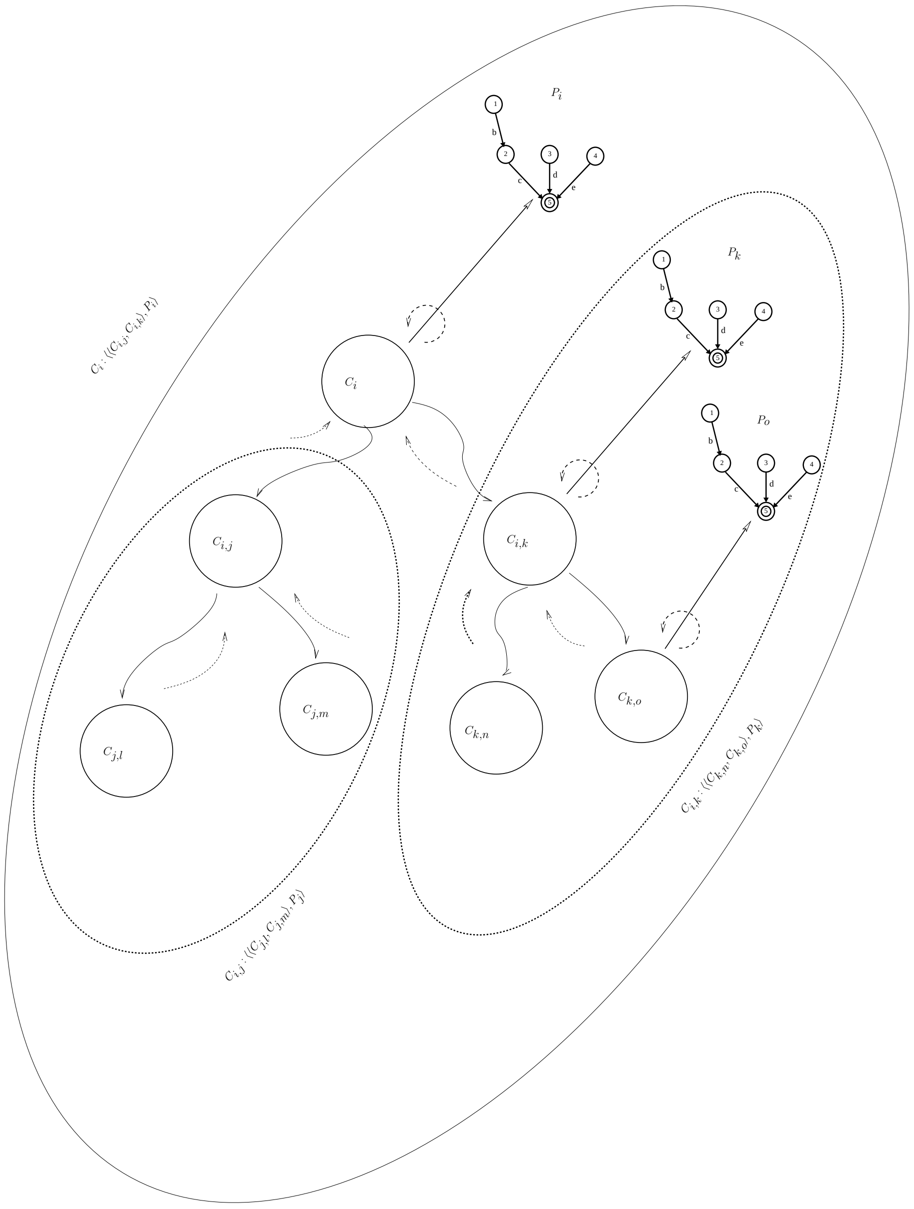

P_i
P_k
P_o
C_i
$C_{i,j}$
$C_{i,k}$
$C_{j,l}$
$C_{j,m}$
$C_{k,n}$
$C_{k,o}$
$C_i : \langle (C_{i,j}, C_{i,k}), P_i \rangle$
$C_{i,j} : \langle (C_{j,l}, C_{j,m}), P_j \rangle$
$C_{i,k} : \langle (C_{k,n}, C_{k,o}), P_k \rangle$
1
2
3
4
b
c
d
e

Figure 5.8: Showing a diffusing computation graph representing a conversation. Using labelling that reflects parent nodes. The root node C_i designates the conversation that spawned other conversations. Each conversation if active maintains protocol execution or set of termination paths, TP. A terminated conversation collapses to the root node C_i

symmetric to produce a relation $E^{'}$. For example consider Figure 5.9.

1. *Reflexivity* , e.g. if $C_i \in V$, $C_j \in V$, $C_k \in V$ etc. then $(C_i, C_i) \in E^{'}$, $(C_j, C_j) \in E^{'}$, $(C_k, C_k) \in E^{'}$, i.e. $R_{reflexive} = \{(C_i, C_j) \mid (C_i, C_j) \in V \times V \wedge \forall i \ \forall j \ i = j\}$. Permitting reflexivity in the model allows a sub-conversation to report its local termination[12] and self removal from the set of active conversations as the diffusing computation tree collapses.

2. *Transitivity* , i.e. if $(C_i, C_j) \in E^{'} \wedge (C_j, C_k) \in E^{'} \Rightarrow (C_i, C_k) \in E^{'}$, or more generally , $R_{transitive} \subseteq V \times V$ = $\{(C_i, C_k) \mid \exists \ C_j \in V \ \ s.t \ C_i \overset{*}{\longmapsto} C_j \ \wedge \ C_j \overset{*}{\longmapsto} C_k \ \}$, where notation $C_j \overset{*}{\longmapsto} C_k$ indicates an existence of path from C_j to C_k. Allowing transitivity provides and additional safety check before removing nodes, e.g. for some root conversation C_i, test if $R_{transitive} = \{(C_i, C_k) \in E^{'} \mid (C_i, C_k) \in E^{'} \ \forall C_k \in V\} = \emptyset$ before removing any C_j in the path to C_k s.t $C_j, C_k \in E^{'}$ and $C_i, C_j \in E^{'}$ from V and C_i from V, i.e. test if there are no descendant conversations transitively related to C_i.

3. Not *symmetric* because clearly if a conversation C_i triggers conversation a C_j i.e. $(C_i, C_j) \in E^{'}$, it is not the case that C_j triggers conversation C_i, i.e. $(C_j, C_i) \notin E^{'}$.

That is, we define *reflexive* and *transitive* closures of G to be the graph $G^{'} = \left(V, E^{'}\right)$, where $E^{'} = E \ \cup \ R_{reflexive} \ \cup \ R_{transitive}$, the idea is to use these extended properties in a procedure that collapses the conversation tree safely when conversations (nodes) complete and are removed from the tree until eventually G reduces to an *empty graph* $(\emptyset, \emptyset)$.

as is the case with diffusing computations, then the nodes can have operations to remove themselves from the computation tree. A data structure to represent the graph $G(V, E)$, can be maintained e.g. in a form of an adjacency matrix and node can inspect and manipulate this data structure.

[12] Recall that these nodes are computational

Example 8. *Consider Figure 5.10 that shows a relation*[13] $E^{'} = R \subset V \times V$ *, an extended edge set* $E^{'}$ *of conversation* $G^{'} = (V, E^{'})$ *representing set of pairs of conversations such that one conversation is triggered by another for the example diffusing computation tree given in Figure 5.8. In this example assume the diffusing computation tree presented in Figure 5.8 is given concrete labels such that the conversation* $G = (V, E)$ *is rooted at* C_1 *with* C_1 *triggering* C_2 *and* C_5*;* C_2 *triggering* C_3 *and* C_5*;* C_5 *triggering* C_6 *and* C_7*. The original conversation,* C_1 *completes when eventually* $E^{'} = \emptyset$*. Nodes designated by* $\bigotimes$ *show reflexivity of the Relation* $E^{'}$*, i.e. we allow that if a conversation is triggered by another, then we assume that the triggered conversation has also self triggered trivially , i.e.* $\{(C_i, C_j) \in E^{'} \mid C_i = C_j\}$*, this can be used in a heuristic to ensure that when a conversation is completed the edge is removed from* $R_{reflexive} \subset E^{'}$ *and the node from* V*. Alternatively for inspecting* $R_{reflexive} \subset E^{'}$ *for active conversations, checking set membership, without traversing the graph in an implementation where the computational nodes update this set themselves, for example. The figure also shows relation* $R_{transitive}$ *used in Algorithm 4.*

[13]Grid used to show the cartesian product $V \times V$. Number labels on the axes for set elements represent identifiers i for conversation nodes.

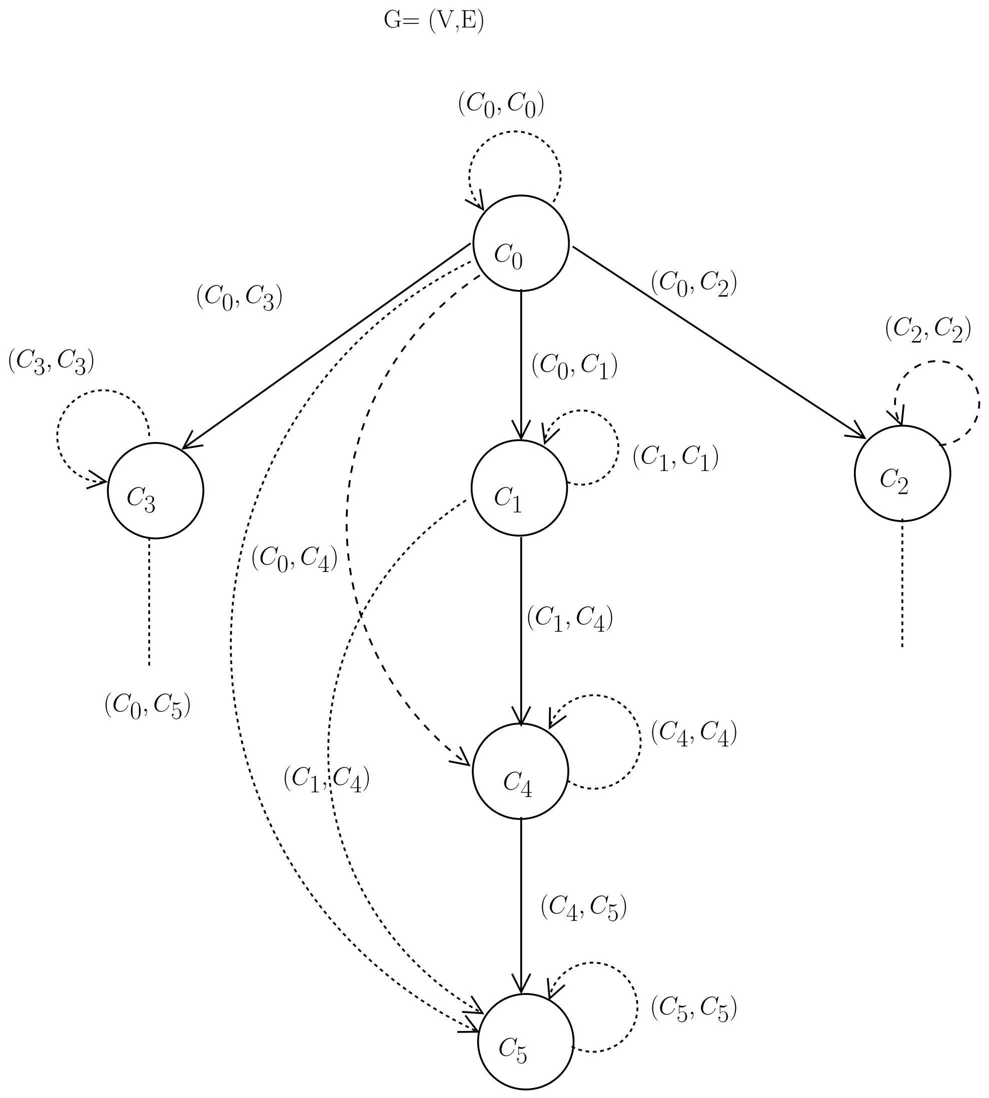


Figure 5.9: Example conversation $G = (V, E)$ showing transitivity and reflexivity of the edge set E after extending edge set E with $R_{reflexive}$ and $R_t ransitive$ to derive a relation $E' = E \cup R_{reflexive} \cup R_{transitive}$

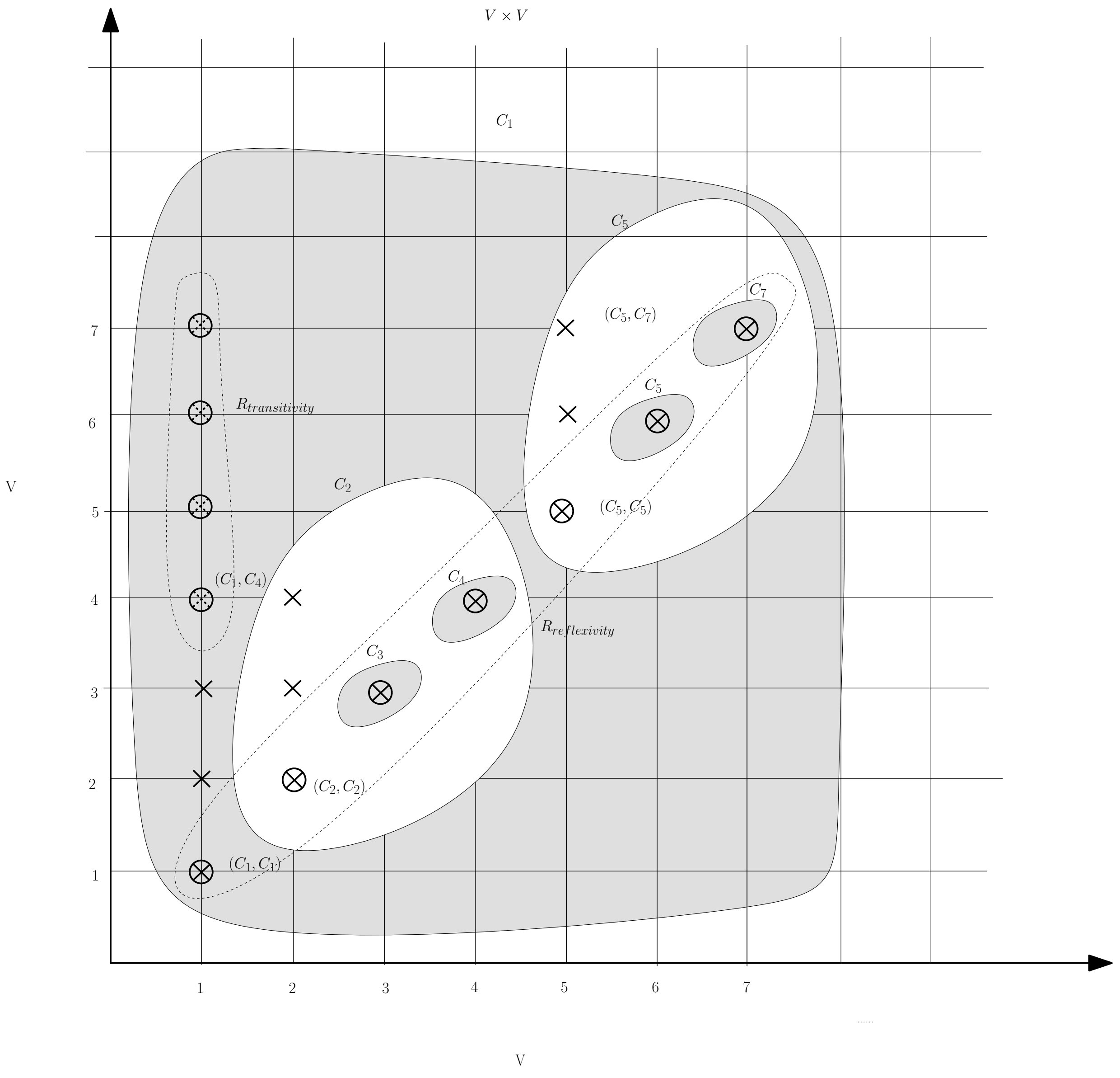


Figure 5.10: Showing $E^{'} \subset V \times V = E \cup R_{reflexive} \cup R_{transitive}$ and some sample elements for a conversation $G = (V, E)$. In this example assume the diffusing computation given in Figure 5.8 is given concrete labels such that the conversation $G = (V, E)$ is rooted at C_1 with C_1 triggering C_2 and C_5; C_2 triggering C_3 and C_5; C_5 triggering C_6 and C_7. The original conversation, C_1 completes when eventually $E^{'} = \emptyset$

Procedure for local termination So the following procedure local termination [14], follows.

Consider a predicate *inactive* raised in Definition 12 page 86 defined over conversations. It evaluates false for an active conversation that has an *active* associated protocol execution and its extended edge set $E^{'}$ is not empty, i.e. consider definition 13. Consider also predicate *inactiveProtocol* defined over protocols that tests whether a given protocol execution has reached a terminal state, i.e. consider Algorithm 5

Definition 13. *For a conversation* $C_i = G(V, E^{'})$*,*
$\neg inactive\,(C_i) \quad \Leftrightarrow \quad active(P_i) \quad \wedge \quad E^{'} \neq \emptyset$*, where for an associated Protocol or sub-protocol ,* P_i*,* $active(P_i) \quad \Leftrightarrow \quad currentstate(P_i) \notin T$

Given this discussion, a sub-conversation C_j locally terminates when predicate $inactive(C_j)$ defined on conversations evaluates true and hence when its associated protocol execution completes and $inactiveProtocol$ evaluates true. A protocol execution completes when one of the terminal states is reached. The overall conversation is completed when the root node is eventually removed from V and last edge from E and G reduces to a *empty graph* $(\emptyset, \emptyset)$. When removing nodes, test to check existence of adjacent nodes and transitively related nodes. appendix:termination.detection.for.protocol

Consider Algorithm 4 below suggested by the discussion so far.The algorithm traverses the tree[15] in breadth-first and at each node evaluating whether there are any descendants conversation nodes i.e. evaluating if $R_{transitive} = \emptyset$, testing for if the protocol is active using $activeProtocol$ predicate and removing that node with a procedure $remove$ if the above is true. When G eventually becomes empty, the associated flag F (defined in Definition 10 in page 84) of a conversation can be set.

[14]Local because we refer to a conversation, not a set of all conversations an agent is engaged in.
[15]For further illustration of this algorithm, consider an example trace for execution of this algorithm given in Appendix A, Figure A.1 in page 261.

Also consider Algorithm 5 that specifies predicate $inactiveProtocol$ defined over protocols.

Algorithm 4 Diffusing conversations algorithm

```
procedure $inactive$ ( $G$)

INPUT:        -Conversation $G$ = $(V, E)$ rooted at $C_i$
OUTPUT:       -Boolean $T$ or $F$

DATA STRUCTURES:        Graph $G$ = $(V, E)$, i.e. sets $V$ and $E$
**INIT**: Initialise sets $V$ and $E$ to vertices and edges of $G$. Initialise sets $R_{reflexive}$,
$R_{transitive}$ and construct set $E^{'} = E \ \cup \ R_{reflexive} \ \cup \ R_{transitive}$

**repeat**
  **for all** $(C_j \in V \ | \ (C_i, C_j) \in E \ \cup \ R_{reflexive})$ **do**
    **if** $(R = \{(C_j, C_k) \ | \ (\forall k \ \exists C_k \in V) \wedge (C_i, C_k) \in R_{transitive}\} = \emptyset)$ **then**
      **if** $\neg inactiveProtocol(C_j \mapsto P_j)$ **then**
        remove $(G, C_j)$
      **end if**
    **else**
      active($G_j$)
    **end if**
  **end for**
**until** $(G = \emptyset, \emptyset)$

procedure $remove$ ( $G$, $C_j$)

INPUT:        -Conversation $G$ = $(V, E)$ rooted at $C_i$; vertex $C_j$ s.t $(C_i, C_j) \in E^{'}$
OUTPUT:       -Pruned $G$ = $(V, E)$; Updated $E^{'}$ i.e. $E$, $R_{reflexive}$ and $R_{transitive}$

Compute $V = V - C_j$
Compute $E^{'} = E^{'} - \{(C_k, C_j) \ | \ \forall k \ (C_k, C_j) \in E^{'}\}$
**return** $G$ = $(V, E)$
```

Algorithm 5 Algorithm for predicate $activeProtocol$

```
procedure $inactiveProtocol$ ( $P$)

Let $P$ be a protocol and let $TP$ be the set of *shortest unique termination paths* with
*observables* $O \subset L$, $L$ set of all labels.

INPUT:          A protocol trace, label $l \in L$ or sequence $[l]$
OUTPUT:         -Boolean $T$ or $F$

DATA STRUCTURES: $\forall\ p = (s_1, \dots, s_n) \in TP$ , there is a state $s_i$ called current
execution state of $p$.
INIT: $\forall p = (s_1, \dots, s_n) \in TP$ initialise its current execution state to $s_1$.
repeat
  Let $l$ be a message sent by an agent.
  if $l \notin O$ then
    for all $p = (s_1, \dots, s_n) \in TP$ do
      Set $p's$ current execution state to $s_1$.
    end for
  else
    for all  $p = (s_1, \dots, s_n) \in TP$ with current execution state $s_i$ do
      if $s_i \overset{l}{\longmapsto} s_{i+1}$ then
        $p$'s current execution state becomes $s_{i+1}$
        if $s_{i+1} = s_n$ then
          set terminated to true. {-Specifically for $p$}
        end if
      end if
      if  $s_i \overset{l}{\not\longmapsto} s_{i+1}$ then
        set $p$'s current execution state to $s_1$
      end if
    end for
  end if
until TERMINATED
```

Discussion of the algorithm's complexity If the algorithm has to visit all nodes and all edges when testing and removing nodes from the computation tree, the time and space complexity will be in the order $\mathcal{O}\left(|V| + |E|\right)$. Or equivalently , if the *branching factor* of the tree b and tree depth d were to be known at any time, then theoretically, like the *breadth-first-search* , the time and space complexities can be expressed as $\mathcal{O}\left(b^d\right)$ [55, 134].

In practical terms, for the protocols we are considering in multiagent applications, it is worth pointing out that the protocol graphs used in agent interactions as for example given in the FIPA protocol suite [1], are not very big. For example, consider the ContractNet protocol [215] given as an example illustration of an FSM representation of a protocol in Figure 5.1 in page 70 or consider an equivalent FIPA Contact-Net representation from the FIPA protocol suite in AUML notation shown in Figure 4.4 in page 60. Both examples show typical graph sizes of the protocols that are considered in multiagent applications.

In addition, if as it is, the implementation can be such that the nodes can individually update the set data structure for $E^{'}$, then an algorithm that then inspects $E^{'}$ to answer questions about the state of the diffusing computation tree will yield better complexity than $\mathcal{O}\left(|V| + |E|\right)$ as it will just test set membership in $E^{'}$. We give a further detailed discussion of algorithms discussed in this chapter in Appendix A in section A.4, page 268.

5.5 Distributed protocol for termination detection

Consider a small [16] set of controllers $\mathcal{C} = \{\mathcal{C}_n \mid n \in \mathbb{N}\}$ modelled as a fully connected network and consider a given agent whose protocol executions are observed by these controllers. Recall from definition 11 that entries $\mathbf{m}_{i,j}$ are tuples $\langle F, \langle C_k \rangle \rangle$, with $F \in \{0, 1\}$ and with $\mathbf{m}_{i,j} \mapsto F$ written to reference F. F indicates whether conversation C_k is active or completed.

So one approach is for each controller $\mathcal{C}_n$, to compute, based on the $c - matrix$ it maintains, for a given ith *row*, (agent), $\bigwedge_{\forall j} \mathbf{m}_{i,j} \mapsto F$, i.e. check if *all* flags F are set at each of the $\mathbf{m}_{i,j}$ entries in the ith row. Also recall that each $\mathbf{m}_{i,j} \mapsto F$ is in turn computed from $\bigwedge_{1 \leq k \leq n} C_k \mapsto F$ where each C_k here refers to an entry in the vector $\langle C_k \rangle$ of active conversations at $\mathbf{m}_{i,j}$.

So consider some logical global $c - matrix$ $\mathbf{M}$ and definition 14 below.

Definition 14. *Given an agent i and a set $\mathcal{C} = \{\mathcal{C}_n \mid n \in \mathbb{N}\}$ of controllers , let the global conversation matrix $\boldsymbol{M}$ be partitioned across Controllers $\{\mathcal{C}_n\}$, i.e. such that $\bigcap_{n \in \mathbb{N}} \mathcal{C}_n \left(\boldsymbol{M}_{i,j}\right) = \emptyset$, where $\mathcal{C}_n \left(\boldsymbol{M}_{i,j}\right)$ indicates a set of entries from $\boldsymbol{M}_{i,j}$ assigned to controller $\mathcal{C}_n$. Write $\bigwedge_{\forall j} \boldsymbol{M}_{i,j} \mapsto F$ as $\mathcal{C}_n^i \mapsto F$ for $\bigwedge_{\forall j} \boldsymbol{M}_{i,j} \mapsto F$ computed by a controller $\mathcal{C}_n$. Then $\bigwedge_{n \in \mathbb{N}} \mathcal{C}_n^i \mapsto F$ for all controllers in $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$ indicates global termination for agent[17] i if true*

Example 9. *Consider Figure 5.11 showing a global matrix $\boldsymbol{M}$ partitioned across controllers in $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$. If all the entries $\boldsymbol{M}_{i,j} \mapsto F$ evaluate to true on row i for all columns j, then all conversations associated with i have completed,i.e. because all the $M_{i,j}$s on row i are partitioned across controllers $\{\mathcal{C}_n\}$ termination is global if $\bigwedge_{n \in \mathbb{N}} \mathcal{C}_n^i \mapsto F$ evaluates true.*

[16] Observing that the number of connections is quadratic in number of nodes, $\frac{n^2-n}{2}$, order $\mathcal{O}\left(n^2\right)$.

[17] The significance of "agent" here is that if we seek a global state where all agent interactions have terminated, e.g. if we seek quiescence, each controller observing conversations for this agent have to report termination.

Now, there are various ways to implement a global $c - matrix$ $\mathbf{M}$.

1. Partition $\mathbf{M}$ logically by allowing each controller to manage a separate copy of a $c - matrix$ and using global identifiers and ensuring that conversations for a particular pair of agents instantiations (i, j) are registered on a particular controller, i.e. $\bigcap_{n \in \mathbb{N}} C_n \left(\mathbf{M}_{i,j}\right) = \emptyset$ in line with definition 14, e.g. Figure 5.11, i.e. there is no overlap.

2. Allocate $\mathbf{M}$ logically across controllers allowing each controller to manage a separate copy of a $c - matrix$ and using global identifiers but allowing that *some*[18]conversations for a particular pair of agents instantiations (i, j) can registered on different controllers, i.e. allowing $\bigcap_{n \in \mathbb{N}} C_n \left(\mathbf{M}_{i,j}\right) \neq \emptyset$, e.g. Figure 5.12

Furthermore, Appendix A, page 263 discusses additional abstract configurations that can be considered for implementing a global $c - matrix$ $\mathbf{M}$.

Figures 5.11 and 5.12 next illustrate configurations one and two respectively as discussed above.

[18]But not the same conversation, i.e. some from the vector C_k at $m_{i,j}$.

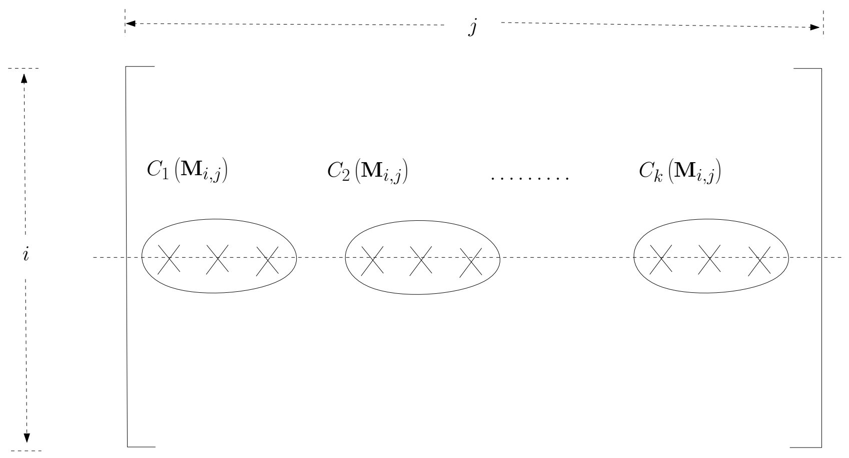


Figure 5.11: Showing example global matrix partitioned across controllers enforcing the condition $\bigcap_{n\in\mathbb{N}} C_n\left(\boldsymbol{M}_{i,j}\right) = \emptyset$

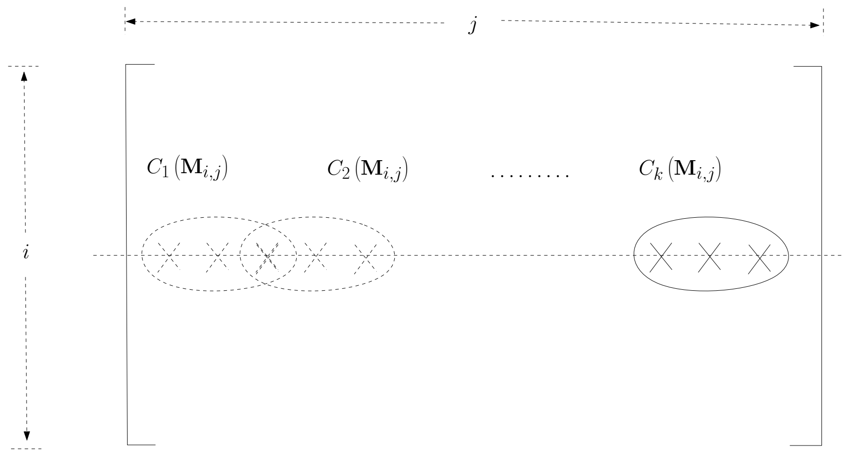


Figure 5.12: Showing example global matrix allocated across controllers allowing overlaps, i.e. $\bigcap_{n\in\mathbb{N}} C_n\left(\boldsymbol{M}_{i,j}\right) \neq \emptyset$

Assuming setups 1 and 2 above, and a set of controllers $\mathcal{C} = \{\mathcal{C}_n \mid n \in \mathbb{N}\}$ modelled as a fully connected network, the controllers can synchronise using message passing to compute $\bigwedge_{n\in\mathbb{N}} \mathcal{C}_n^i \mapsto F$ for a given agent i to ascertain termination or compute $\bigwedge_{n\in\mathbb{N}} \mathcal{C}_n^{\forall i} \mapsto F$ if quiescence [19] of the system is required.

A designated controller, say $\mathcal{C}_0$, may propagate $query$ messages to other controllers and aggregate to compute $\bigwedge_{n\in\mathbb{N}} \mathcal{C}_n^i \mapsto F$ from the $reply$ messages carrying local values of $\mathcal{C}_n^i \mapsto F$, this at a cost of $2 \times (n-1)$ messages, order $\mathcal{O}(n)$ [20].

Local computations by controllers in $\mathcal{C} = \{\mathcal{C}_n\}$ to derive $\mathcal{C}_n^i \mapsto F$ will involve tests of the predicate $inactive$ (defined in Algorithm 4) over all active conversations in the vector $\langle C_k \rangle$ at $\mathbf{m}_{i,j}$ entries in a local $c - matrix$ **m** at each controller $\mathcal{C}_n$.

Consider Algorithm 7 below that implements the scheme above.

[19] No activity in the system, state of being quite will all protocol execution complete.

[20] $\mathcal{O}(n)$ because one controller sends and collates results to declare termination, clearly in the worst case, theoretically if every controller was sending to other we will have $\mathcal{O}(n^2)$.

Algorithm 6 Global termination on controllers $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$, reported by $\mathcal{C}_0$ for a given agent i. C_0 sends $query$ messages and concludes termination by aggregating results including its own

```
procedure $globaltermination$ ( $i$)

INPUT:              -agent identifier $i$;
OUTPUT:             -Boolean $F = \{0, 1\}$; $c - matrix$ $\mathbf{m}$

DATA STRUCTURES:              $\mathcal{C}$ = $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$

for all $\mathcal{C}_n \in \mathcal{C}$ do
  if $\mathcal{C}_n = \mathcal{C}_0$ then
    repeat
      $(\mathcal{C}_0^i \mapsto F) \longleftarrow localtermination(\mathbf{m}, i)$
      $(\mathcal{C}_n^i \mapsto F) \longleftarrow query\,(C_n, i)$
    until $\bigwedge_{n \in \mathbb{N}} \mathcal{C}_n^i \mapsto F$
  else
    $(\mathcal{C}_n^i \mapsto F) \longleftarrow localtermination(\mathbf{m}, i)$
  end if
end for

return $(\bigwedge_{n \in \mathbb{N}} \mathcal{C}_n^i \mapsto F)$
```

Algorithm 7 Local termination on controller $\mathcal{C}_n$ for agent i, handles $query\,(\mathcal{C}_n, i)$ messages from $\mathcal{C}_0$, returns $\mathcal{C}_n^i \mapsto F$ from this controller

```
procedure $localtermination$ (m, $i$)

INPUT:            -agent identifier $i$; $c - matrix$ m
OUTPUT:           -Boolean $F = \{0, 1\}$

DATA STRUCTURES:           local $c - matrix$ m

repeat
  for all $j$ do
    for all $\mathbf{m}_{i,j} \mapsto C_k$ do
      $F \longleftarrow \bigwedge_{1 \leq k \leq n} inactive\,(\mathbf{m}_{i,j} \mapsto C_k)$
    end for
  end for
until $F$

return ($F$)
```

Procedure for global termination So given the discussion so far, we can summarise the procedure for global termination. In addition consider the following observations and practical considerations regarding the procedure.

1. Define a $wave$ as the basic sequence of sending of *query* messages followed by the reception of associated *reply* messages in line with the discussion in section 2.3, page 350.

2. Observe the symmetric nature of conversations, i.e. for a pair $\mathcal{A}_i$, $\mathcal{A}_j \in \mathcal{A}$ then $(\mathcal{A}_i, \mathcal{A}_j) \in A \times A \iff (\mathcal{A}_j, \mathcal{A}_i) \in A \times A$, so it is sufficient to maintain one entry in the $c - matrix$ for the pair $(\mathcal{A}_i, \mathcal{A}_j)$.

3. All agents $\mathcal{A}_i \in \mathcal{A}$ send partial protocols (*termination paths*) or full protocols as payload for *registration* messages to $\mathcal{C}_n \in \mathcal{C}$ and *reply* to *query* control message in wave with payload as protocol traces or labels $l \in \mathcal{L}$.

4. Designate one controller $\mathcal{C}_0 \in \mathcal{C}$ to conclude termination using predicate *local-termination* tested on the global $c - matrix$ $\mathbf{M}$ cache. $\mathcal{C}_0$ sends $sync$ control signals to all controllers $\mathcal{C}_n \in \mathcal{C}, n \neq 0$ to trigger cache updates. For failover, this role can be transferred to any $\mathcal{C}_n$ as all controllers have the same instructions apart from one being identified as $\mathcal{C}_0$ for this purpose.

5. Using a global $c - matrix$ $\mathbf{M}$ cache instead of the full use of bidirectional communication to coordinate,reduces control message traffic to other controllers. This can be reduced further if cache updates are made periodic on independent controllers without the need for $sync$ signals. Regarding potential failure of the tuple spaces, standard replication and recovery mechanisms for shared memory can be used. for example [182] discusses distributed shared memory issues.

6. We distinguish between two usage scenarios; global termination of an agent $\mathcal{A}_i$'s

interactions or *quiescence* of the system, i.e. considering global termination of all agents $\mathcal{A}_i \in \mathcal{A}$.

7. While the procedure is configured to ascertain termination, it can be easily adapted to provide continuous observation of protocol executions and be used as a basis for some crash recovery mechanism for agents.

8. Furthermore as observed in Chapter section 2.1 page 16, the semantics of garbage collection problem are fully contained in the semantics of the termination detection problem, hence we can a derive garbage collection scheme from the termination detection scheme. Therefore we could use this procedure as a basis for marking terminated agents for garbage collection [21].

Figure 5.13 summarises the discussion so far.

Note in that figure, that in the block $\mathcal{S}2$ regarding registration of active conversations, this refers to conversations a particular agent embarks on and hence it is aware of its conversation partners , its role and the protocol it participates in. At the start, if $(A_i, A_j) \in \mathcal{A} \times \mathcal{A}$ are a pair of agents participating in a protocol P, where $\mathcal{A} = \{A_n\}$ is the set of all agents, then initiator of the conversation [22] can register a conversation with a controller C_i from the set of controllers $\mathcal{C} = \{\mathcal{C}_n\}$. For this registration, consider a predicate $register(i, j, P, \mathcal{C}_n)$ that can be implemented to send a control message to some controller $\mathcal{C}_n \in \mathcal{C}$. The controller $\mathcal{C}_n$ handles this message by inserting an entry [23]into the $c - matrix$ $\mathbf{M}$ at (i, j).

Furthermore as remarked above (second point), because conversations are symmetric, one entry need be maintained in the $c - matrix$ $\mathbf{M}$, i.e at (i, j) and not at (j, j) too in $\mathbf{M}$.

[21] This maybe an interesting research work to follow work discussed here.

[22] Recall that a conversation encapsulates a protocol execution

[23] A new conversation as given in Definition 10, a tuble $\langle C_1, \langle\rangle, P, F\rangle$ with F = 0, empty sub conversations initially, i.e $\langle\rangle$

Also note in Figure 5.13, that in the statement block $\mathcal{S}4$, regarding $sync$ operation, this is discussed above (fourth point). Consider a predicate $sync(\mathcal{C}_n)$ over all controllers in $\mathcal{C}_n \in \mathcal{C}$. The implementation of this involves sending a control message to all controllers by a designated controller $\mathcal{C}_0$, querying for the flag F computed by all other controllers $\mathcal{C}_n \in \mathcal{C}$ for a given agent A_i. This is shown in the $query$ predicate in within Algorithm 7 presented in page 101. Similarly see Algorithm 8 in Appendix A in page 267 for the other configurations for distributing the $c - matrix$ $\mathbf{M}$.

5.5.1 Evaluation of the distributed termination detection protocol

As discussed in section 2.4, page 34, a set of metrics can be considered regarding evaluation of distributed termination schemes, namely *detection latency*, *message complexity* and *message-size complexity*. Briefly;

1. *Detection latency*: Quantifying the period between when the underlying computation completes and when the termination algorithm actually announces termination.

2. *Message complexity*, also communication complexity, refers to the number of control messages exchanged by the termination detection algorithm in order to detect termination. In general, it is indicated by [153], that this is less significant in a distributed algorithm unless the communication complexity causes sufficient congestion to slow down processing. Clearly also in practice, say in an agent platform, there will be an upper limit on the number of agents a platform can host and scalability issues naturally arise, but theoretically in a broadcast scheme complexity is of the order $\mathcal{O}(n)$ as discussed below.

3. *Message-size complexity* refers to the size of the control data as payload on the message by the termination detection algorithm, that is, how big the messages are.

Regarding *detection latency*, we investigate this extensively in the experimental setup discussed in the next chapters for the prototype implementation.

Regarding *message complexity*, we do not assume a particular topology for the agents in the multiagent system in that associations between agents are dynamic. One possibility is to assume a *clean graph*, (see definition 16).

In addition we can discount topologies that require agents to pass on control messages to other agents en route to controller [24].

Therefore we choose to use a broadcasting scheme (from controllers to agents), to implement the *waves*. Also in the broadcasting scheme, the total number of control messages will be influenced by the periodicity parameter of the control waves, but the message complexity for the wave is $\mathcal{O}(n)$.

Definition 15 (Broadcast)**.** *A broadcast operation is initiated by a single node , the source, and the receivers are all other nodes in the system.*

Remark 1 (Lower bound)**.** *Message complexity of the broadcast is at least* $n-1$*, order* $\mathcal{O}(n)$.

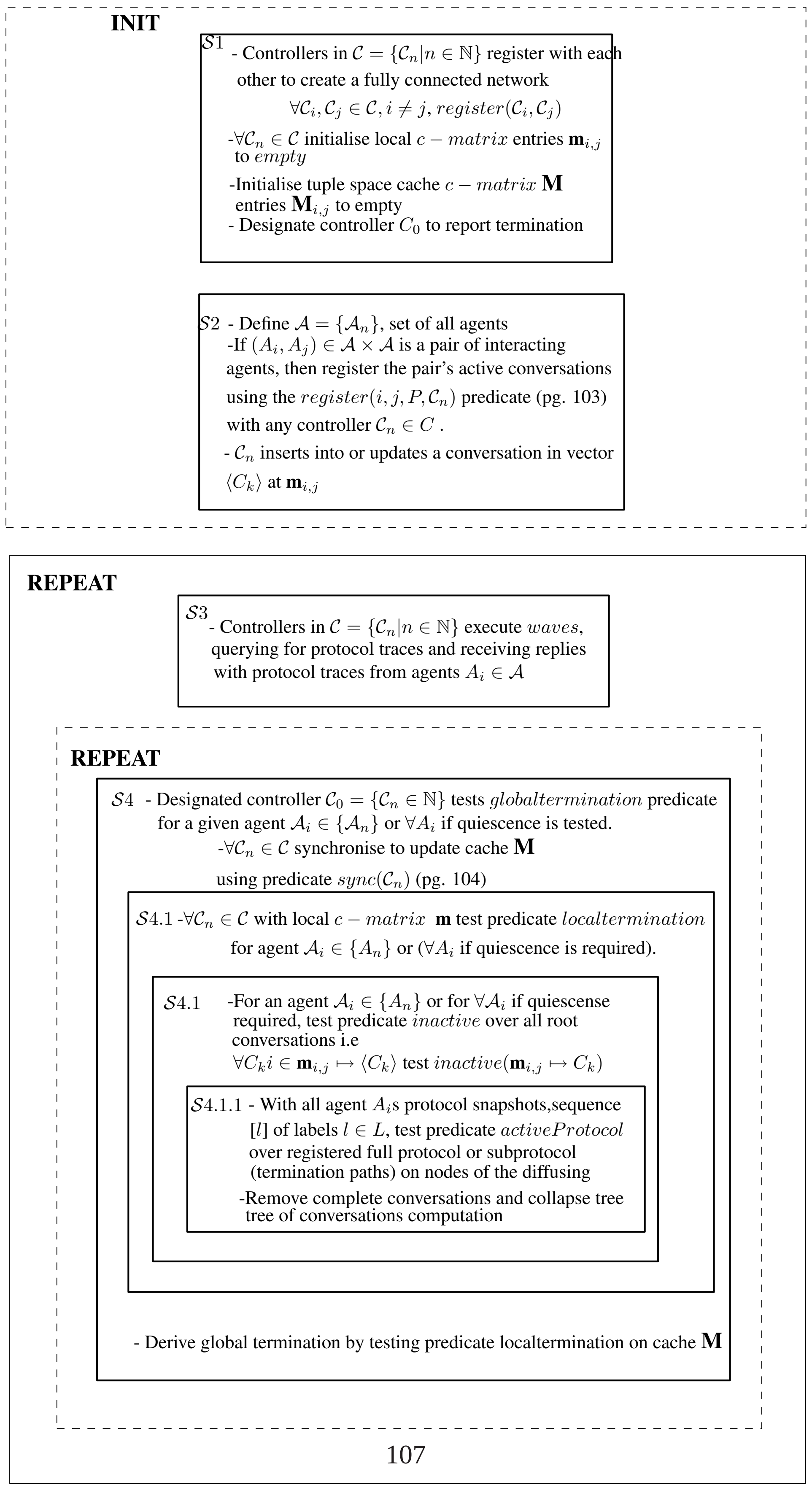
Proof. The proof is trivial, every node must receive the message. ☐

Definition 16 (Clean)**.** *A graph , system or network is clean if nodes do not know the topology of the graph.*

Regarding the *message-size complexity*, $query$ messages are light and as remarked above $reply$ messages carry as payload[25] protocol traces or labels and therefore are equally light. Registration messages a heavier but sent only once for a given conversation, carrying partial protocol (*termination paths*) or full protocol graphs.

[24]There are notions of malicious agents in multiagent system that can manipulate messages. Consider a scenario of an auction.

[25]Serialisation of structures can be used to implement this.



INIT

$\mathcal{S}1$ - Controllers in $\mathcal{C} = \{\mathcal{C}_n | n \in \mathbb{N}\}$ register with each other to create a fully connected network

$\forall \mathcal{C}_i, \mathcal{C}_j \in \mathcal{C}, i \neq j, register(\mathcal{C}_i, \mathcal{C}_j)$

-$\forall \mathcal{C}_n \in \mathcal{C}$ initialise local $c-matrix$ entries $\mathbf{m}_{i,j}$ to $empty$

-Initialise tuple space cache $c-matrix$ $\mathbf{M}$ entries $\mathbf{M}_{i,j}$ to empty

- Designate controller C_0 to report termination

$\mathcal{S}2$ - Define $\mathcal{A} = \{\mathcal{A}_n\}$, set of all agents

-If $(A_i, A_j) \in \mathcal{A} \times \mathcal{A}$ is a pair of interacting agents, then register the pair's active conversations using the $register(i, j, P, \mathcal{C}_n)$ predicate (pg. 103) with any controller $\mathcal{C}_n \in C$.

- $\mathcal{C}_n$ inserts into or updates a conversation in vector $\langle C_k \rangle$ at $\mathbf{m}_{i,j}$

REPEAT

$\mathcal{S}3$ - Controllers in $\mathcal{C} = \{\mathcal{C}_n | n \in \mathbb{N}\}$ execute $waves$, querying for protocol traces and receiving replies with protocol traces from agents $A_i \in \mathcal{A}$

REPEAT

$\mathcal{S}4$ - Designated controller $\mathcal{C}_0 = \{\mathcal{C}_n \in \mathbb{N}\}$ tests $globaltermination$ predicate for a given agent $\mathcal{A}_i \in \{\mathcal{A}_n\}$ or $\forall A_i$ if quiescence is tested.

-$\forall \mathcal{C}_n \in \mathcal{C}$ synchronise to update cache $\mathbf{M}$ using predicate $sync(\mathcal{C}_n)$ (pg. 104)

$\mathcal{S}4.1$ -$\forall \mathcal{C}_n \in \mathcal{C}$ with local $c-matrix$ **m** test predicate $localtermination$ for agent $\mathcal{A}_i \in \{A_n\}$ or ($\forall A_i$ if quiescence is required).

$\mathcal{S}4.1$ -For an agent $\mathcal{A}_i \in \{A_n\}$ or for $\forall \mathcal{A}_i$ if quiescense required, test predicate $inactive$ over all root conversations i.e

$\forall C_k i \in \mathbf{m}_{i,j} \mapsto \langle C_k \rangle$ test $inactive(\mathbf{m}_{i,j} \mapsto C_k)$

$\mathcal{S}4.1.1$ - With all agent A_is protocol snapshots,sequence $[l]$ of labels $l \in L$, test predicate $activeProtocol$ over registered full protocol or subprotocol (termination paths) on nodes of the diffusing

-Remove complete conversations and collapse tree tree of conversations computation

- Derive global termination by testing predicate localtermination on cache $\mathbf{M}$

107

Figure 5.13: Showing the distributed termination detection protocol

5.5.2 Proposed architecture for termination detection

Figure 5.14 illustrates how a termination detection mechanism may fit in with a generic multiagent management architecture, not only to report termination, but also to provide continual observation of interactions and visualisation perhaps by driving a visualiser. Furthermore the mechanism can be used to drive an automatic garbage collection scheme, for example in the mark phase of a *mark-and-sweep* type garbage collection algorithm [130] that can be used to clear multiagent registries of terminated agents[26].

We implemented a prototype for aspect I and II for flat conversations[27] in order to evaluate the detection delays metric using an experimental setup based on simulated execution of an arbitrary protocol to make the experiment as general as possible and generate large datasets. We propose aspect IV for further work, and III as in the current setup we assume existence of the protocol libraries. Protocol libraries are also discussed in existing literature, though the is no standardised implementation of such libraries. We imagine that the core framework of the proposed set up can be implemented with existing finite state machine libraries e.g. [5] or related languages.

Agent registries and some aspects of visualisation components are implemented on most multiagent platforms and these visualisation components can be easily augmented with protocol visualisation primitives.

Finally, Appendix A, page 268 gives a discussion of the algorithm complexity issues for algorithms for predicates discussed in this chapter and the associated data structures.

[26]As remarked, the discussion of garbage collection in agents is outside the scope of this research, but maybe an interesting and natural consequent follow on research work.

[27]Regarding implementation of nested conversations, this will be dependent of the agent platform, for example JADE agent platform provides a construct called a behaviours, a non-deterministically scheduled multi-threaded construct, therefore this can be used.

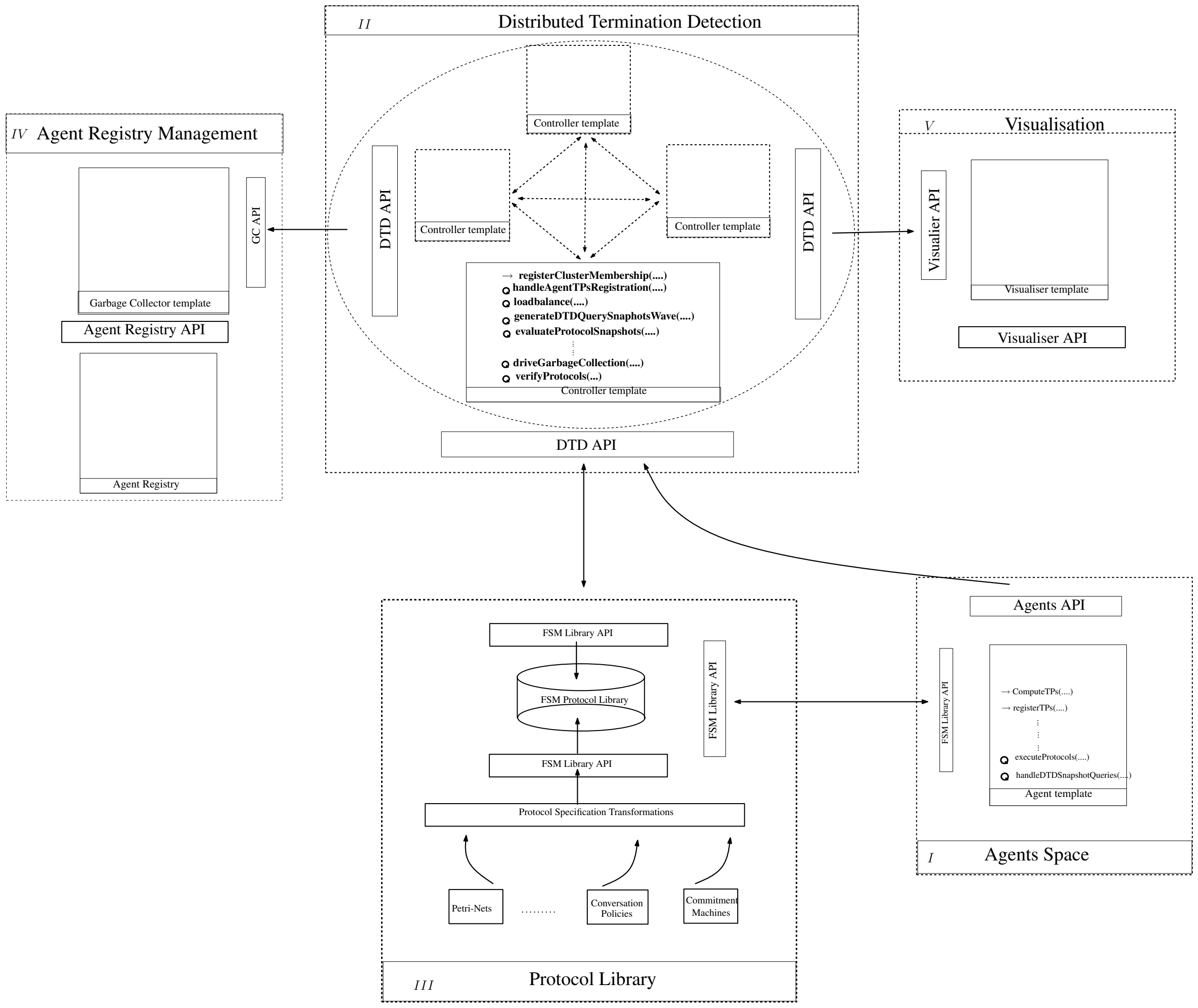


Figure 5.14: A proposed termination detection architecture for a multiagent system.

5.6 Summary and Contributions

We adopted a computational model where agents are autonomous [28], distributed, asynchronous processes that use an agent communication language to communicate and use interaction protocols for coordination and interaction with other agents in conversations to achieve their goals. And, agents form societies called multiagent systems that are modular distributed systems and have decentralized data and control.

We viewed protocols as behaviour specifications that are publicly viewable (whereas the individual agents' strategies that generate response utterances are private) and assume that protocols they are in the form of finite state machines, edge-labelled directed graphs , and to support homogeneity in a heterogenous multiagent environment, assumed existence of unified protocol framework where there are preserving transformations of other protocol specifications to finite state machines for the purposes of termination detection.

In this context we have;

– i – Presented definitions in relation to protocol graphs leading to the definition of minimal information in the form of shortest unique termination paths that agents can register with an observer of interactions.

– ii – Presented an off-line procedure and concrete algorithms to take as input a given protocol graph to produce a sub-protocol, a set TP of shortest unique termination paths given possibly multiple terminal state to represent this minimal information.

– iii – Presented a termination detection model. In the model we defined the notion of a conversation that encapsulates protocol execution as a basis of interactions

[28]With restrictions that agents have incentive to participate or there are enforceable conditions for participation in the society.

from an agent's perspective.

– iv – We modelled a branching conversation as a diffusing computation tree, and provided a definition of a data structure, a conversation matrix, *c-matrix*, a structure that can be used by controllers, entities that oversee conversations.

– v – Given this model and definitions, we presented a procedure for local termination of conversations and presented accompanying algorithms and some complexity analysis.

– vi – Explored possibilities for distribution, and presented a distributed protocol for termination detection over a cluster of controllers, either fully connected or coordinated through shared memory, a tuple space. Furthermore discussed practical considerations.

– vii – Identified the main metrics for evaluation and provided preliminary evaluation of the protocol given these defined metrics for evaluating the termination scheme.

– viii – Explored how the termination detection mechanism may fit in within a larger generic multiagent systems management infrastructure, potentially driving garbage collection of agent registries and interaction visualisation components.

– ix – Offered some perspective on the complexity issues of the algorithms proposed herein

Following the discussion here, in **Part III** next, **Chapter** 6 next discusses the prototype implementation, simulation, experimental design and the proposed data analysis for the experimental part of this thesis for the flat conversational model used to evaluate the termination detection mechanism and its configurations in an existing agent middleware implementation.

Then **Chapter** 7, page 131, presents results for the partial protocol configuration of the mechanism, followed by **Chapter** 8, page 157 that presents results for the full protocol configuration, followed by **Chapter** 9, page 175 that offers some perspective and exploration of the comparisons between these two setups.

Finally, **Chapter** 10, page 187 presents results for the distributed configuration.

PART III
EXPERIMENTS AND RESULTS

CHAPTER 6

Prototype implementation, experimental design and data analysis

6.1 Introduction

We have discussed in the previous chapter, **Chapter** 5, a termination detection mechanism for making termination of agent interactions explicit in a distributed setting and provided a general framework for implementation and defined three standard metrics for evaluating this mechanism.

This chapter follows this and discusses the prototype implementation , experimental setup , experimental design and data analysis methods to be used in this research.

The chapter provides an overview of a set of experiments to be considered and also discusses how the evaluation of these experiments will be done.

6.2 Overview

Section 6.3 starts by presenting an experimental setup for exploring the termination detection scheme. Here we present a sample protocol for the experiments and discuss the setup for the simulation. A simulation was chosen to make the evaluation as general as possible and generate large datasets in a controlled environment to aid exploratory comparisons between configurations. In addition this approach will provide a standard experimental environment for evaluating and comparison of future work on further mechanisms for termination detection.

The details of the prototype implementation are given in **Section** 6.4, where a popular FIPA compliant multiagent framework was chosen. The framework provides agent containers, agent communication language (FIPA-ACL) based messaging , protocol templates, distribution and inter-platform interoperability for agents.

Section 6.5 discusses the experiments to be conducted on the prototype implementation within the experimental setup, detailing various levels of quantitative experiments as briefly introduced in this section.

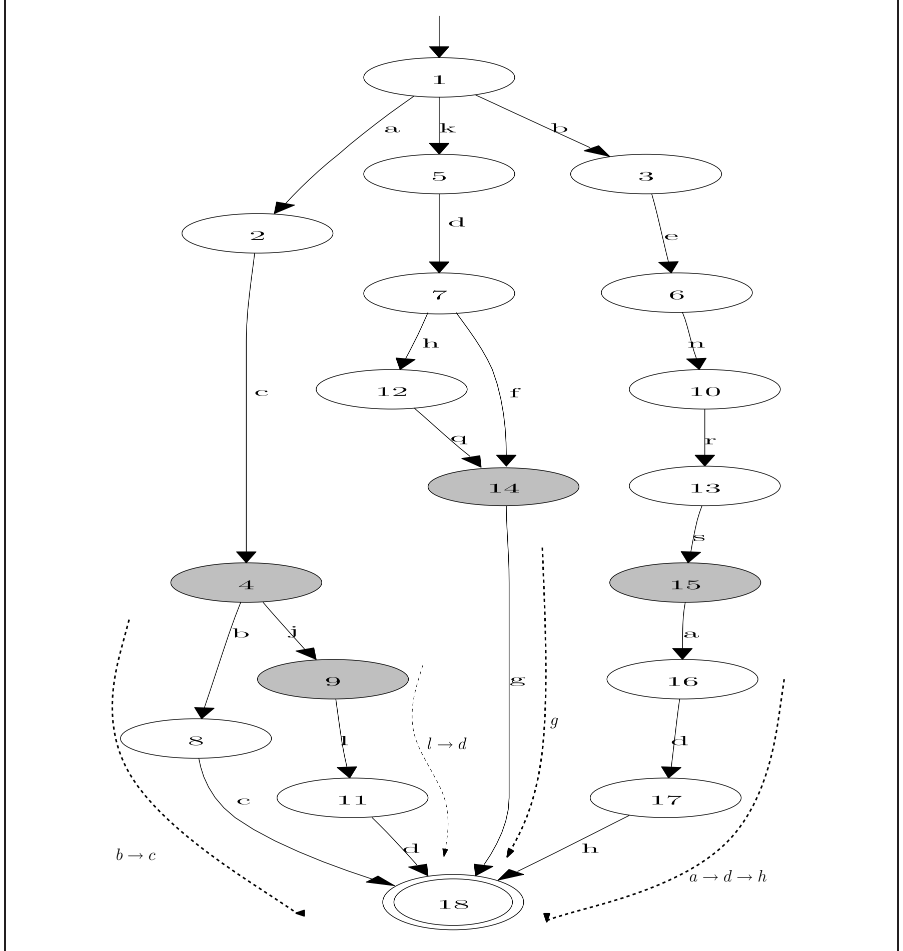
Section 6.6 presents the data collection and data analysis methods adopted. We opted to use resampling methods, e.g. *the bootstrap* for analysis of data sets, the rationale as explained in section 6.6 being that these methods do not make underlying distributional assumptions on the datasets ,e.g. normality, of the data sets, and also provide robust confidence intervals and offer a mechanism for treating outliers. The main cost however associated with these methods is that they are computationally intensive as they involve resampling a dataset that is treated as population. A separate theoretical overview of these methods is given in Appendix D or in the references provided.

Finally, **Section** 6.8 then provides a summary to reflect on this chapter.

6.3 Experimental setup

Details of the implemented aspects in the prototype are given section 6.4, next. So this section can be read in parallel with section 6.4.

For the experimental setup, consider a simulated scenario where agents execute a finite state machine based protocol as depicted in Figure 6.1 below. For all intents and purposes, this could equally have been any finite state machine representation of any protocol for any of the scenarios given in section 5.1, where transitions labels are ACL messages that make sense in a given scenario's protocol. In this example, protocol labels depicts contents of ACL messages.



(a) An arbitrary protocol for experimental analysis

Figure 6.1: An arbitrary protocol, showing observable states, observables *and* termination paths

Example 10. *In protocol shown in Figure 6.1, observable states are in set* $\{4, 9, 14, 15\}$

Using this protocol graph as input to an implementation of the off-line shortest termination path procedure, *stp*, (**Algorithm** 1 sketched in page 80), produces a partial protocol shown in Figure 6.2, a set of termination paths.

Example 11. *Given the protocol P in Figure 6.1, then for example, paths $p_1 = (4, 8, 18)$, $p_2 = (4, 9, 11, 18)$, $p_3 = (14, 18)$, $p_4 = (15, 16, 17, 18)$ are valid termination paths with labels (b, c), (j, l, d), (g), (a, d, h). These are shown in Figure 6.2*

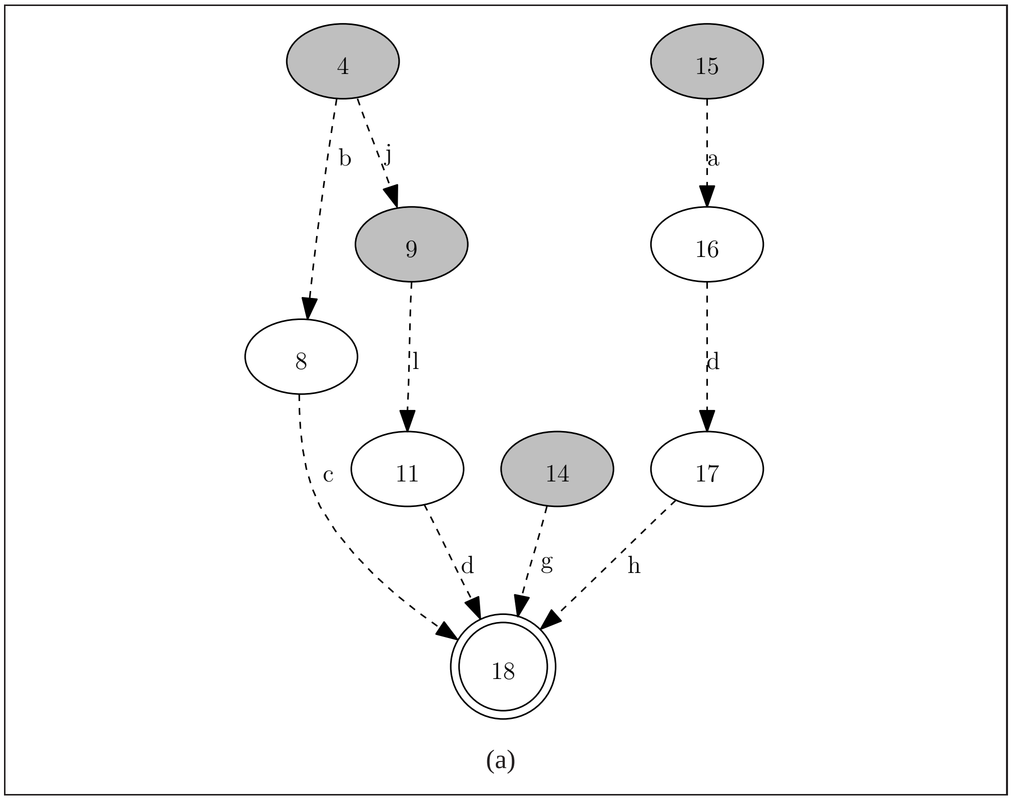


Figure 6.2: Showing unique termination paths for protocol in Figure 6.1

We implemented this protocol using JADE's [1] FSM behaviour template [24] that defines;

[1] An agent framework introduced in page 119.

1. `MessageTemplate` class which provides `MessageTemplate.MatchContent(Label)` operation for pattern matching protocol labels, e.g. `MessageTemplate mt = MessageTemplate.MatchContent(label);`

2. `registerFirstState(StartState)`, `registerState(State)`, `registerTransition(Transition)`, `registerLastState (LastState)` operations for the protocol fsm states and transitions. The operations accept states and transitions as arguments and model an fsm. The states themselves are modelled as JADE behaviours[2], i.e. in the case of protocol state, these were instances of a one off task implemented as instances of a `OneShotBehaviour`.

3. States process an incoming messages and match against a defined template, e.g. `ACLMessage msg = Agent.blockingReceive(mt)` and return next transition.

We implemented a simulation environment that instantiated agents and generated messages, protocol labels, $l \in \mathcal{L}$ to drive the agents. The simulation environment also scheduled and repeated experimental cycles.

Agents register with controllers and register protocol information according to experiments to be scheduled (see page 120 for various experiments), e.g. full protocol, partial protocol.

For data collection purposes, agents collect data on their side about the protocol execution, in particular *start* and *end* times, using system calls. These local time measurements form a basis for the detection delays metric used for the evaluation of the termination detection mechanism.

Equally, controllers on the other side, collect data about *end* times for the protocol execution as determined when *inactiveProtocol* (**Algorithm** 5, page 94) predicate evalu-

[2]Tasks that can be run in parallel, as mentioned in page 274.

ates to true for a given registered protocol or partial protocol.

In the implemented prototype, we consider a flat conversational scenario, given in **Example** 7, page 85, where conversations do not have descendants, which is more common usage scenario. Hierarchical scenarios using a diffusing computation tree to model conversations and using predicate *inactive* on the graph representing the diffusing computation tree can be easily implemented by maintaining a graph structure for JADE's behaviours that gets executed and dequeued on completion and testing for graph emptiness as for completion as discussed in **Algorithm** 5, page 94.

In the currently implemented setup, the controllers co-register to create a fully connected network [3]. Each controller maintains the data structures to store individual agents registrations and their protocol information. The connections between controllers allow individual controllers to forward agent registrations to other controllers in the cluster for load balancing purposes. The details of the distribution setup are given in chapter 10. Also because of the absence of global clocks in distributed systems, the issue of synchronisation is discussed together with that of network delays that may affect the distribution of detection delays in the distributed setting. The approach adopted for these issues is outlined in Appendix H.

6.4 Prototype implementation

An overview of the JADE agent framework used is given in Appendix B, page 273

Also a high level discussion of the prototype implementation is given in Appendix B, section B.2, page 275. The discussion there describes implemented processes[4]. Figures B.1, B.2, B.3 provide illustration of these various processes executed by agents

[3]We propose to evaluate the use of a share memory tuple space alternative of the architecture in future work.

[4]JADE provides a construct called a *behaviour* that can implement these agent processes.

and controllers.

6.5 Scheduled experiments

Overview The evaluation of termination detection schemes for multi-agent systems will be performed at various levels.

Level 1 Here the evaluation is done using two standard approaches, namely;

1. *Quantitative*; Evaluation of the termination detection scheme through the use of experiments for the defined detection delays metrics given the collected datasets followed by statistical analysis (discussed in section 6.6). The results for the all quantitative experiments are given in chapters 7, 8 and 10.

2. *Qualitative*; Evaluation of the termination detection through qualitative means, exploring non functional requirements and, issues raised by the use of these mechanisms , for example compromises in the agent autonomy assumptions, effects on interactions if any. The qualitative evaluation is done in section 11.1, page 230

Level 2 At this level the view is that of using the quantitative approach to explore the two configurations or approaches to observing protocol executions for termination detection, namely;

1. The *full protocol scheme*, where individual agents register with the monitor(s) full protocol specifications of the interaction protocols they are executing. The results for this are given in chapter 8.

2. The *partial protocol scheme* where agents only supply partial information about their protocols, i.e. termination paths, with the monitors. The results of this view are given in chapter 7.

Level 3 Here experiments can be viewed as coming from the two broad categories according to whether the monitoring is done locally (centralised) or distributed, i.e.

1. *Centralised*, where agents register and are observed by a monitor that resides in the same node as the agents and there is no network traffic for control messages. The results are given in chapters 7 and 8. The centralised experiments were done under a controlled setting on a dedicated host machine to allow exploratory comparison of level 2 experiments above . This exploratory comparison is considered in chapter 9, page 175.

2. *Decentralised* (distributed) where agents can register with remote monitors, and this involves message traffic over network interfaces. The results are given in chapter 10.

Quantitative experiments The experiments in this category are aimed at exploring termination detection schemes quantitatively to evaluate the defining metric of detection delays. This is done by collecting *detection delays* data for both co to highlight the underlying distribution of the detection delays parameter. The experiments will consider scalability to evaluate performance as the number of agents hosted is varied upwards.

Regarding the experimental setups, consider figures 6.3 and 6.4 below for the centralised and distributed architectures and also consider figures B.1 and B.4 discussed earlier. Agents execute an arbitrary public protocol. The individual agents record termination times Ta_is for the protocol they have registered to the *local* controller for monitoring and the controller records corresponding Tc_is , termination observed locally.

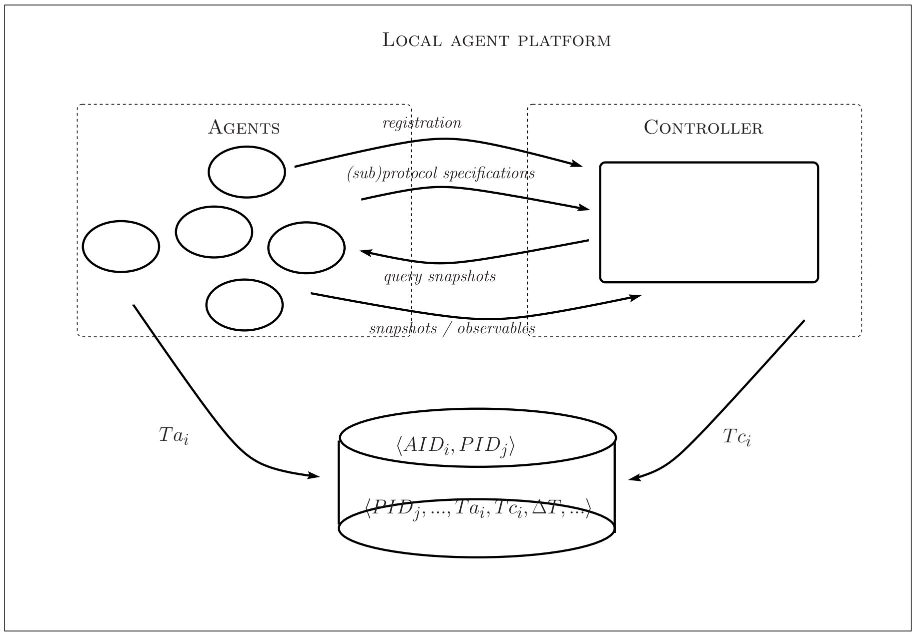

LOCAL AGENT PLATFORM
AGENTS
CONTROLLER
registration
(sub)protocol specifications
query snapshots
snapshots / observables
Ta_i
Tc_i
$\langle AID_i, PID_j \rangle$
$\langle PID_j, ..., Ta_i, Tc_i, \Delta T, ... \rangle$

Figure 6.3: Architecture for centralised experiments. Showing agents and the controller hosted in the same local agent platform

Centralised In this setup, experiments are run under a *controlled* environment on a single machine. This to provide a way of making objective comparison mechanism especially for scalability experiments.

Figure 6.3 gives the high level architectural setup for this experiments, showing agents and the controller hosted on the same local agent platform and interacting as detailed in figures B.1 to B.4. Detailed experimental results and analysis are given in chapters 8 and 7, and a comparison and hypothesis tests given in chapter 9.

Distributed In this setup experiments are run on hosts in a local area network with controllers forming a cluster on which agents hosted on individual hosts can register.

Figure 6.4 presents a high level architectural setup for this experiments where cluster controllers interact as also detailed in Figure B.3 in Appendix B

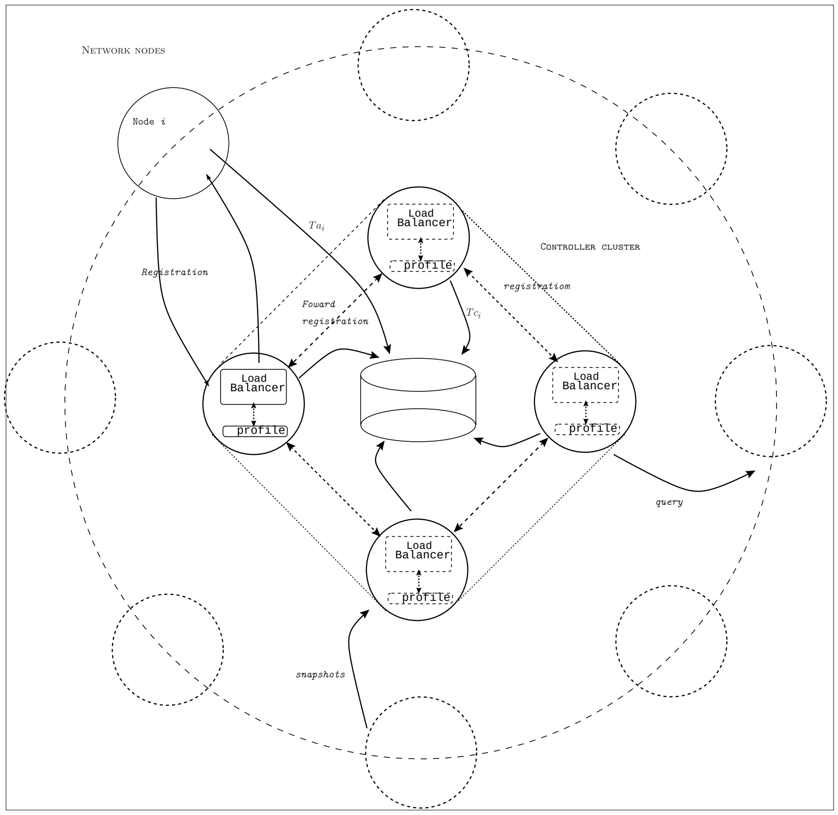


Figure 6.4: Distribution architecture, showing nodes hosting agents in a local area network and a cluster controllers in distributed agent platforms.

The details of this setup, including possible data collection scenarios and experimental results are presented in chapter 10.

Finally , some qualitative evaluation is given in section 11.1, page 230.

6.6 Data collection

Regarding data collection, see figures 6.5 for centralised experiments [5], where individual agents and monitors use a relational database to store details of the protocol execution.

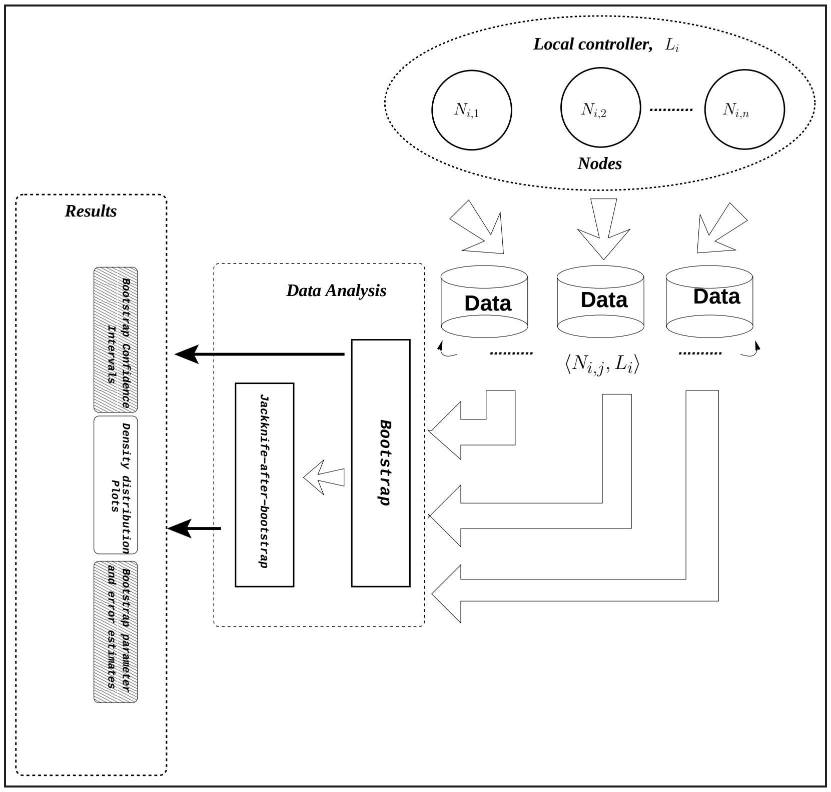

Local controller, L_i
$N_{i,1}$
$N_{i,2}$
$N_{i,n}$
Nodes
Results
Bootstrap Confidence Intervals
Density distribution Plots
Bootstrap parameter and error estimates
Data Analysis
Jackknife-after-bootstrap
Bootstrap
Data
Data
Data
$\langle N_{i,j}, L_i \rangle$

Figure 6.5: Data collection and analysis for scenario 1. Where the notation $\langle N_{i,j}, L_i \rangle$ signifies data (detection delays) logged on a particular node by a local controller. Each dataset is analysed separately by the bootstrap and jackknife-after-bootstrap (statistical procedures as described in section 6.7) to yield various results plots and tables shown in the figures that follow.

Also consider the following data collection setup, that describes various collection scenarios for evaluating the distributed setting. Scenario 1 was used also in the setup for the centralised experiments.

[5] And Figure C.3 in appendix C, page 284 for the distributed setting data collection setup.

Data collection setup and scenarios Consider a set of controllers $\mathcal{C} = \{R_i \mid i \in \mathbb{N}\}$ making up a cluster. Denote these as *remote* controllers. Also consider a set $\mathcal{C}_L = \{L_i \mid L_i \notin \mathcal{C}, i \in \mathbb{N}\}$, controllers not in the cluster. Denote these *local* controllers.

Consider a set of agents $\mathcal{A} = \{N_i \mid i \in \mathbb{N}\}$ and consider a set of agents $\mathcal{A}_L \subset \mathcal{A}$ that reside in the same host as *some* local controller, and consider a set of agents $\mathcal{A}_{Li} \subset \mathcal{A}_L$ be those agents registered with a local controller L_i.

Equally, consider also a set of agents $\mathcal{A}_R \subset \mathcal{A}$ be agents registered with *some* remote controller, and consider $\mathcal{A}_{Ri} \subset \mathcal{A}_L$ be those agents that reside in the same host as a controller L_i but registered with some controller $R_i \in \mathcal{C}$

Finally, consider a set of agents $\mathcal{A}_C \subset \mathcal{A}$ that reside in the same host as some remote controller, and consider $\mathcal{A}_{Ci} \subset \mathcal{A}_C$ be agents registered with an $R_i \in \mathcal{C}$.

Consider the illustration[6] of this given below in example 12.

Example 12. *Consider the setup below,*
$\mathcal{C} = \{R_1, R_2, R_3, R_4\}$, $\mathcal{C}_L = \{L_1, L_2\}$,
$\mathcal{A} = \{N_1, N_2, \ldots, N_{13}\}$
$\mathcal{A}_L = \{N_1, N_2, \ldots, N_8\}$,
$\mathcal{A}_R = \{N_4, N_7, N_8\}$, $\mathcal{A}_{L1} = \{N_5, N_6, N_7, N_8\}$,$\mathcal{A}_{L2} = \{N_1, N_2, N_3, N_4\}$, *and* $\mathcal{A}_{R1} = \{N_7, N_8\}$,$\mathcal{A}_{R2} = \{N_4\}$,*and*
$\mathcal{A}_C = \{N_9, N_{10}, N_{11}, N_{12}, N_{13}\}$,$\mathcal{A}_{C1} = \{N_9, N_{10}, N_{11}, N_{12}\}$, $\mathcal{A}_{C2} = \{N_{13}\}$,$\mathcal{A}_{C3} = \emptyset$, $\mathcal{A}_{C4} = \emptyset$

With this background , consider the following data collection scenarios;

1. *Scenario 1*: Agents monitored by their *local* controllers (dataset for $\mathcal{A}_{Li}$).

[6] And an alternative graphical depiction is given in Figure C.1 on page 281.

2. *Scenario 2*: A collective population of all agents monitored on *all* local controllers (dataset for $\bigcup_{\forall i} \mathcal{A}_{Li}$). This is used to give the distribution of detection delays on local controllers.

3. *Scenario 3*: Agents from a specific host registered to be monitored remotely in the cluster $\mathcal{C}$ (dataset for $\mathcal{A}_{Ri}$). This is used to give the distribution of detection delays for the host's agents and give a sense of the cluster's $\mathcal{C}$ performance as viewed from this host.

4. *Scenario 4*: A collective population of *all* remotely monitored agents in the cluster, (dataset for $\bigcup_{\forall i} \mathcal{A}_{Ri}$).

5. *Scenario 5*: A breakdown *per* cluster node of all agents monitored on that specific cluster node (dataset for $\mathcal{A}_{Ci}$). This to give a *per* cluster node centric view of distribution of detection delays, and to a certain extent the load characteristics for that cluster controller $R_i \in \mathcal{C}$.

Also regarding the simulation and experimental data collection, consider various scenarios for data collection during experimental cycles.

1. *Individual* detection delays recorded for repeated execution of the protocol over many cycles.

2. For every experimental cycle period, detection delays from the repeated execution of the protocol for that cycle recorded and some statistic calculated, i.e. treat each cycle as an experiment.

3. *Accumulated* detection delays recorded over the entire experiment for the repeated execution of the protocol and a statistic calculated over each period and the process repeated without discarding previous measurements but accumulating, see figure

An illustration is given in Figure C.2 in page 283

6.7 Data analysis

Regarding data analysis for quantitative experiments, resampling methods[7] were used, in particular the *bootstrap* [83], and the *jackknife-after-bootstrap* [85]. These methods are particulary relevant due to the non-normality of the detection delays data. These resampling methods were used to determine *distributions* of location parameters, and determine *standard errors* and calculate robust *confidence intervals* for the detection delays for all schemes . Appendix D provides the theoretical details and background for the bootstrap and justification of why these methods work and their suitability.

Briefly, procedures such as the bootstrap and jackknife are covered in discussions in the general subject of *robust statistical procedures* [207, 119, 217, 216]. Robust statistical procedures refer to statistical procedures which are not overly dependent on critical assumptions regarding an underlying population distribution. [52] observes that robustness is most commonly applied to methods that are employed when the normality assumption underlying an inferential statistical test is violated.

It is pointed out that though when sample sizes are reasonably large certain tests such as the *single-sample t-test* and the *t-test* for two independent samples are known to be robust with respect to violation of the normality assumption (i.e., the accuracy of the tabled critical alpha values for the test statistics are not compromised), if the underlying distribution is not normal, the power of such tests may be appreciably reduced [207], p.327.

[7] Also referred to as computer-intensive methods. These methods are becoming increasingly attractive with improvement in computational resources and availability of their implementations in statistical packages. Compared to Monte Carlo methods resampling methods use the available dataset and treat it as a population from which samples can be taken, whereas by contrast in monte carlo methods samples are drawn from theoretical probability distributions

Related to this is the fact that, as observed in [219] regarding the power of commonly employed goodness-of-fit tests for normality, unless a sample size is relatively large, goodness-of-fit tests for normality (such as the Kolmogorov-Smirnov goodness-of-fit test for a single sample or the chi-square goodness-of-fit test) will generally not result in rejection of the null hypothesis of normality, unless the fit with respect to normality is dramatically violated. Consequently, some researchers conclude that most goodness-of-fit tests are ineffective mechanisms for providing confirmation for the normal distribution assumption that more often than not researchers assume characterizes an underlying population.

It is also argued in [219] that as a result of the failure of goodness-of-fit tests to reject the normal distribution model, procedures based on the assumption of normality all too often are employed with data that are derived from non-normal populations. In instances where the normality assumption is violated [219] the researcher is encouraged to consider employing a robust statistical procedure (such as the bootstrap) to analyze the data. In accordance with this view [217] notes that the bootstrap will often yield a more accurate result for a non-normal population than will analysis of the data with a statistical test which assumes normality.

Another characteristic of data that is often discussed within the framework of robust statistical procedures is the subject of *outliers*. Research has shown that a single outlier can substantially compromise the power of a parametric statistical test. [219] provides an excellent example of this involving the single-sample *t-test*. Various sources suggest that when one or more outliers are present in a set of data, a computer-intensive procedure (such as the bootstrap or jackknife) may provide more accurate information regarding the underlying population(s) than a parametric procedure.

In the literature, bootstrap procedures and algorithms are described in the standard reference on the Bootstrap [83], the more practical aspects are discussed in [69] while the more theoretical discussions can be seen in [204] and [105].

The general procedure for performing the bootstrap is shown in Appendix D, Algorithm 9 in page 288 for example bootstrapping the mean. The algorithm repeats the process of drawing samples with replacement from the data[8] and calculating the required metric and displaying the distribution of the bootstrap replicates [9].

Equally we can use the bootstrap to compute various types of bootstrap confidence intervals, e.g. *BCa* (Bias corrected) and *Bootstrap-t*, *ABC*. A review of bootstrap confidence intervals in also given in Appendix D. In the literature treatment is given in [84, 82] [74, 104, 104].

Statistical software routines exist in statistical packages like R, Matlab , S-Plus and others. Figure 6.5 shows the data analysis phase where data was retrieved by R and Matlab engines for data analysis using resampling methods.

Chapters 7, 8 and 10 present experimental results and detailed analysis using the bootstrap.

Finally as a result of the non normality of detection delay data, nonparametric hypothesis tests were used when seeking to compare the two configurations in level 2 experiments, (full and partial protocol schemes). That is, a non-parametric equivalent of the parametric t-test has been used, namely Kruskal-wallis. The details and results of these test can be seen in chapter 9.

6.8 Summary and contributions

Following the discussion in **Chapter** 5 on the proposed mechanism for termination detection, this chapter has;

[8] That is treated as a population.
[9] Bootstrap "replicates" is the standard term used

– i – Presented an experimental setup for exploring the termination detection scheme, a setup that can provide a basis for a standard experimental environment for evaluating and comparisons of future work on further mechanisms for termination detection.

– ii – Presented a concrete prototype implementation based on a widely used FIPA compliant multiagent framework.

– iii – Discussed the experiments to be conducted on the prototype implementation within the given experimental setup, detailing various levels of quantitative experiments and metrics to be evaluated.

– iv – Discussed data collection scenarios for the centralised and distributed configurations and proposed data analysis methods to be adopted, detailing the use resampling methods, e.g. *the bootstrap* for analysis of datasets and for providing robust confidence intervals.

Chapter 7 next presents results for partial protocol experiments, i.e. experiments where agents register minimal information, sub-protocols or shortest termination paths about their protocol execution to controllers.

CHAPTER 7

Partial Protocol Results

The previous chapter, **Chapter** 6 presented an experimental setup, prototype implementation, experimental design and data analysis following the discussion in **Chapter** 5 of a termination detection mechanism for agents.

7.1 Introduction

This chapter presents results for the partial protocol experiments as described in section 6.5, page 121. Recall that these were a set of controlled experiments[1] where and increasing number of agents were hosted and monitored on a local controller. Agents register and submit partial information about the public protocol being executed. The chapter provides detailed exploratory data analysis of the datasets and proceeds to performing bootstrap analysis and then derives robust bootstrap confidence intervals for the detection delays metric in this experimental set. The next chapter, chapter 8 provides the corresponding results for the full protocol setup.

A note on data analysis and presentation This chapter first conducts exploratory data analysis, on the dataset. For basic summary statistics of the datasets, tables of the type shown in Table 7.1 are presented to summarise results for all agent experiments. The summary statistics were calculated using DATAPLOT statistical package [90].

[1] Conducted on a dedicated host machine.

Regarding *exploratory data analysis*, typically the relevant graphs are those shown in a 4-plot e.g. Figure 7.1 used for initially testing whether the four underlying initial assumptions[2] of a typical measurement process hold. The plot consists of;

- A *run sequence plot* (time series), to give an indication of any significant shift in location or scale of the data over the period of the experiment and identify any outliers.
- The *lag plot* can be used to check the randomness of the time series data. Random structure of the data indicates that the underlying data is randomly generated by a random process. This is not particulary important for our purposes as we are not evaluating the randomness of the measured detection delays time series. We include this here for completeness of the standard 4-plot.
- *Histogram* [43] To visualise the distribution of data and explore the symmetry (skewness), spread, center and check for heavy tails and outliers. Symmetrical data with no significant outliers and heavy tails may indicate normality. The histogram can also show the presence of multiple modes in the data. All these can give an idea of an appropriate distributional model if required.
- *Normal probability plot* [43]. Used to verify any assumptions of normality of the data. The data are plotted against a theoretical normal distribution in such a way that the points should form an approximate straight line. Departures from this straight line indicate departures from normality.

We have performed standard normality tests on the datasets, see Appendix E, page, 308 and Table 7.2, page 143. These tests confirm that the detection delays dataset is non-normal, as could also confirmed by visual inspections of qqplots and histograms.

[2]For example fixed distribution, location, variance etc.

Therefore we have as discussed in chapter 6, decided not to make distributional assumptions about the data and instead use non-parametric and re-sampling methods such as the bootstrap for all statistical analysis in these datasets.

The bootstrap procedure was performed using the statistical package R [122, 197] and its libraries *boot* and *bootstrap*.

For the bootstrap results, we present;

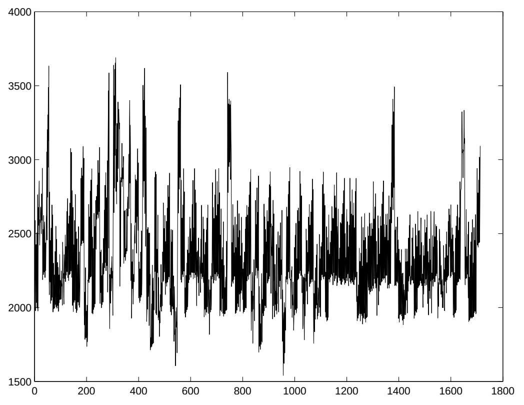
1. Figures of the type shown in figures 7.6 displaying distributions of bootstrap replicates of the mean detection delays.

2. Corresponding qqplots for these bootstrap replicates.

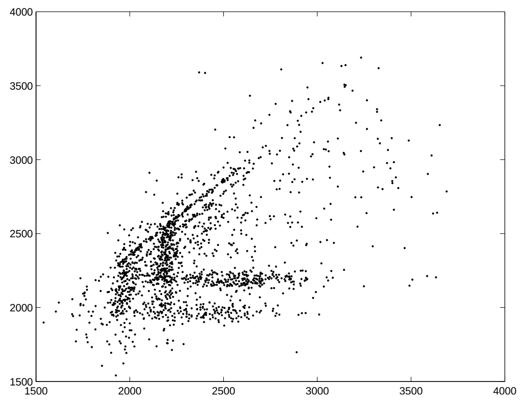
3. A summary table for the bootstrap analysis for all experiments with agent numbers varied. The table shows numerical results for parameter and error and bias estimation. It also shows results for the non-parametric bootstrap confidence intervals. Bootstrap confidence intervals were calculated using the R package *bootstrap* [40].

7.2 Results: Exploratory Data Analysis

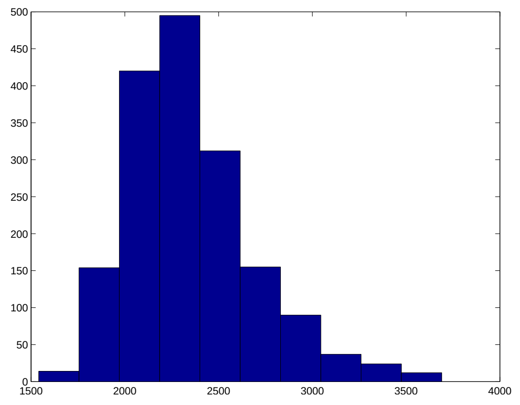
As a starting point, Figure 7.1 shows the *4-plot* for an experiment where 5 agents were instantiated for this partial protocol scheme where all measurements are in $millseconds$ as obtained from a unix system call. Inspection of the plots can be used to test underlying assumptions about the data. As it is, the figure shows the data to be *non-normal* as highlighted by the skewness in the histogram (c), fat tails and departures from the straight line in the normal probability plot (d). The data are *random* as there is no inherent structure in the lag plot (b). Inspecting the time series run plot (a) The data seems not to have a fixed variation when observed across long periods of time as was done in the experiments and as plotted here.



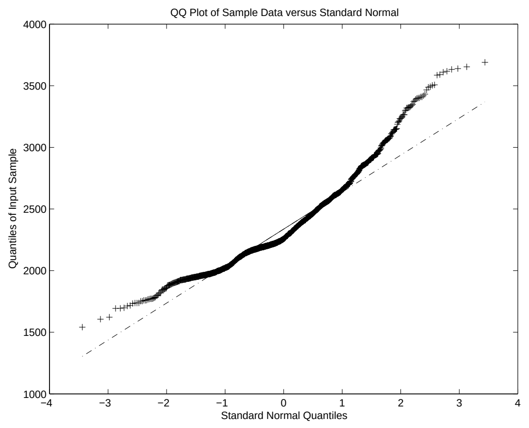
(a) Run sequence plot, showing delays in milliseconds (y-axis) and experimental cycle (x-axis)



(b) Lag plot, showing delays (x-axis and y-axis) in milliseconds



(c) Histogram, showing distribution of delays (y-axis) in milliseconds and bins (x-axis)



(d) Normal probability plot, showing delays in milliseconds (y-axis) and quartiles (x-axis)

Figure 7.1: Figures a-d show elements of the 4-plots for the purposes of exploratory data analysis for centralised experiments using the partial protocol scheme where 5 agent were hosted. All measurements are in ***milliseconds****.*

The rest of the 4-plots for other experiments with agent numbers varied from 10 to 100 are given in Appendix in figures F.1 and F.2 from page 321. Again the figures there confirm the observations highlighted above.

Furthermore we can consider additional plots for descriptive statistics, such as the boxplot [234] to give the *5-number* summaries[3] given in Table 7.1 in a graphical format for visual inspection and also to see outliers, and get a sense of the data dispersion. Boxplots can also be used for comparison between datasets.

We can also present the plot of the cumulative density function that describes the probability density function of random variable X.

$$x \mapsto F_X(x) = P\left(X \leq x\right) = \int_{-\infty}^{x} f\left(t\right) dt \tag{7.1}$$

As example consider Figure 7.2 showing the box plots and the cdf plots given together with the histogram and series data plots for the agent experiment where 5 agents were hosted. Again, a quick inspection confirms non normality and the presence of outliers.

The rest of the figures for other experiments are given in figures 7.3 to 7.4.

A note about those figures;

1. The parameter for agent numbers was chosen for the experiments to explore scalability as discussed in the experimental design, That is, we wish to explore how detection delays vary with an increase in the number of agents hosted in the agent platform, i.e. to explore whether it is linear , exponential, etc.

2. Regarding the histograms in the figures, the bin size chosen affects the visual appearance of the distribution of the data, therefore visual inspection alone is not sufficient to establish the underlying normality of the data. This can only be

[3]Minimum, Mean, Maximum,Median, Quartiles.

verified by standard normality tests [131, 222, 229]. The results of the normality tests [4] are shown in Table 7.2

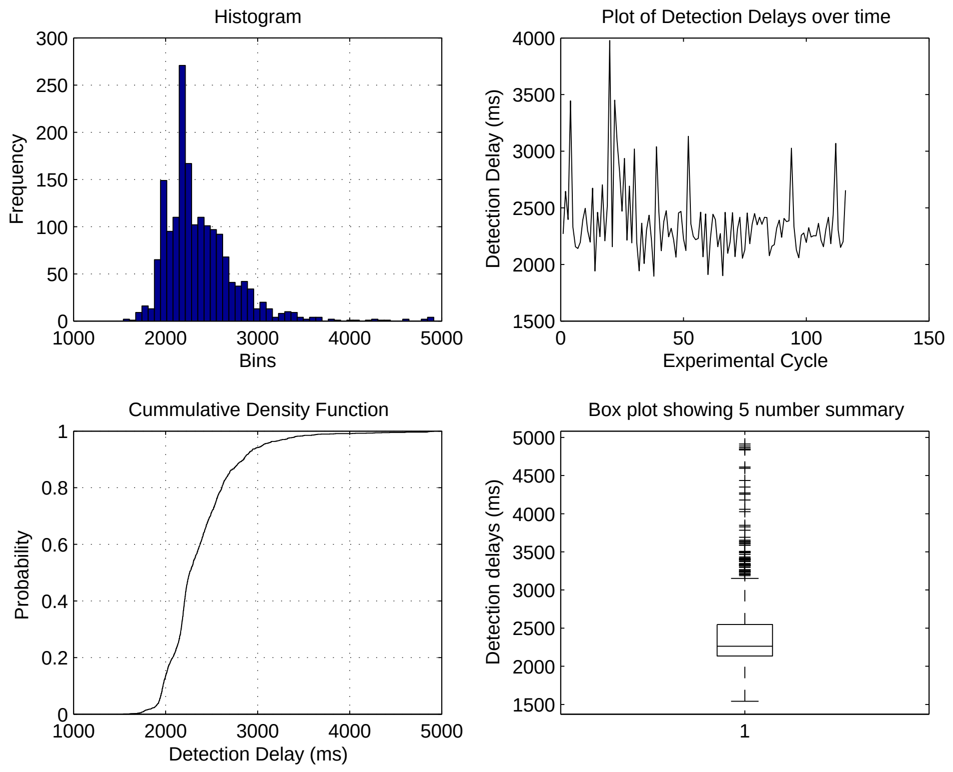
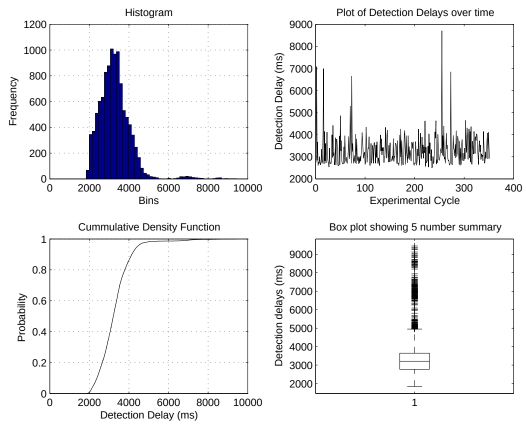
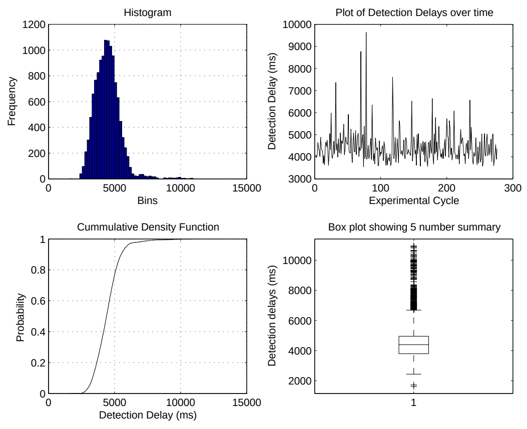


Figure 7.2: Showing plots for descriptive statistics and exploratory data analysis for centralised experiments using the partial protocol scheme for the experiment with 5 agents hosted. For example, the Box plot shows the 5 number summary, e.g. Median of around 2200 ms, and upper and lower quartiles on either side of the Median, and Maximum and Minimum values. Also showing outliers. All measurements are in ***milliseconds****.*

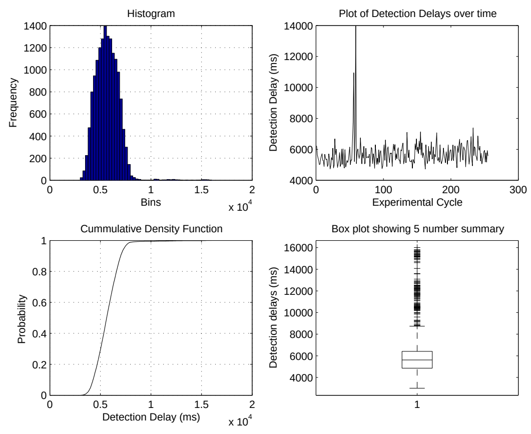
[4]All quantities for normality test are standard and definitions can be found in the given references together with interpretation of results. Most statistical tools provide implementation.



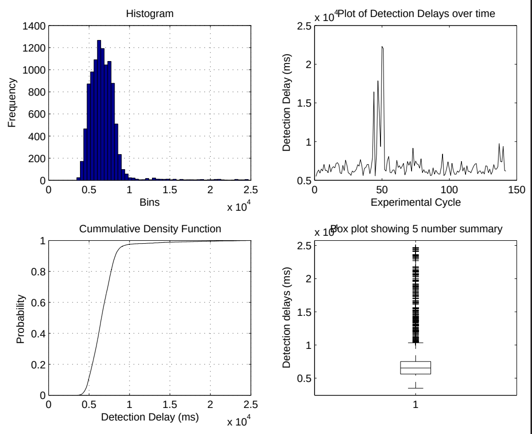
(a) 10 agent experiment



(b) 15 agent experiment

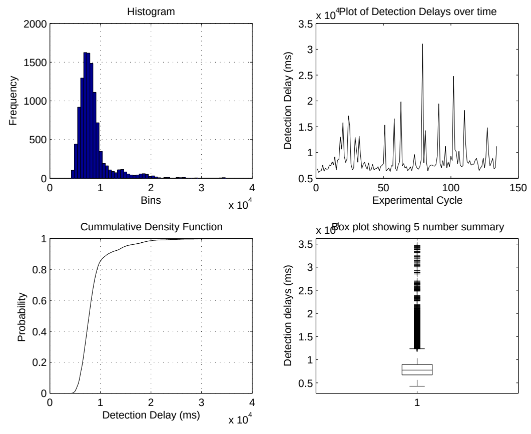


(c) 20 agent experiment

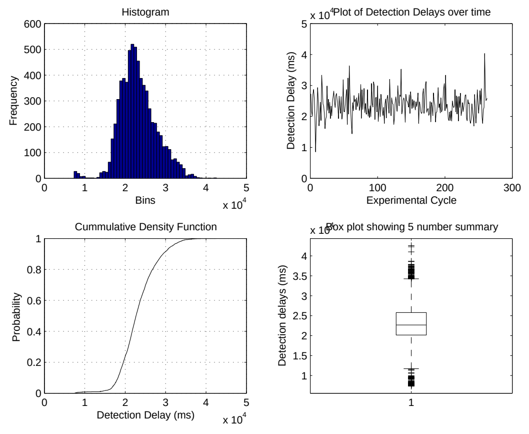


(d) 25 agent experiment

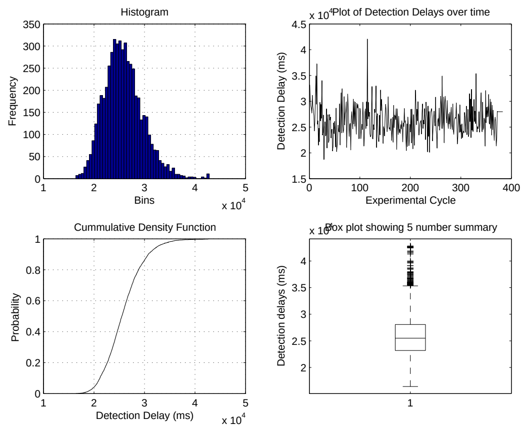
Figure 7.3: Figures (a)-(d) show plots for descriptive statistics and exploratory data analysis for centralised experiments using the partial protocol scheme where agent numbers were varied from 10 through to 25. All measurements are in ***milliseconds***



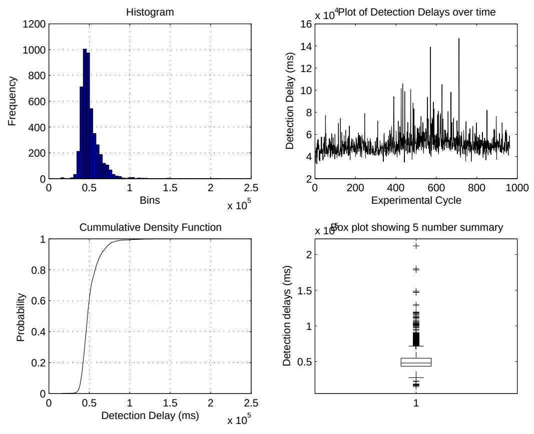
(a) 30 agent experiment



(b) 40 agent experiment



(c) 50 agent experiment



(d) 100 agent experiment

Figure 7.4: Figures (a)-(d) show plots for descriptive statistics and exploratory data analysis for centralised experiments using the partial protocol scheme where agent numbers were varied from 30 through to 100. All measurements are in ***milliseconds***

Table 7.1 presents summary statistics for all experiments in this setup. The table presents; *location measures*; to find a central value that describes the data, *dispersion measures;* to capture the spread in the data; *randomness measures* ,distributional measures; The third and fourth moments are the skewness and kurtosis of the distribution. The table also complements 4-plot figures above by exploring properties of the detection delays data by presenting some distributional measures [5]. For example, the probability plot correlation coefficient, PPCC [91] can be used to identify the shape parameter for a distributional family that best fits the data [181]. DATAPLOT ^{TM}produces PPCC values for the distributions shown under distributional measures in Table 7.1. Again, the distributional measures strongly point to the fact that the detection delays data are not from a normal distribution.

Regarding repeatability of experiments, how many experiments were carried out, detailed analysis of variance and comparisons between experiments, these are all discussed in detail in chapter 9, but briefly, for each experiment, 10 experimental runs with each experimental run spanning an experimental period of about 10 hours partitioned into experimental cycles of about 10 mins.

For completeness, additional standard normality tests [6] were carried out and results are given in Table 7.2. All quantities for normality tests as presented in Table 7.2 are standard and their definitions can be found in [131, 222, 229], together with interpretation of results. All tests show that the data is non-normal.

[5]To characterise properties of the data, e.g. shape.
[6]Most statistical tools implement these tests.

SUMMARY STATISTICS									
	AGENT EXPERIMENTS								
	5	10	15	20	25	30	40	50	100
	LOCATION MEASURES								
Midrange	0.2615	0.3966	0.5406	0.6264	0.6544	1.170	2.3002	2.8856	5.3735
Mean	0.2347	0.3118	0.4406	0.5419	0.6392	0.7959	2.2309	2.5469	4.7220
Midmean	0.2286	0.3162	0.4334	0.5615	0.6486	0.7384	2.2826	2.4961	4.7301
Median	0.2257	0.3096	0.4338	0.5313	0.6309	0.7555	2.1874	2.5298	4.5053
DISPERSION MEASURES									
Range	0.2149	0.4246	0.5925	0.6695	0.5271	1.486	3.1345	2.4870	7.2360
Stand. Dev	0.03363	7.2346	9.4198	0.1155	0.1100	0.2490	0.5009	0.4203	0.88013
Av. Ab. Dev	0.2555	0.5850	0.7227	0.09032	0.09057	0.1669	0.3788	0.3389	0.6533
Minimum	0.1541	0.1843	0.2443	0.2917	0.39090	0.4270	0.7329	1.6421	1.7555
Lower Quart	0.2134	0.2543	0.3712	0.4569	0.5581	0.6406	1.9477	2.2201	4.1030
Lower Hinge	0.2134	0.2543	0.3712	0.4571	0.5582	0.6407	1.9482	2.2211	4.1033
Upper Hinge	0.2539	0.3567	0.4942	0.6118	0.7206	0.8659	2.5424	2.8251	5.1080
Upper Quart	0.2539	0.3567	0.4944	0.6118	0.7207	0.8663	2.5426	2.8251	5.1084
Maximum	0.3690	0.6089	0.8368	0.9612	0.9180	1.913	3.8674	4.1291	8.9915

RANDOMNESS MEASURES									
Autoco coef	0.4599	0.4923	0.3192	0.2968	0.02397	0.7335	0.7816	0.53977	0.2018
	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
DISTRIBUTIONAL MEASURES									
St. 3rd Mom	0.9998	0.4268	0.9279	0.7934	0.1413	1.7548	-0.2721	0.3794	0.1187
St. 4th Mom	4.2160	2.8806	4.7044	3.8961	2.3011	6.8150	3.8474	2.8459	4.6144
St Wilk-Sha	63.5385	-23.675	-51.244	-43.0298	-15.749	-178.219	-29.879	-11.825	-88.798
Uniform ppcc	0.9405	0.9800	0.9434	0.9560	0.9913	0.8692	0.9496	0.9805	0.9239
Normal ppcc	0.9703	0.9886	0.9759	0.9803	0.9941	0.9194	0.9852	0.9936	0.9558
Tuk -.5 ppcc	0.7746	0.7450	0.7862	0.7681	0.7275	0.7605	0.7844	0.7583	0.7783
Cauchy ppcc	0.3472	0.3182	0.3529	0.3333	0.2905	0.3415	0.3423	0.3259	0.3732

Table 7.1: Summary statistics for all agents experiments in the centralised setup for partial protocol scheme, showing location, dispersion and distributional measures

Distributional Normality Tests										
Anderson-Darling		$A = -n - \frac{1}{n}\sum_{i=1}^{n}[2i-1][ln(p_{(i)}) + ln(1-p_{(n-i+1)})]$								
Statistic	A	26.99178	6.518426	11.87086	11.07163	4.86468	68.7629	11.86567	4.23203	43.77127
	P-value	3.160e-61	5.45e-16	1.80e-28	1.23e-26	4.85e-12	1.68e-132	1.85e-28	1.61e-10	3.00e-93
Conclusion		REJECT								
Wilkson-Shapiro										
Statistic	W	0.94143	0.97707	0.95221	0.96082	0.98741	0.84531	0.97045	0.98697	0.91408
	p-value	1.71e-25	6.40e-16	3.27e-23	4.30e-21	4.58e-11	5.90e-38	2.85e-18	2.57e-11	4.23e-30
Conclusion		REJECT								
Shapiro-Francia		$W = \left(\sum_{i=1}^{n} a_i x(i)\right) / \sum_{i=1}^{n}(x_i - \overline{x})$								
Statistic	W	0.941533	0.977359	0.95234	0.96110	0.98803	0.84541	0.97064	0.98726	0.91364
	p-value	3.34e-23	1.45e-14	3.70e-21	3.17e-19	5.60e-10	2.60e-34	1.03e-16	2.21e-10	2.20e-27
Conclusion		REJECT								
Lillie(Kolg-Smir)		$D^{+} = \max_{i=1,\dots,n} i/n - p_{(i)}, D^{-} = \max_{i=1,\dots,n} p_{(i)} - (i-1)/n$								
Statistic	D	0.107311	0.040742	0.041497	0.04118	0.03484	0.14700	0.05665	0.034663	0.10399
	p-value	2.25e-53	5.61e-07	2.98e-07	3.89e-07	4.96e-05	2.10e-102	5.55e-14	5.61e-05	5.35e-50
Conclusion		REJECT								
Jarque-Bera		$JB = \frac{n}{6}\left(S^2 + \frac{(K-3)^3}{4}\right)$								

Statistic	X^2	391.5488	53.03562	453.9846	237.4310	40.43232	1920.713	72.68503	42.78078	589.0186
	p-value	0	3.04e-12	0	0	1.66e-09	0	1.11e-16	5.13e-10	0
Conclusion		REJECT								
PEARSON		$P = \sum(C_i - E_i)^2/E_i$								
Statistic	P	520.8587	231.6351	142.4349	166.3462	98.1617	625.0502	205.6223	108.9965	465.7974
	p-value	1.06e-86	5.15e-30	2.72e-14	2.53e-18	1.92e-07	6.05e-108	2.91e-25	5.07e-09	1.37e-75
Conclusion		REJECT								
CRAMER-VON MISES		$W = \frac{1}{12n} + \sum_{i=1}^{n}(p_{(i)} - \frac{2i-1}{2n})$								
Statistic	W	4.648875	0.655082	1.28574	1.29745	0.73091	10.99672	1.54436	0.58801	7.43111
	p-value	6.91e+51	1.35e-07	3.75e-10	3.70e-10	3.87e-08	Inf	6.43e-10	4.60e-07	7.29e+197
Conclusion		REJECT								

Table 7.2: Showing results of a number of normality tests on all datasets for experiments where the agent numbers monitored was varied from 5 to 100. All normality tests reject the hypothesis that the data is normally distributed as evidenced by low p-values, i.e. $p - values \ll 0.05$. All quantities for normality test are standard and definitions can be found in [131, 222, 229]

7.3 Confidence intervals, standard errors and the Bootstrap

Summary statics as given in Table 7.1, give point estimates of the statistic of interest, e.g. mean of detection delays for a given experiment.

We are not only interested in obtaining a point estimate of a statistic but also the *confidence interval* for the true value of the parameter and some estimate of the *variation* in this point estimate. For example, we wish to calculate not only a sample mean , but also the standard error of the mean and a confidence interval for the mean.

Commonly, data analysis has relied on the *central limit theorem* [89] [7] and normal approximations to obtain standard errors and confidence intervals.

But as discussed earlier, the available literature stipulates that these techniques are valid only if the statistic, or some known transformation of it, is *asymptotically* normally distributed. Hence, if the normality assumption does not hold as we have just seen on the normality tests, then the traditional methods should not be used to obtain confidence intervals.

A major motivation for the traditional reliance on normal-theory methods was been computational *tractability*. Now, with the high availability of computational resources, there is an alternative to using asymptotic theory to estimate the distribution of a statistic. This alternative is resampling methods [8] which can be used to return inferential results for either normal or non-normal distributions.

In this section we would like to determine the confidence intervals and standard er-

[7]CLT is a profound result in statistics, simply put, it stipulates that the distribution of the mean tends to be *normal*, even when the distribution from which the mean is computed is decidedly *non-normal*. The closer the parent distribution is to a normal distribution, the smaller is the required sample size for this to hold. Larger sample sizes are required from parent distributions with strong *skewness*s and/or strong *kurtosis*.

[8]Estimating the precision of sample statistics (medians, variances, percentiles) by using subsets of available data (jackknife) or drawing randomly with replacement from a set of data points (bootstrapping).

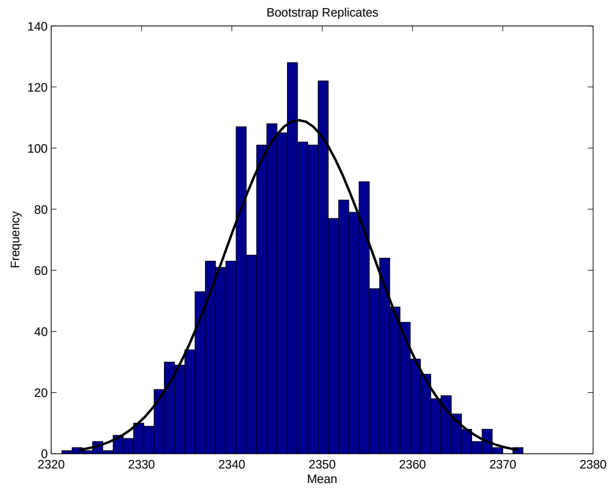
rors of the mean for the delays data without making any assumptions[9] regarding such statistics. Resampling [10] methods such as "*The Bootstrap*" and "*The Jackknife*" allow this to be done. These methods provide estimates of the bias, standard error, confidence intervals, and distribution for any statistic. A self contained review of bootstrap procedures is given Appendix D.

For the confidence intervals, we have explored the *BCa* and Bootstrap-t bootstrap confidence intervals [74]. The general procedure for bootstrap confidence intervals is also given in Appendix D in pages 293-293.

7.3.1 Bootstrap results

Replicates distributions Figure 7.5 shows the distribution of bootstrap replicates for the mean for an experiment in which the number of agents was set to 5. Figures 7.6 and 7.7 present the plots for the rest of the experiments with agent numbers varied from 10-100.

[9]The only assumption here is that the data is representative of the underlying population.
[10]Resampling refers to the process of drawing samples from original data.



(a) 5 agent experiment

Figure 7.5: Figures shows bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where 5 agent were hosted (in milliseconds)

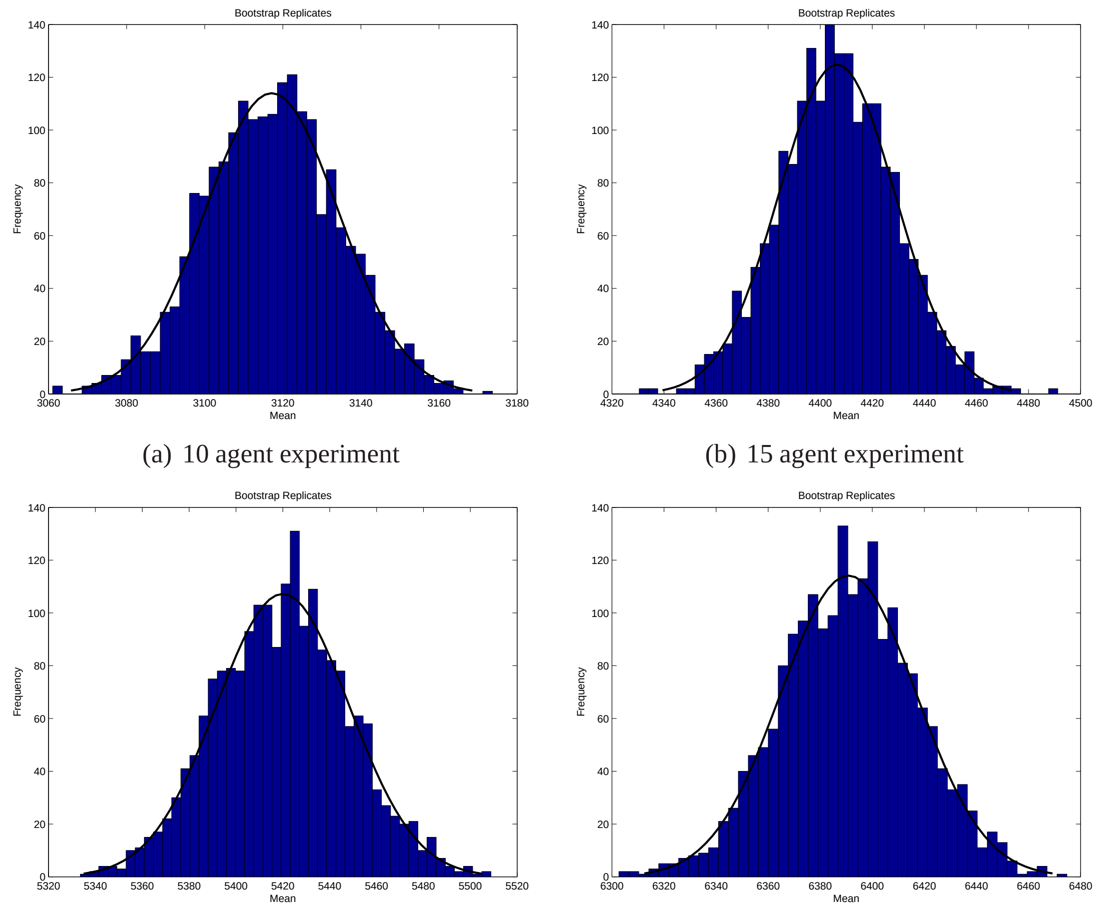
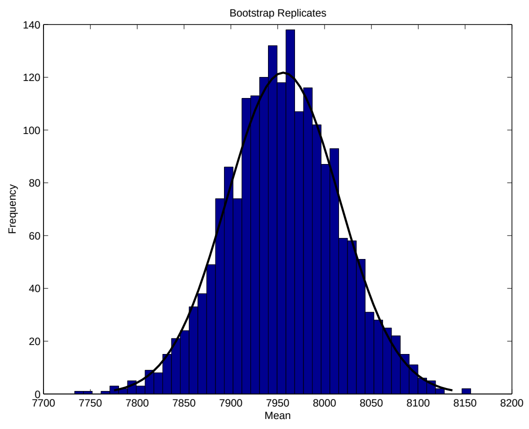
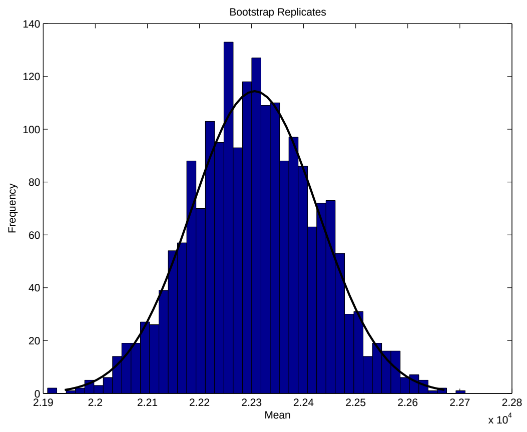


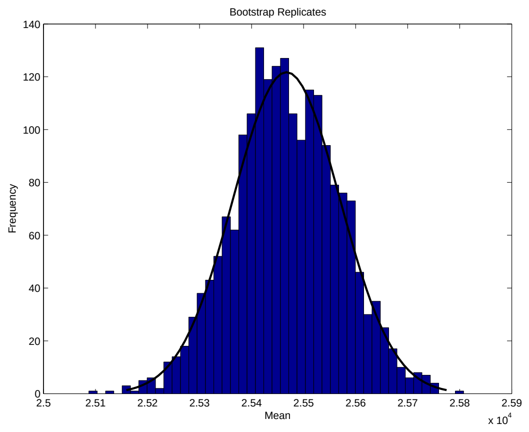
Figure 7.6: Figures (a)-(d) show bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where agent numbers were varied from 10 through to 25 (in milliseconds)



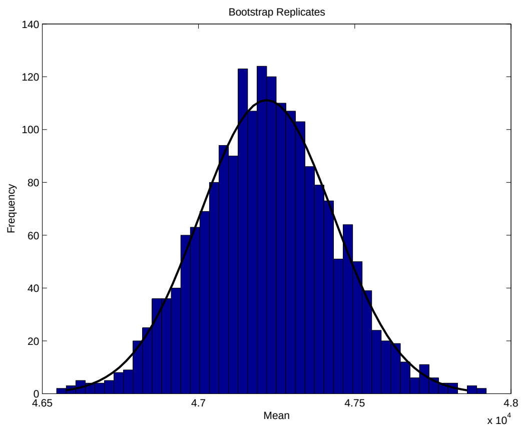
(a) 30 agent experiment



(b) 40 agent experiment

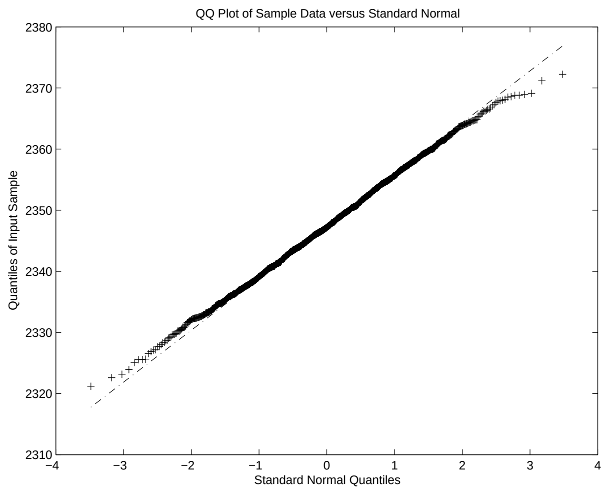


(c) 50 agent experiment



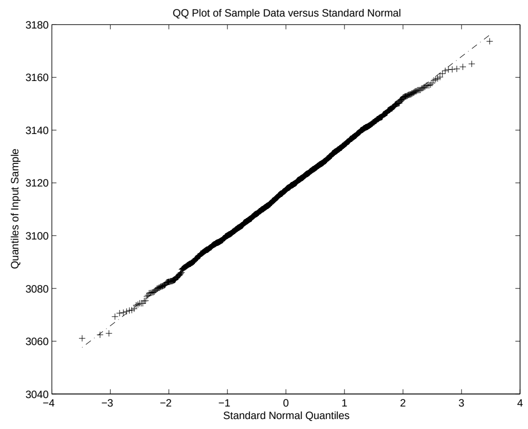
(d) 100 agent experiment

Figure 7.7: Figures (a)-(d) show bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where agent numbers were varied from 25 through to 100

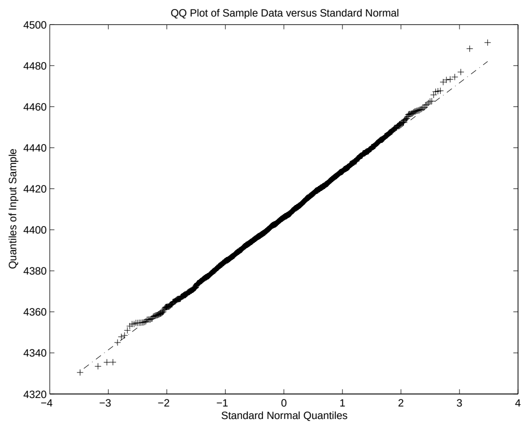


(a) 5 agent experiment

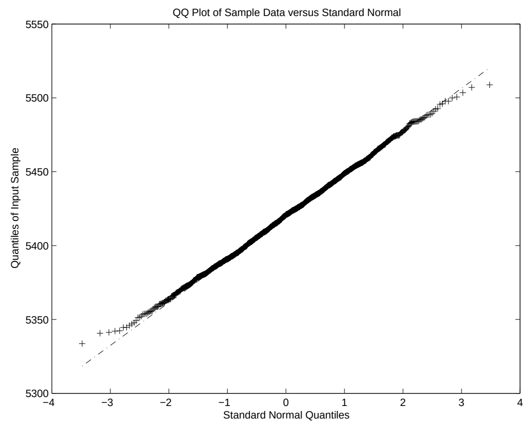
Figure 7.8: Figures shows QQ plot of the bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where 5 agent were hosted



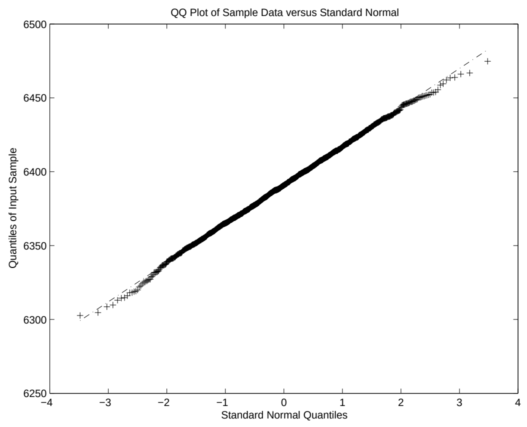
(a) 10 agent experiment



(b) 15 agent experiment

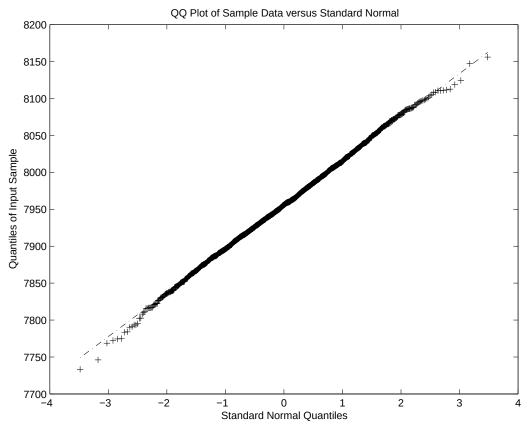


(c) 20 agent experiment

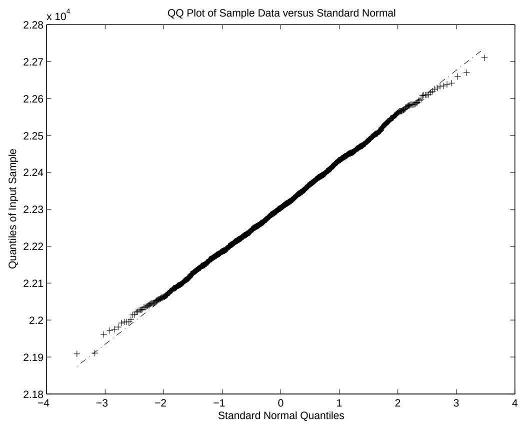


(d) 25 agent experiment

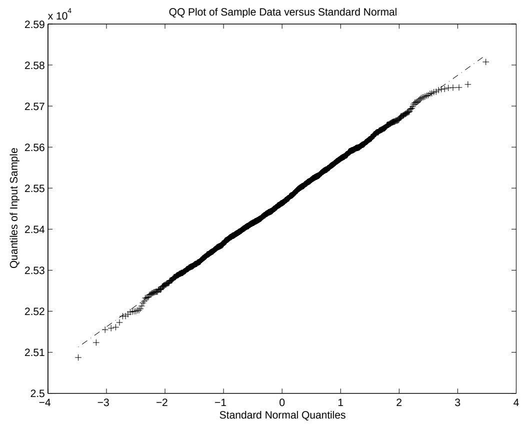
Figure 7.9: Figures (a)-(d) show qqplots of bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where agent numbers were varied from 5 through to 25



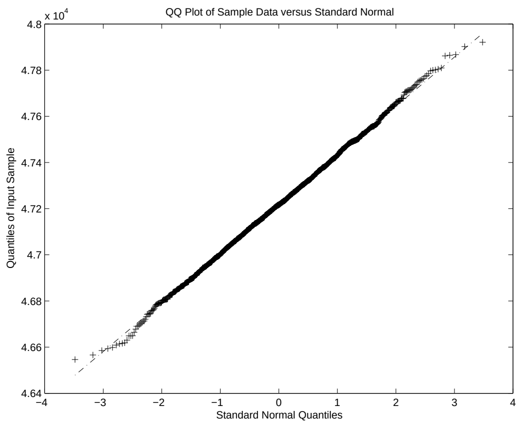
(a) 30 agent experiment



(b) 40 agent experiment



(c) 50 agent experiment



(d) 100 agent experiment

Figure 7.10: Figures (a)-(d) show qqplots of bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where agent numbers were varied from 25 through to 100.

Regarding the above figures, note that the distribution of the replicates is confirmed as from a normal distribution, this it to be expected, i.e. distribution of the *mean* of the samples is normal.

Confidence Intervals Table 7.3 presents example bootstrap confidence intervals for the 5 agent experiment. to determine the 95% confidence limits, we inspect the row entry for $\alpha = 0.975$ and for $\alpha = 0.025$ giving upper and lower limits $[2349.54, 2399.037]$ *ms* for this detection delays metric in this setup.

Table 7.4 presents calculations for the rest of the experiments, as agent numbers are increased.

The Bootstrap										
Statistic				$\hat{\theta}$	$\widehat{bias}$	$\hat{se}$				
				2368.7	-0.1252	9.5650				
BCa Confidence Intervals										
α	**0.025**	**0.050**	**0.100**	**0.160**	**0.840**	**0.900**	**0.950**	**0.975**	**zo**	**ahat**
	2349.540	2353.661	2357.764	2361.228	2385.452	2389.277	2394.028	2399.037	0.030084	0.011023
Bootstrap-t Confidence Intervals										
	2372.321	2378.683	2364.194	2366.829	2389.747	2395.564	2399.318	2406.109	N/A	N/A

Table 7.3: Results of the Bootstrap : showing parameter and confidence interval estimates and errors for the 5 agent partial protocol experiment

The Bootstrap									
	Agent Experiments								
Statistic	5	10	15	20	25	30	40	50	100
$\hat{\theta}$	2368.7	3276.6	4.4387	5494.0	6.6525	8.7102	2323.5	25881	50245
$\hat{bias}$	-0.1252	-0.0551	0.0547	-0.1657	14.9021	-0.0910	3.5600	0.8322	-1.3729
$\hat{se}$	9.5650	8.0808	7.472	8.5910	0.0182	29.2004	49.4293	51.9828	164.0740
α	BCa Confidence Intervals								
0.025	2351.298	3254.18	4413.420	5472.743	6607.26	8602.914	23120.31	25775.31	49738.4
0.05	2355.748	3257.527	4416.714	5475.863	6611.46	8613.277	23143.15	25796.98	49785.46
0.1	2359.919	3260.786	4420.286	5479.788	6617.996	8625.625	23163.14	25817.84	49866.36
0.16	2363.233	3263.537	4423.535	5482.699	6623.039	8635.785	23180.06	25835.93	49922.65
0.84	2387.173	3282.660	4444.921	5505.597	6660.774	8709.68	23305.07	25961.35	50335.21
0.9	2390.904	3285.649	4447.842	5508.922	6665.891	8720.07	23322.24	25978.27	50401.8
0.95	2395.835	3289.220	4452.419	5513.014	6672.447	8734.115	23343.92	26004.42	50490.48
0.975	2399.491	3292.459	4455.826	5516.398	6678.548	8747.263	23364.43	26023.57	50563.98
z0	-0.008773312	-0.01128007	0	-0.042625	0.023815	-0.056429	-0.048898	-0.0501535	0.04011681
ahat	0.010340	0.003990	0.002510	0.0005407	0.007535	0.004789	0.0005907	0.001648	0.0079676
α	Bootstrap-t Confidence Intervals								
0.25	2355.215	3271.297	4438.064	5486.136	6630.281	8695.599	23192.12	25808.98	50215.2

0.5	2362.165	3277.688	4445.646	5493.074	6645.079	8722.06	23239.52	25853.75	50339.33
0.1	2348.396	3265.342	4430.27	5478.616	6619.187	8666.96	23161.44	25773.42	50083.86
0.16	2351.85	3267.964	4434.654	5483.248	6624.685	8681.778	23177.58	25788.92	50172.58
0.84	2373.611	3287.029	4455.322	5505.508	6667.146	8752.567	23296.21	25902.67	50562.23
0.9	2376.067	3290.984	4458.773	5507.355	6672.376	8761.062	23313.79	25925.31	50608.76
0.95	2380.804	3294.424	4463.995	5513.72	6678.142	8781.134	23329.67	25960.83	50685.08
0.975	2385.228	3296.151	4467.794	5516.016	6685.014	8796.38	23350.32	25988.21	50756.23

Table 7.4: Bootstrap : showing parameter and confidence interval estimates and errors for agents monitored locally at given nodes

7.4 Summary

This chapter has presented detailed results and exploratory data analysis for the partial protocol experiments discussed in chapter 6.

The analysis verifies that the detection delays data are not normally distributed as was observed from the *4-plot* and confirmed by normality test results in Table 7.2. Therefore the non parametric bootstrap method was used for parameter estimation and calculation of confidence intervals. That is, normality is important to us because we wish to derive robust confidence intervals for the detection delays statistic.

The bootstrap BCa confidence intervals in Table 7.4 show that the detection delays (at 95% confidence) range from $[2351.298, 2399.491]$ *ms* for 5 agent experiments to $[49738.4, 50490.48]$ *ms* for 100 agent experiments.

Also examining the results in view of scalability there is concern with detection delays for large number of agents, e.g. the 100 agents experiment with delays approaching close to a minute in the worst case. This may suggest an upper limit in the number of agent hosted, for an agent platform like the one used for these experiments. Clearly maybe a concern in applications where there are strict time constraints for resources used by terminated agents to be reclaimed, but less so in those where detection just has to eventually succeed. In distributed systems , a standard approach to improving the scalability is to consider distribution of the service, this is explored in chapter 10 from page 187.

Chapter 8 next however considers the *full protocol* experiments as introduced in the experimental design. The chapter considers a similar data analysis procedure for the datasets as has been done here and also the discussion is similar to that given here

CHAPTER 8

Full Protocol Results

8.1 Introduction

Following on from the previous chapter, this chapter presents briefly the corresponding results for the full protocol experimental setup as described in the experimental design, section 6.5, page 120. Recall that these were a set of controlled experiments where an increasing number of agents were hosted and monitored on a local controller. Agents in this setup register and submit full information about the public protocol they are executing and are then monitored.

The next chapter, **Chapter** 9 provides an exploratory comparative analysis of the datasets of this setup and the partial protocol discussed in the previous chapter.

8.2 Results: Exploratory Data Analysis

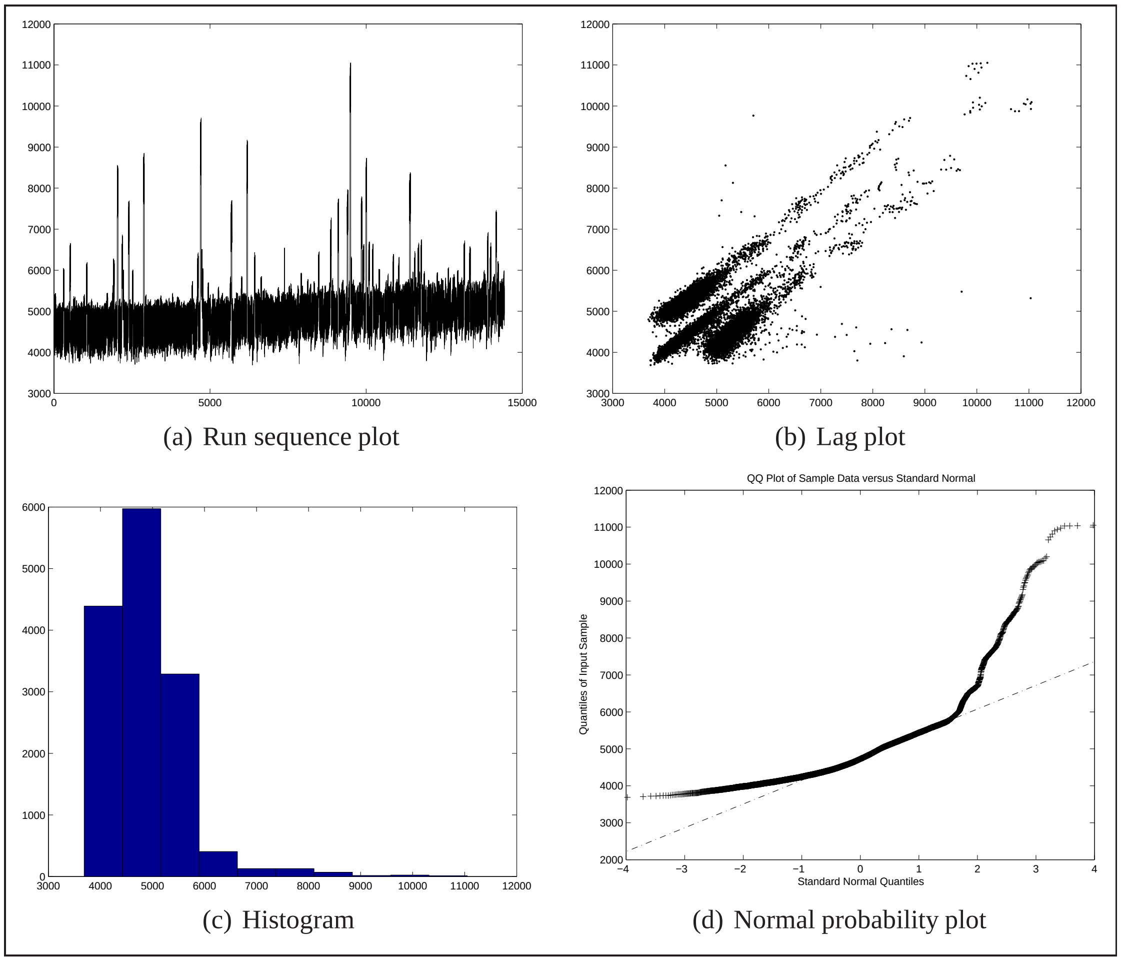
```
                         SUMMARY STATISTICS
                       NUMBER OF OBSERVATIONS =     5317
***********************************************************************
*       LOCATION MEASURES         *      DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.3564000E+04  *  RANGE        =   0.2470000E+04  *
*  MEAN         =   0.2921928E+04  *  STAND. DEV.  =   0.4257777E+03  *
*  MIDMEAN      =   0.2882001E+04  *  AV. AB. DEV. =   0.3228591E+03  *
*  MEDIAN       =   0.2795000E+04  *  MINIMUM      =   0.2329000E+04  *
*               =                  *  LOWER QUART. =   0.2597000E+04  *
*               =                  *  LOWER HINGE  =   0.2597000E+04  *
*               =                  *  UPPER HINGE  =   0.3158000E+04  *
*               =                  *  UPPER QUART. =   0.3158000E+04  *
*               =                  *  MAXIMUM      =   0.4799000E+04  *
***********************************************************************
*      RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =   0.4246227E+00  *  ST. 3RD MOM. =   0.1471157E+01  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.5648685E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.3380893E+03  *
*               =                  *  UNIFORM PPCC =   0.9070824E+00  *
*               =                  *  NORMAL  PPCC =   0.9325916E+00  *
*               =                  *  TUK -.5 PPCC =   0.6920604E+00  *
*               =                  *  CAUCHY  PPCC =   0.2227653E+00  *
***********************************************************************
```

Figure 8.1: Showing detection delays summary statistics for the 5 agent experiment calculated using DATAPLOT

The analysis and data presentation is the same as the previous chapter where;

- Figure 8.1 shows summary statistics of the detection delay metric for an example experiment where 5 agents were hosted.
- Figures 8.2 to 8.4 show the 4-plots for all agent experiments for exploratory data analysis purposes.
- Table 8.1 shows the rest of the summary statistics for all experiments with agent numbers varied.
- Table E.1 in Appendix E, page 316 shows results of normality tests for these dataset. Inspection of the table shows that the data fail normality tests.

- Regarding the bootstrap, figures 8.5, to 8.7 show bootstrap replicates for the mean of detection delays.
- Figures 8.8 to 8.10 shows the corresponding qqplots for the bootstrap replicates.
- Tables 8.2 presents results for bootstrap parameter estimates and detailed *BCa* and *Bootstrap-t* confidence intervals.



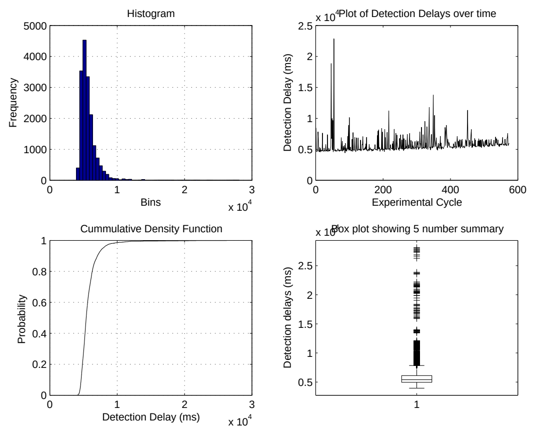
(a) Run sequence plot (b) Lag plot

(c) Histogram (d) Normal probability plot

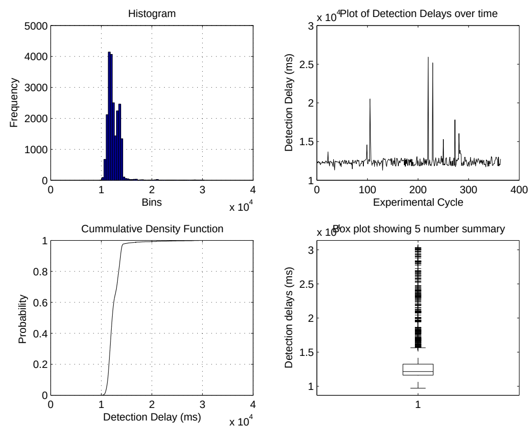
Figure 8.2: Figures a-d show elements of the 4-plots for the purposes of exploratory data analysis for centralised experiments using the full partial protocol scheme where 5 agent were hosted. All measurements are in ***milliseconds***

Again, as in the previous chapter, Table 8.1 presents summary statistics for all experiments in this setup. The table presents *location measures* to find a central value that describes the data, *dispersion measures* to capture the spread in the data, *randomness measures* and distributional measures. The third and fourth moments are the skewness and kurtosis of the distribution.

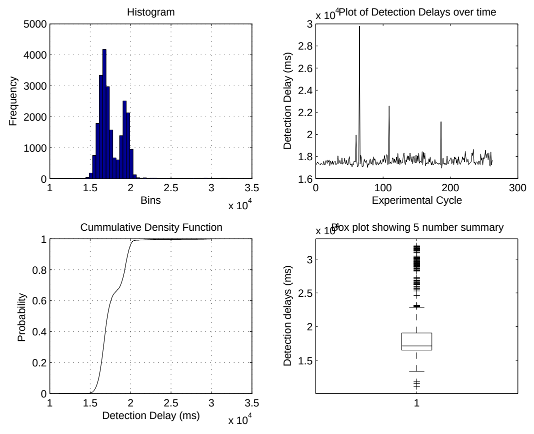
The table also complements 4-plot figures, i.e. figures 8.2 to 8.4 by exploring properties of the detection delays data by presenting some distributional measures. Again, the distributional measures strongly indicate that the detection delay data for the full protocol experiments are also not from a normal distribution.



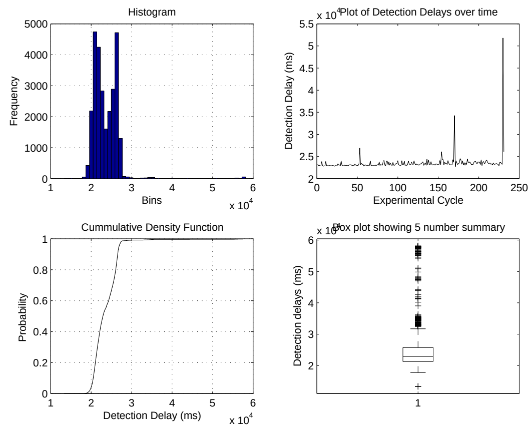
(a) 10 agent experiment



(b) 20 agent experiment

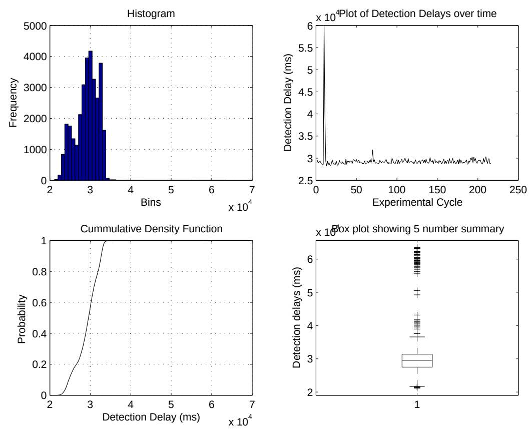


(c) 30 agent experiment

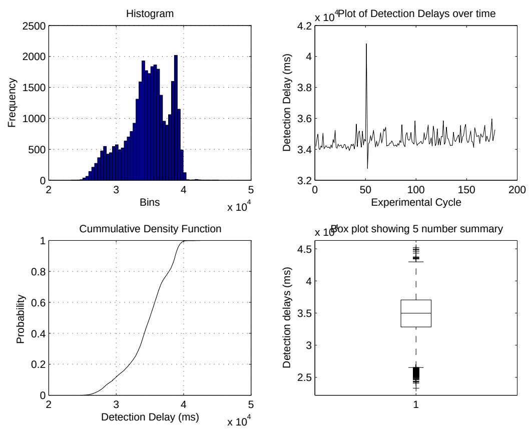


(d) 40 agent experiment

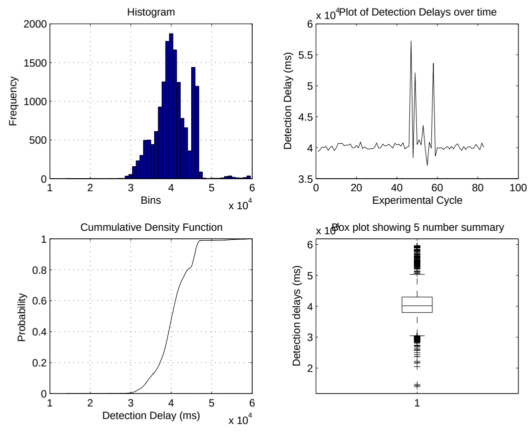
*Figure 8.3: Figures (a)-(d) show 4-plots for the purposes of exploratory data analysis for centralised experiments using the full protocol scheme where agent numbers were varied from 10 through to 25. All measurements are in **milliseconds***



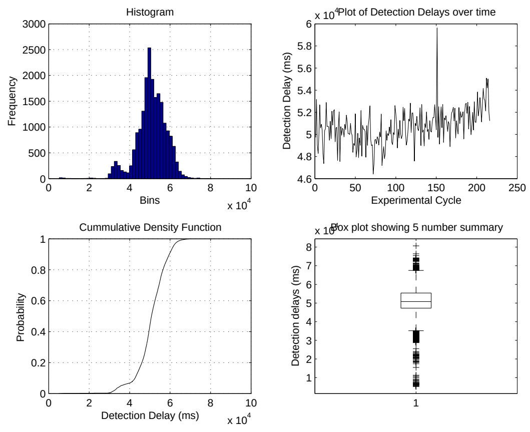
(a) 50 agent experiment



(b) 60 agent experiment



(c) 70 agent experiment



(d) 100 agent experiment

Figure 8.4: Figures (a)-(d) show 4-plots for the purposes of exploratory data analysis for centralised experiments using the full protocol scheme where agent numbers were varied from 50 through to 100. All measurements are in ***milliseconds***

SUMMARY STATISTICS

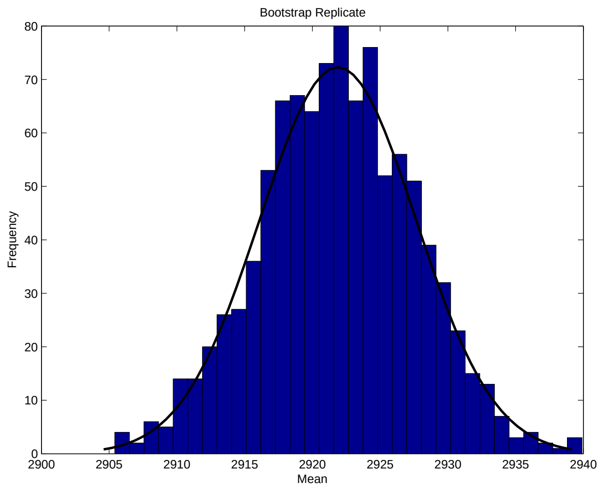
	AGENT EXPERIMENTS								
	5	10	20	30	40	50	60	70	100
	LOCATION MEASURES								
Midrange	0.3564	0.7022	0.1336	0.1715	0.2415	0.3152	0.3425	0.4061	0.4940
Mean	0.2921	0.5628	0.1237	0.1760	0.2338	0.2918	0.3462	0.4032	0.5080
Midmean	0.2882	0.5644	0.1238	0.1762	0.2341	0.2919	0.3464	0.4054	0.5052
Median	0.2795	0.5395	0.1214	0.1712	0.2287	0.2954	0.3497	0.4019	0.5079
DISPERSION MEASURES									
Range	0.2470	0.6155	0.7295	0.1207	0.2167	0.2075	0.2189	0.3793	0.6254
Stand. Dev	0.4257	0.9632	0.9852	0.1411	0.2435	0.2817	0.3305	0.4198	0.7051
Av. Ab. Dev	0.3228	0.6999	0.8050	0.1167	0.2120	0.2275	0.2619	0.3201	0.5334
Minimum	0.2329	0.3945	0.9717	0.1111	0.1331	0.2114	0.2330	0.2164	0.1813
Lower Quart	0.2597	0.4950	0.1161	0.1651	0.2128	0.2748	0.3284	0.3798	0.4730
Lower Hinge	0.2597	0.4950	0.1161	0.1651	0.2128	0.2748	0.3284	0.3798	0.4730
Upper Hinge	0.3158	0.6074	0.1321	0.1904	0.2571	0.3137	0.3705	0.4299	0.5537
Upper Quart	0.3158	0.6074	0.1321	0.1904	0.2571	0.3137	0.3705	0.4299	0.5537
Maximum	0.4799	0.1010	0.1701	0.2319	0.3498	0.4190	0.4520	0.5958	0.8067

RANDOMNESS MEASURES									
Autoco coef	0.4246	0.7945	-0.3532	-0.2015	-0.1445	0.1621	0.2307	0.2472	0.3237
	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
DISTRIBUTIONAL MEASURES									
St. 3rd Mom	0.1471	0.14051	0.6062	0.4725	0.3875	-0.4408	-0.5565	0.2060	-0.5456
St. 4th Mom	0.564	0.5350	0.3205	0.2211	0.2782	0.2420	0.2742	0.4255	0.3907
St Wilk-Sha	-0.3380	-0.5873	-0.3354	-0.4984	-0.4959	-0.3858	-0.3651	-0.1803	-0.2048
Uniform ppcc	0.9071	0.9122	0.9707	0.9714	0.9786	0.9809	0.9734	0.9556	0.9512
Normal ppcc	0.9326	0.9451	0.9767	0.9635	0.9678	0.9799	0.9817	0.9869	0.9875
Tuk -.5 ppcc	0.6921	0.6660	0.6588	0.6231	0.6320	0.6121	0.6199	0.7210	0.7004
Cauchy ppcc	0.2227	0.1550	0.1502	0.1427	0.1418	0.1181	0.1137	0.1933	0.1686

Table 8.1: Summary statistics for all agents experiments in the centralised setup for full protocol scheme, showing location, dispersion and distributional measures

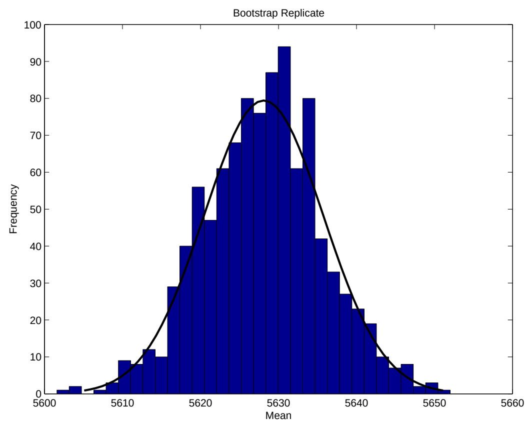
8.2.1 Bootstrap results

Replicates distributions The bootstrap was also carried out on the full protocol experiments datasets. Figure 8.5 shows the distribution of bootstrap replicates for the mean for an experiment in which the number of agents was set to 5. Figures 8.6 and 8.7 present the plots for the rest of the experiments with agent numbers varied from 10-100.

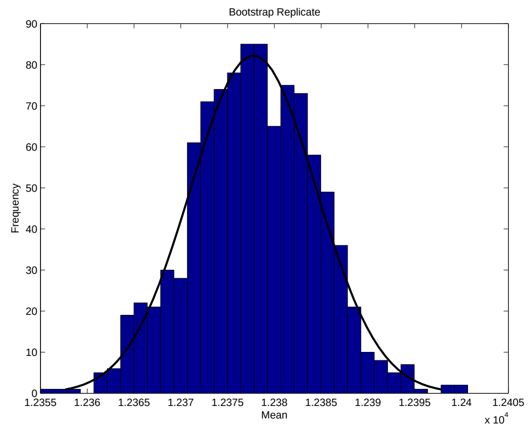


(a) 5 agent experiment

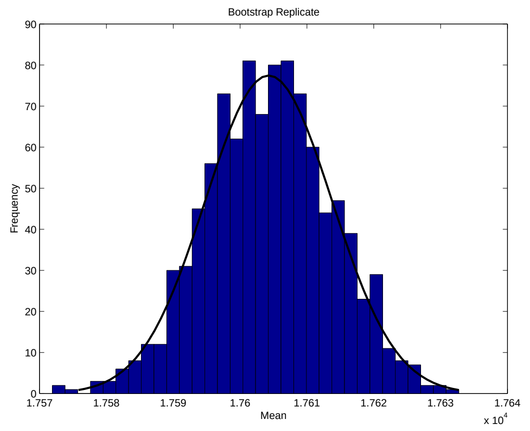
Figure 8.5: Figures shows bootstrap replicates of the mean for centralised experiments for the full protocol setup where 5 agent were hosted



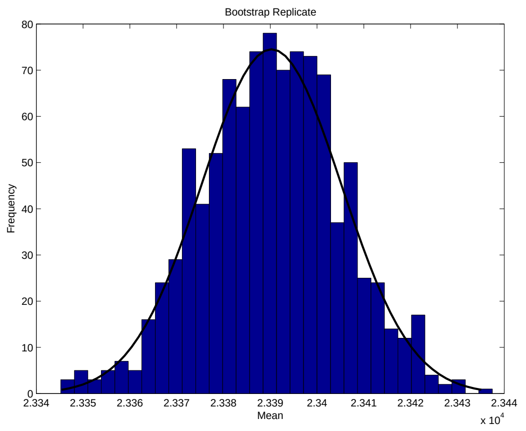
(a) 10 agent experiment



(b) 20 agent experiment

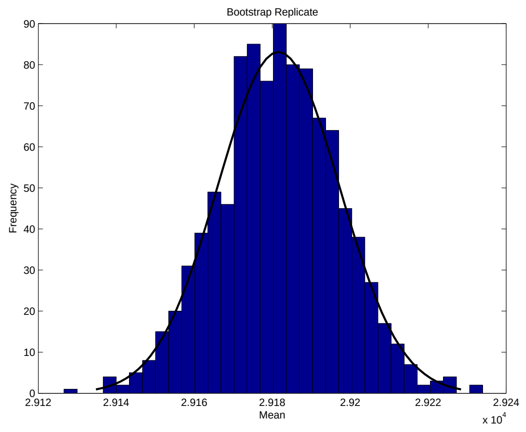


(c) 30 agent experiment

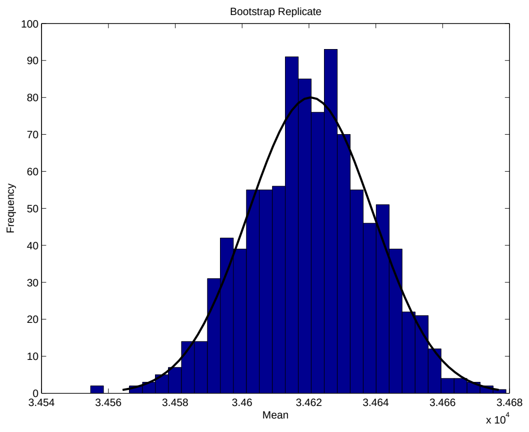


(d) 40 agent experiment

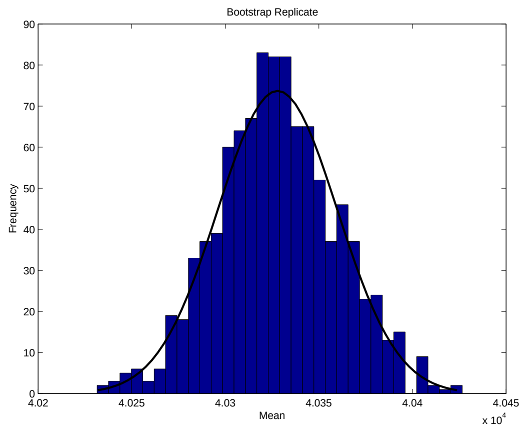
Figure 8.6: Figures (a)-(d) show bootstrap replicates of the mean for centralised experiments using the full protocol scheme where agent numbers were varied from 10 through to 40



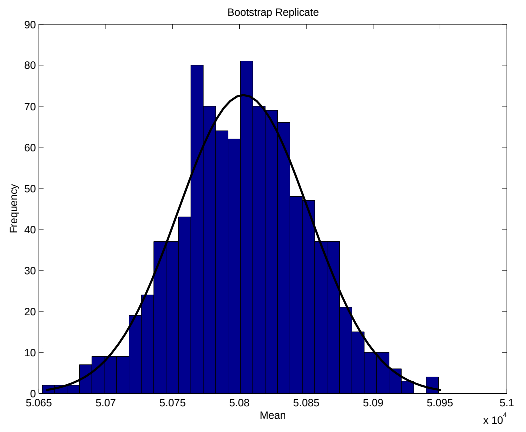
(a) 50 agent experiment



(b) 60 agent experiment

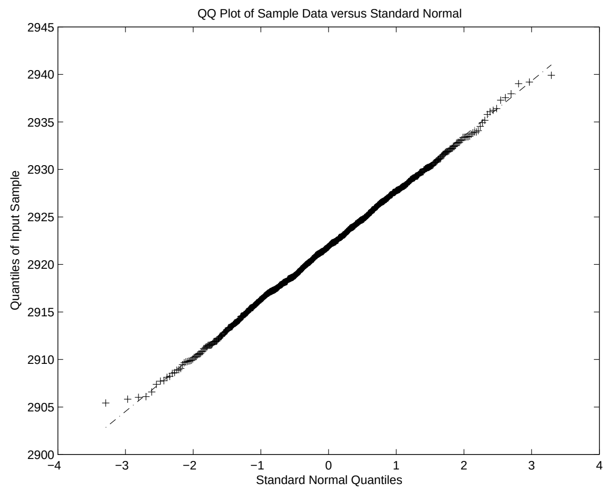


(c) 70 agent experiment



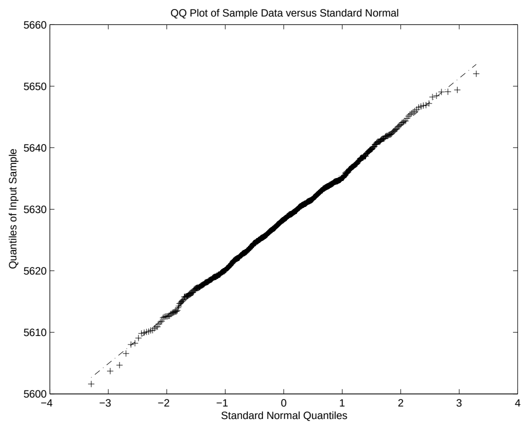
(d) 100 agent experiment

Figure 8.7: Figures (a)-(d) show bootstrap replicates of the mean for centralised experiments using the full protocol scheme where agent numbers were varied from 50 through to 100

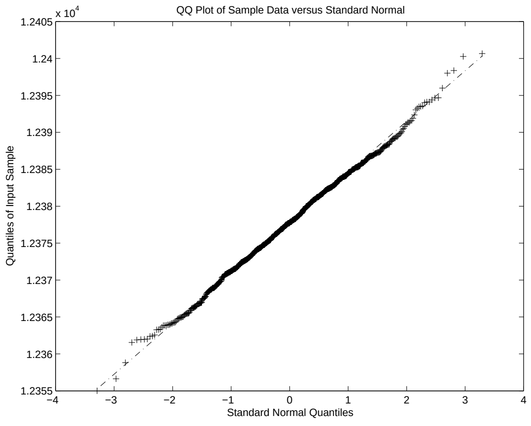


(a) 5 agent experiment

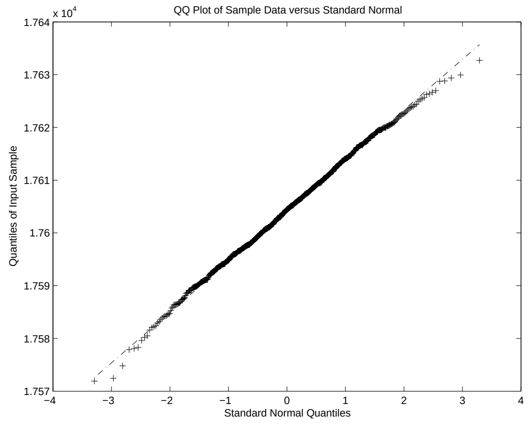
Figure 8.8: Figures shows QQ plot of the bootstrap replicates of the mean for centralised experiments using the full protocol scheme where 5 agent were hosted



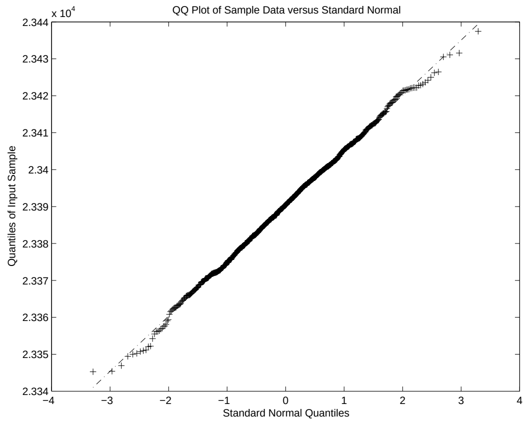
(a) 10 agent experiment



(b) 20 agent experiment

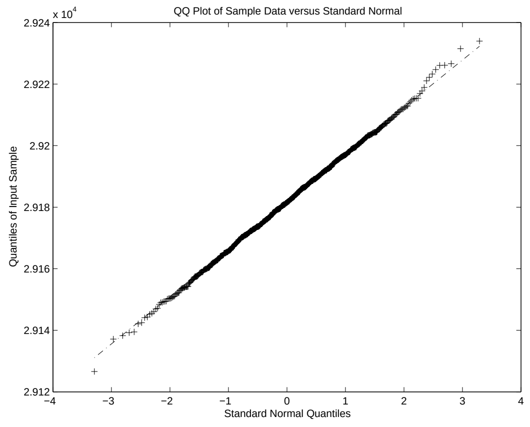


(c) 30 agent experiment

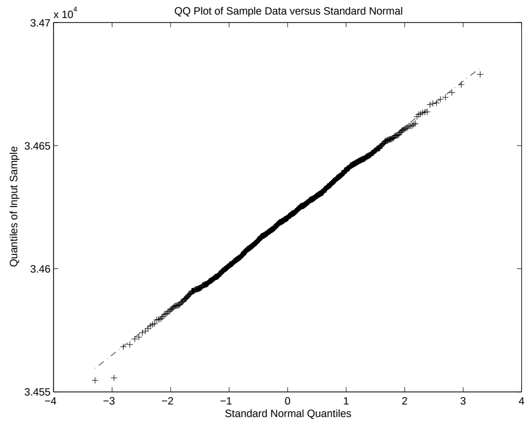


(d) 40 agent experiment

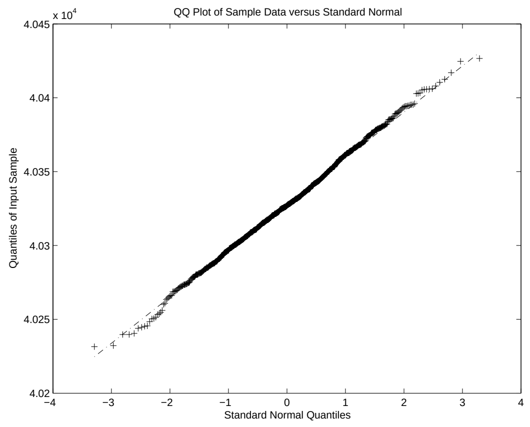
Figure 8.9: Figures (a)-(d) show qqplots of bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where agent numbers were varied from 5 through to 25



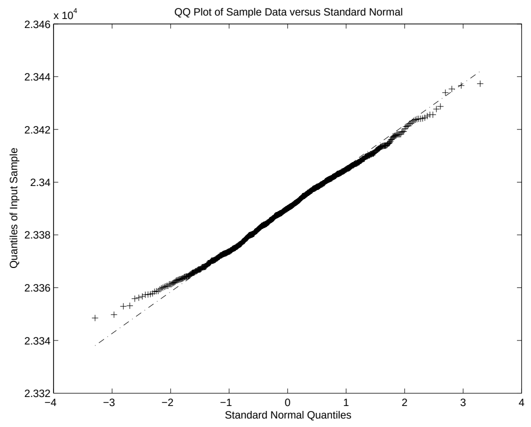
(a) 50 agent experiment



(b) 60 agent experiment



(c) 70 agent experiment



(d) 100 agent experiment

Figure 8.10: Figures (a)-(d) show qqplots of bootstrap replicates of the mean for centralised experiments using the partial protocol scheme where agent numbers were varied from 25 through to 100

THE BOOTSTRAP										
Statistic				$\hat{\theta}$	$\hat{bias}$	$\hat{se}$				
				2921.9	-0.1355	5.7427				
BCA CONFIDENCE INTERVALS										
α	**0.025**	**0.050**	**0.100**	**0.160**	**0.840**	**0.900**	**0.950**	**0.975**	**zo**	**ahat**
	2911.449	2913.017	2915.459	2917.579	2932.65	2934.920	2937.318	2939.172	0.005013	0.004118
BOOTSTRAP-T CONFIDENCE INTERVALS										
	2910.406	2915.466	2906.191	2907.716	2922.859	2924.821	2926.756	2928.058	N/A	N/A

Table 8.2: Results of the Bootstrap: showing parameter and confidence interval estimates and errors for the 5 agent partial protocol experiment

THE BOOTSTRAP										
		AGENT EXPERIMENTS								
Statistic		5	10	20	30	40	50	60	70	100
$\hat{\theta}$		2921.9	5628.1	12378	17604	23389	29182	34620	40328	50802
$\hat{bias}$		-0.1355	0.0675	-0.0539	-0.1703	0.2829	0.2786	-0.0249	0.2946	1.108
$\hat{se}$		5.7427	7.1345	6.7262	9.3927	15.1439	15.1500	18.8177	33.8798	49.22
BCA CONFIDENCE INTERVALS										
	0.025	2906.491	5610.928	12358.17	17578.95	23365.79	29160.62	34573.32	40198.16	50731.81
	0.05	2909.073	5613.733	12360.10	17583.12	23371.86	29166.20	34579.56	40207.29	50752.74
	0.1	2911.269	5617.496	12363.35	17587.13	23378.35	29173.00	34587.53	40223.12	50775.32
	0.16	2913.235	5620.099	12365.75	17590.40	23384.30	29178.39	34593.61	40235.18	50794.06
α	**0.84**	2927.572	5638.16	12382.56	17613.48	23421.14	29216.92	34639.48	40315.38	50915.34
	0.9	2929.336	5640.67	12385.04	17616.66	23426.1	29223.20	34646.05	40326.9	50934.47
	0.95	2932.1	5643.975	12387.93	17621.1	23432.39	29229.65	34652.57	40342.91	50955.93
	0.975	2934.693	5646.726	12390.66	17624.62	23438.29	29237.13	34659.53	40355.27	50975.58
	z0	0.01880082	-0.01629380	0.007519956	0.03008408	-0.015040	0.010026	-0.017547	-0.021307	0.002506
	ahat	0.004241	0.002223	0.0008736	0.000656	0.000483	-0.000515	-0.000646	0.000314	-0.000776
BOOTSTRAP-T CONFIDENCE INTERVALS										
	0.25	2911.418	5620.909	12374.19	17594.53	23393.67	29162.04	34618.29	40305.35	50828.32

	0.5	2916.186	5626.214	12379.72	17600.44	23404.59	29176.78	34634.62	40327.38	50870.38
	0.1	2907.302	5615.322	12368.59	17587.30	23379.13	29153.53	34606.42	40282.18	50789.62
	0.16	2909.011	5617.324	12371.72	17590.51	23389.45	29156.82	34611.70	40287.34	50811.06
α	**0.84**	2923.899	5635.363	12389.17	17613.29	23424.81	29193.99	34658.33	40365.11	50931.26
	0.9	2926.192	5637.218	12392.49	17616.56	23427.65	29199.36	34661.78	40380.38	50951.45
	0.95	2928.504	5641.817	12395.95	17619.62	23434.68	29205.86	34670.39	40391.95	50967.11
	0.975	2931.67	5644.654	12396.84	17622.21	23444.42	29214.42	34682.38	40396.86	50990.37

Table 8.3: Bootstrap results: showing parameter and confidence interval estimates and errors for centralised experiments in the full protocol scheme agents numbers ranging from 5 to 100.

8.3 Summary and observations

This chapter has presented detailed results and exploratory data analysis for the full protocol scheme. As in the previous chapter, detection delays data are not normally distributed as shown by all tests and exploratory data analysis diagrams. Therefore as in the previous chapter, the non parametric bootstrap method was used for parameter estimation and calculation of confidence intervals.

The bootstrap BCa confidence intervals in Table 8.3 in page 173 shows that the detection delays (at 95% confidence) range from $[2911.449, 2939.172]$ *ms* for 5 agent experiments to
$[50731.81, 50975.58]$ *ms* for 100 agent experiments.

Regarding scalability, there are similar concerns as those expressed in the previous chapter 7, page 156, i.e. for large agent numbers, the delays as delays approach a minute.

Following on from this discussion, the next chapter, **Chapter** 9 provides a detailed exploratory comparative analysis of the datasets of this setup, the *full protocol* and the *partial protocol* discussed in the previous chapter.

CHAPTER 9

Comparisons

9.1 Introduction

The previous two chapters, **Chapter** 7 and **Chapter** 8 presented and analysed datasets for the partial and full protocol schemes. This chapter aims to provide a exploratory comparative analysis of those two datasets. The chapter also provides results of the nonparametric hypothesis tests on the location parameter of the two datasets, e.g. mean detection delays. Non-parametric tests were chosen again because the normality and homogeneity of variances assumptions do not hold and cannot be justified fully for these datasets.

This chapter also presents results of the scalability experiments, showing how both the partial and full protocol schemes scale with increasing number of agents monitored.

In addition, regarding repeatability of experiments, results of the non parametric analysis of variance (ANOVA) and multiple comparison tests of experimental runs for a given experiment are presented in Appendix E, page 318.

A note on datasets and data analysis and figures The datasets analysed here are those considers in chapter 7 and chapter 8 and in addition, for comparisons we will consider datasets for data collection criteria 2 and 3 below. Recall the data collection criteria introduced in chapter 6, page 175 for experimental cycles data collection, i.e.

1. *Individual* detection delays recorded for repeated execution of the protocol over many cycles.

2. For every experimental cycle period, detection delays from the repeated execution of the protocol for that cycle recorded and some statistic calculated, i.e. treat each cycle as an experiment.

3. *Accumulated* detection delays recorded over the entire experiment for the repeated execution of the protocol and a statistic calculated over each period and the process repeated without discarding previous measurements but accumulating.

And recall that an illustration for these scenarios was given in Appendix C, figures C.2 (a) to (c) respectively in page 283.

We can use these to evaluate differences across experimental cycles and runs and investigate variations and repeatability of these experiments.

Regarding the presentation, figures and tables, consider

1. Figures showing time series plots for the above datasets for selected experiments e.g. Figure 9.1 shows plots for the 100 agent experiment with the error bars derived from the above confidence intervals for the 100 agent experiment. Note that in Figure 9.1, the scale is 10^4 milliseconds therefore the values are consistent with the confidence intervals calculated in the previous chapters for *partial protocol* and *full protocol* schemes.

2. Corresponding tables e.g. Table 9.1 presenting 95% non parametric confidence intervals for the cyclic and accumulated datasets. Inspecting these tables indicates little differences in the cyclic datasets and the cumulative datasets in the confidence intervals.

BOOTSTRAP CONFIDENCE INTERVALS							
AGENT EXPERIMENTS							
	10	20	30	40	50	70	100
α	FULL PROTOCOL CONFIDENCE INTERVALS						
0.025	3179.829	5449.963	8546.278	22914.46	25994.38	49484.24	41173.35
0.975	3372.211	5635.5	10048.27	24162.62	26696.33	50061.83	46026.54
α	PARTIAL PROTOCOL CONFIDENCE INTERVALS						
0.025	3185.246	5406.278	8274.083	22799.43	25731.95	49456.42	39468.48
0.975	3365.526	5583.806	9813.204	24025.67	26442.18	50105.94	44283.94

Table 9.1: BCa Confidence intervals for the cyclic datasets

BOOTSTRAP CONFIDENCE INTERVALS

AGENT EXPERIMENTS							
	10	20	30	40	50	70	100
0.025	4940.808	12409.70	34326.23	23173.17	29220.01	40257.03	63309.14
0.975	4998.555	12429.27	35431.32	23211.35	29268.05	40326.01	63489.35
α	PARTIAL PROTOCOL CONFIDENCE INTERVALS						
0.025	3252.303	5376.824	8093.37	22855.9	25760.06	49528.07	44041.75
0.975	3293.456	5413.694	8302.37	23030.87	25949.57	49622.07	44318.33

Table 9.2: BCa Confidence intervals for cumulative data sets

9.2 Hypothesis testing

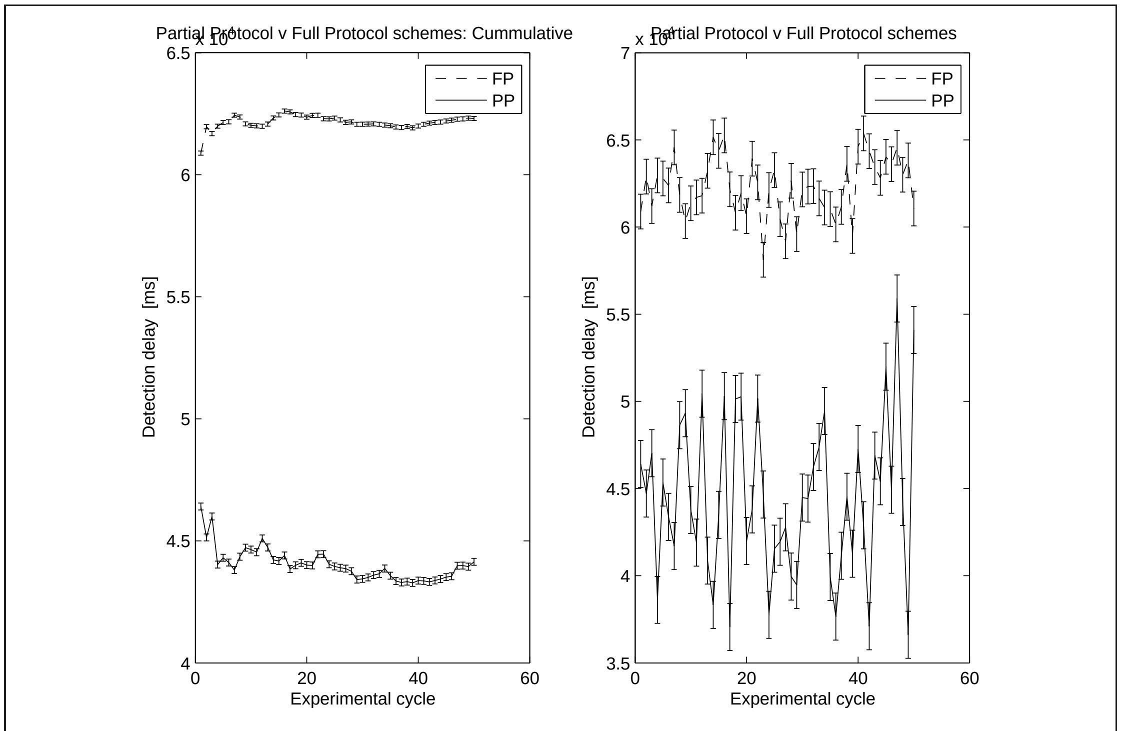
Though the work here was exploratory, we can compare, test hypothesis and make statements about the partial and full protocol data sets. In particular statements about the location parameters e.g. means or the medians of detection delays in the data sets.

Recall that for doing comparisons and hypothesis testing[1] on independent samples for example, testing if the samples represent underlying populations with different mean values (but assuming equal variances and normal distribution), a standard approach is to use the t-test [207][2]

But given the discussion earlier in chapters 7 and chapter 8 about the distribution of

[1]Hypothesis testing is a mechanism for determining if an assertion about a characteristic of a population is reasonable.

[2]The most common is the two sample t-test that tests whether or not two independent populations have different mean values on some prescribed measure. The t-test uses a t-test statistic to determine a p-value that indicates how likely we could have gotten these results by chance. By convention, if there is a less than 5% chance (for 95%confidence) of getting the observed differences by chance, we reject the null hypothesis and say we found a statistically significant difference between the two groups.



(a) 100 agent experiment

Figure 9.1: Showing superimposed time series plots to compare partial protocol and full protocol schemes for experiments with agent numbers set to 100. Showing error bars with width computed from non parametric confidence intervals. NB, on the y-axis, the scale is 10^4 ***milliseconds*** *so the values are consistent with the bootstrap confidence intervals calculated in previous chapters for* partial protocol *and* full protocol *schemes respectively.*

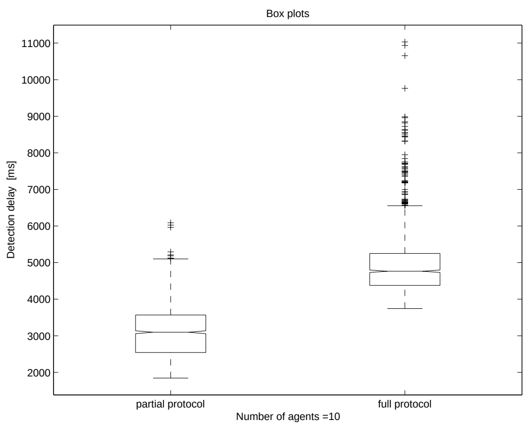
detection delays not being normally (gaussian) distributed, and failed test for equality of variances, in order to do hypothesis tests regarding the partial and full protocol schemes, we explored the use of non-parametric tests which do not make distributional assumptions.

Non-parametric statistical methods and non-parametric statistical inference are discussed in [110, 95] and [207]. For example the Kruskal-Wallis test originally discussed in [138] is a non-parametric test for equality of the location parameter (e.g. median) among datasets or groups (i.e. $N \geq 2$) , I used this test for example in analysing repeated experimental runs and to make comparisons across the runs to test repeatability.

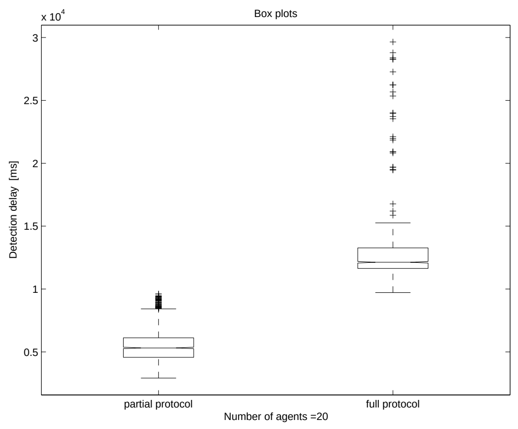
If only two datasets are considered and we are interested in whether the two samples come from the same distribution then the Mann-Whitney U-test [155] is the nonparametric alternative to the two-sample t-test.

Regarding implementation of these tests, the R Statistical environment [225] provides implementation of the Kruskal-Wallis test procedure and Mann-Whitney test.

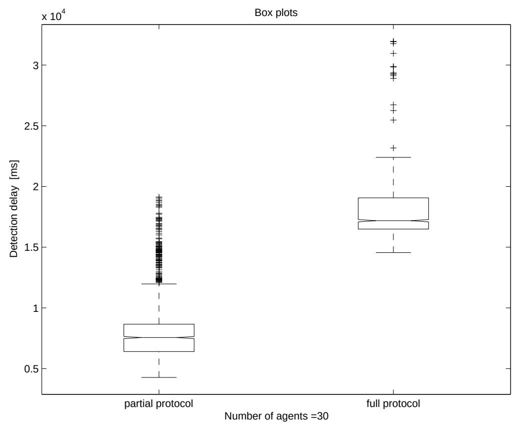
We can do hypothesis testing on the partial and full protocol datasets on the differences of the location parameters, e.g. mean, median of detection delays, see Table E.2 in Appendix E , page 317. Alternatively we can visually inspect the box plots as they also give a nonparametric mechanism for comparing populations. These are shown in Figure 9.2.



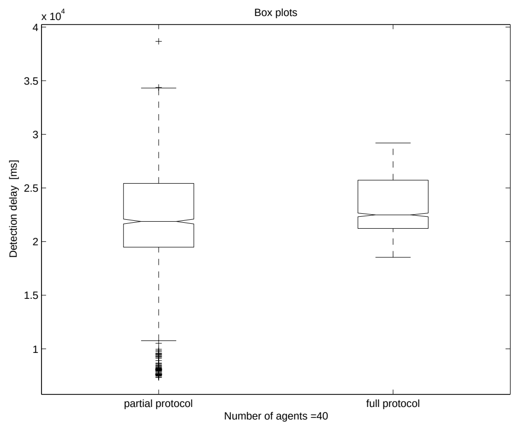
(a) 10 agent experiments



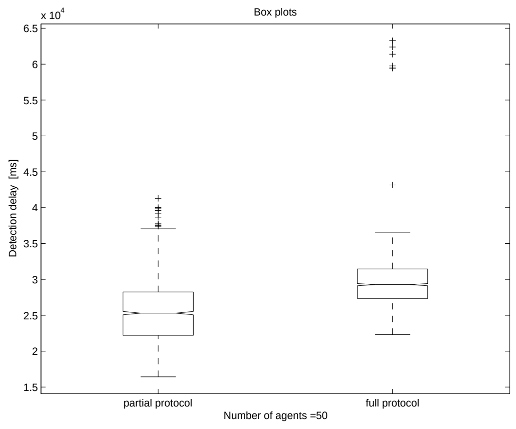
(b) 20 agent experiments



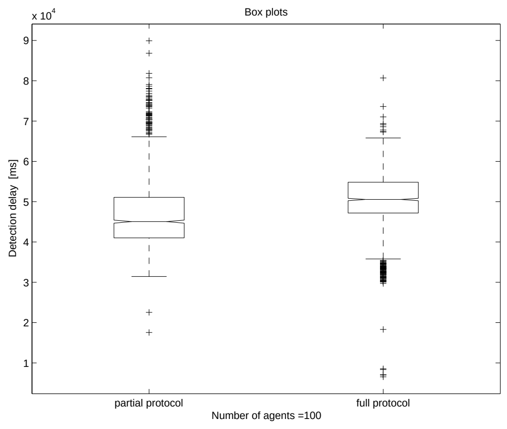
(c) 30 agent experiments



(d) 40 agent experiments



(e) 50 agent experiments



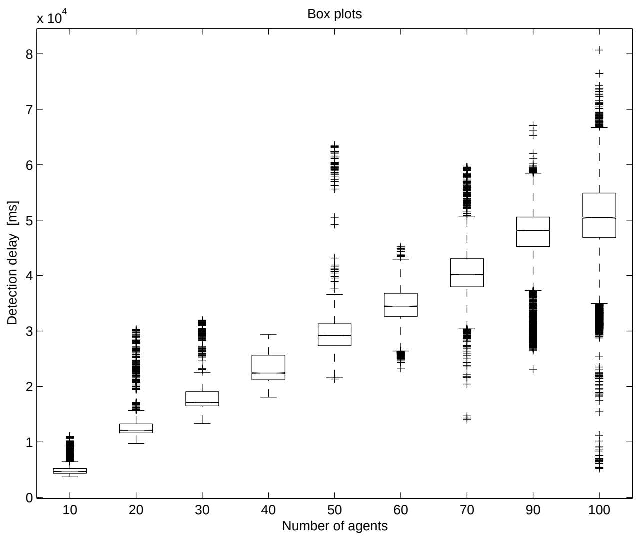
(f) 100 agent experiments

Figure 9.2: Figure shows box plots for partial and full protocol experiments for various agents experiments. This accompanies the non parametric hypothesis tests about the location parameter.

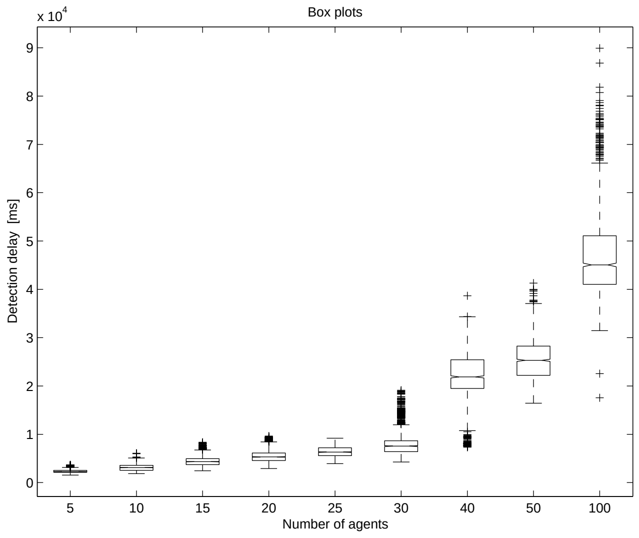
9.3 Scalability Results

To explore how both setups scaled, consider Figure 9.3 showing the boxplot generated by the kruskal-wallis test[3] on datasets for all agent experiments with numbers varied from 10 through to a 100. Also consider Figure 9.4 showing an equivalent line plot of the mean detection delays for each experiment against agent numbers,i.e. showing how detection delays vary with the number of agents monitored in both the partial and full protocol schemes. This figure also shows error bars around values as an indicator error margins derived from confidence intervals.

[3]Implemented by *multiple compare* in Matlab.

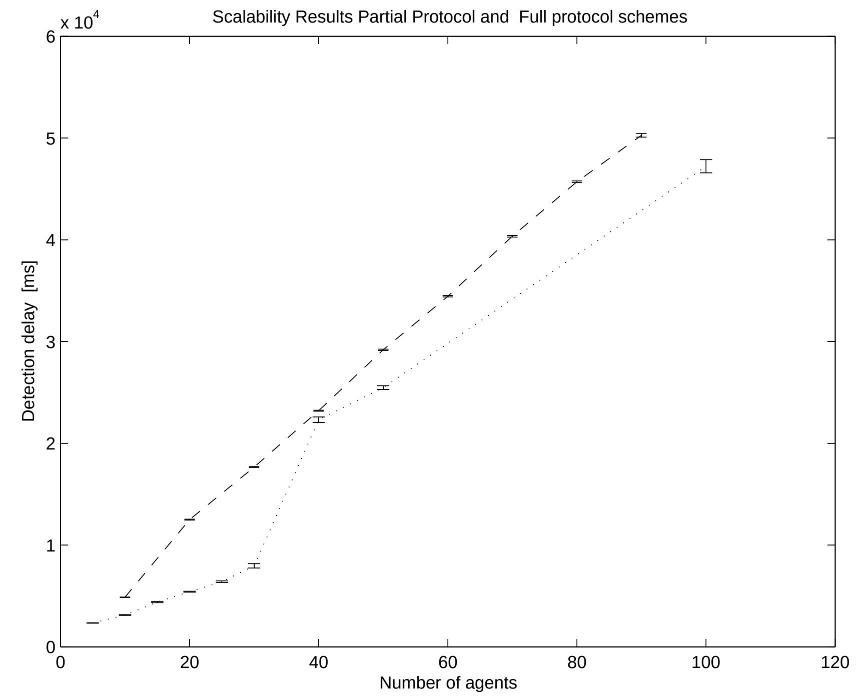


(a) Full protocol experiments



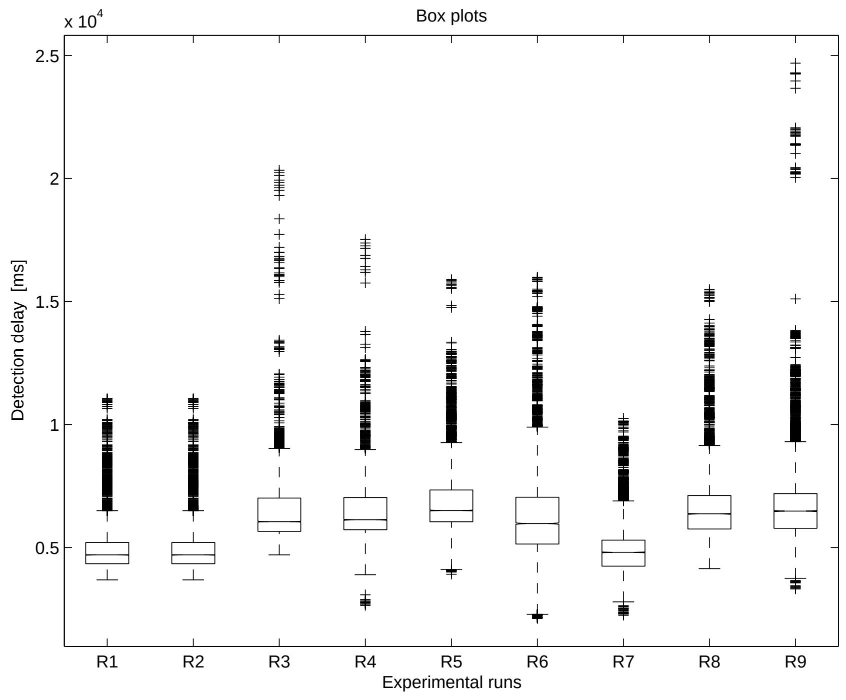
(b) 10 agent experiment

Figure 9.3: Box plots for full protocol and partial protocol schemes for all agent experiments. Figure generated as part of the non parametric analysis of variance Kruskal-Wallis procedure

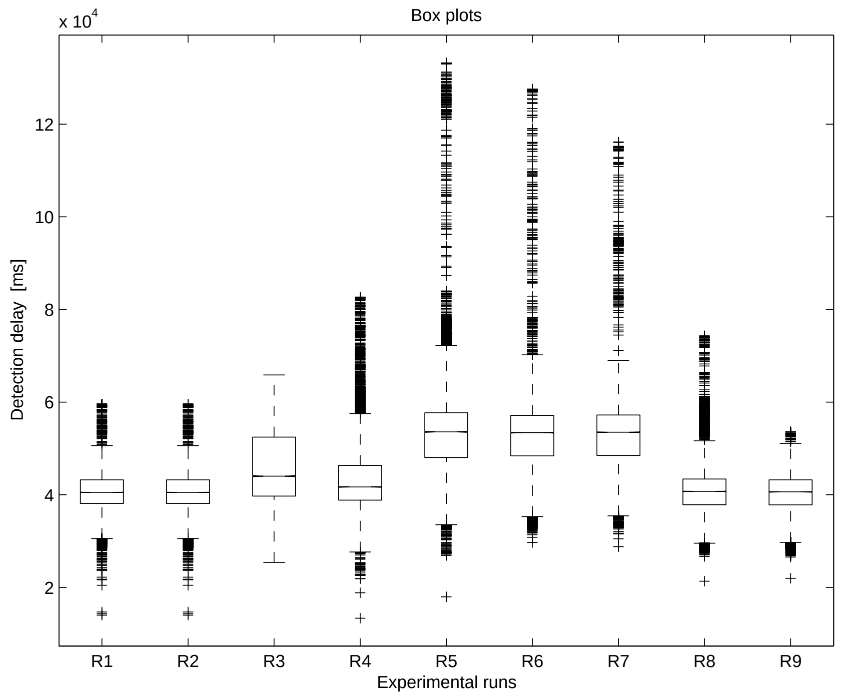


(a) Full protocol experiments

Figure 9.4: Plots for full protocol and partial protocol schemes for all agent experiments.



(a) 10 experimental runs



(b) 70 agent experimental runs

Figure 9.5: Showing box plots for 9 experimental run for the 10 and 70 agent full protocol experiment

9.4 Summary

This chapter has presented some exploratory comparative exploration of the partial and full protocol datasets. The data in both setups in non normal and variances are not homogeneous. The non parametric one sided two sample Wilcoxon (Mann-Whitney) tests, Table (E.2 page 318 in Appendix E) for all experiments provide strong evidence against (as given by extremely low p-values) the hypothesis that the location parameters are equal and therefore very likelihood that the location parameter for the partial protocol dataset is lower than that of the full protocol datasets. Inspecting other figures, e.g. side by side Boxplots also provide supporting visual evidence that even for a small sized protocols as one used in the simulation, using the partial protocol scheme should yield better results

On repeatability of experiments, it is worth reporting that the analysis of variance tests do show some notable variation between experimental runs. The scalability experiments also demonstrate that across all experiments where agent numbers were varied from low to high, the partial protocol scheme records low values of detection delays.

This results are fairly significant given that the depth and size of the protocol used in this experiments was small. Analytically, the differences will be even more pronounced for the large protocol graphs, the variance maybe explained by the underlying agent middleware and scheduling of agents behaviours in a framework.

CHAPTER 10

Distribution

10.1 Introduction

This chapter presents experimental results for the distributed setting as described in section 6.5, page 122. The chapter starts by providing an overview of the architecture and the experimental setup as discussed there.

Experiments in the setup necessarily have to consider network delays, to get a distribution profile of these delays as they have a bearing on detection delays distributions. Regarding the well known problem of the absence of global clocks in distributed systems, the approach we followed was to choose a practical synchronisation mechanism and synchronise hosts using network time protocol, NTP, a distributed time protocol available as a network service on an operating system.

Therefore in Appendix H we present an experiment for determining distribution of network delays and in this chapter summarise the results for this additional experiment.

Experimental Architecture The architecture for distribution was discussed in the experimental design, section 6.5, page 123 and shown in Figure 6.4 there, where at the core are a number of peer to peer interconnected nodes that act as redundant controllers

[1]. For practical considerations, each controller executes a simple *load balancer*[2]. Each node maintains a *profile* structure which provides an up-to-date data on number of locally registered. At the outer second tier of that figure are *client* nodes[3].

Dynamics On *initialisation*, the controllers in the cluster execute a simple *controller cluster registration protocol* to register with each other, in the process making available their profile information. At *runtime*, controllers routinely update their remotely cached profiles if local conditions change, consider for example, when local load exceeds the declared threshold (e.g. if the limit of registered agents is reached) or if a controller becomes unavailable. Note that to scale, the cluster is easily extendable to include the second tier client nodes by allowing them to execute the *controller cluster registration protocol*.

Agents in the network participate in a *agent registration protocol* shown in Figure 10.1. Agents always attempt first to register with the immediate local controller [4] if one exist. A local controller may *forward* registration details of newly registering agents to suitable controller(s) in the cluster after consulting updated cached profiles of peer controllers.

Data and Data collection Each local controller monitors locally registered agents and records detection delays for these agents. Remotely registered agents are monitored by *cluster nodes* and the detection delays are recorded by these cluster nodes for each agent registered with them.

[1]Controller is a *role* assigned to an agent providing the protocol monitoring service in the cluster, we can use monitor and controller interchangeably.

[2]load here refers to the number of agents monitored, the threshold can be set or determined dynamically.

[3]Agents being monitored.

[4]Controller executing in the same machine.

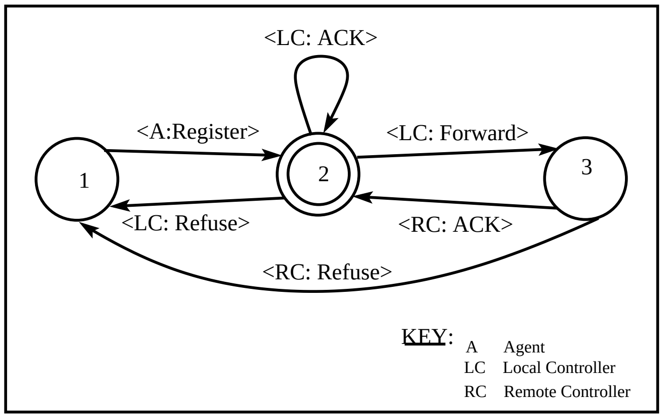


Figure 10.1: Registration protocol executed by client agents to register with monitoring controllers

Data collection setup and scenarios For the data collection setup and scenarios, recall and consider the discussion in the experimental design, section 6.6, pages 125 to 126.

Regarding the hardware configuration for these distributed setup experiments, consider Table C.1 in appendix C.3, page 286.

The scheduled experiment was for the following configuration. 6 host machines in a local area network, each hosting one $local$ controller, $L_i \in \mathcal{C}_L$ and 20 agents ($|\mathcal{A}_{Li}| = 20$), with 15 locally registered, (therefore $|\mathcal{A}_{Ri}| = 5$) and a cluster $\mathcal{C}$ of 3 *remote* controllers, ($|\mathcal{C}| = 3$). Each cluster controller hosted 20 agents, $|\mathcal{A}_{Ci}| = 20$, therefore giving a total number of agents, $|\mathcal{A}| = 180$. This is summarised in Table 10.1 below.

$\mathcal{C}_L$	6		
$\mathcal{A}_{Li}$	15	$\bigcup \mathcal{A}_{Li}$	$6 \times 15 = 90$
$\mathcal{A}_{Ri}$	5	$\bigcup \mathcal{A}_{Ri}$	$6 \times 5 = 30$
$\lvert\mathcal{C}\rvert$	3		
$\mathcal{A}_{Ci}$	20		
$\lvert A\rvert = \lvert\bigcup \mathcal{A}_{Li}\rvert + \lvert\bigcup \mathcal{A}_{Ri}\rvert + \lvert\bigcup \mathcal{A}_{Ci}\rvert$			180

Table 10.1: Showing experimental setup

section 10.2 next presents experimental results and plots for the network latency data and section 10.3 presents experimental results and plots for detection delays data.

10.2 Experimental Results and Analysis

This section presents experimental results for the distributed setting and results for experiments exploring the all issues discussed in this chapter, namely *network delays*

10.2.1 A note on the presentation of results

When presenting results in this section , for each experiments, all or *some* of the following will be shown :

1. A histogram to show the *density distribution* of the data to highlight the general shape and any show any outliers.
2. In the case of network delays experiment, a *time series* and various a *probability density function fits* and a table showing corresponding *parameter* and *Maximum likelihood estimation*, MLE [5] are presented.
3. For the *bootstrap* estimation of a statistic, the density distribution of the replicates is shown in a figure followed by the corresponding *QQ plot* of the replicates against *normal quartiles*.
4. A table presenting numerical results for the bootstrap. The table shows;
   - The bootstrap estimate of a statistic,e.g. mean and the related bias and standard error.

[5]MLE is a statistical method used to determine a mathematical model to fit some data.

- Results of the nonparametric bootstrap confidence interval estimates as discussed in section 7.3. The *Bootstrap-t* and the more accurate *BCa* type confidence intervals are presented.

5. Regarding the jacknife-after-bootstrap we have observed (as it has also been documented elsewhere [83] pp 279-280, [107, 237] that the jackknife-after-bootstrap technique grossly over-estimates this error, but that the accuracy increases (and the estimate converges) with the increase in B, i.e. the jackknife-after-bootstrap method is only reliable for large values of B [83] p. 280. Improvements on this in the form of the *Weighted jackknife-after-bootstrap* [237], have to my knowledge not been implemented in statistical software packages in use today. Therefore we do not use the procedure, the bootstrap estimates are sufficient for our purposes. In Appendix D, page 303 however, we experimented with the experimental implementations of the procedure available and present Figure D.1 page 307 showing results of an experiment to investigate the effect of increasing the number of bootstrap replicates, B, on the *accuracy* of the error estimates. Therefore we do not use this procedure.

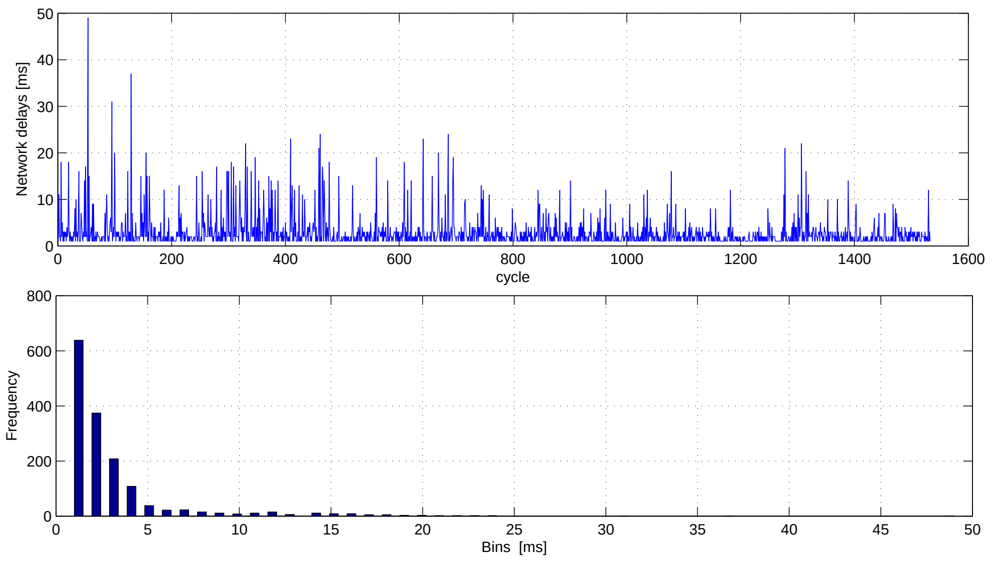
10.2.2 Results for network latency

The time series and the distribution of the *network latency* quantity Δt, as determined by experiment is shown in Figure 10.2 (a). Figure 10.2 (b) shows various probability distribution fits on the data. Observation of these figures and the table suggest that the *lognormal*, *log logistic* or *poisson* probability distributions possibly provide the best fits for the distribution of network delays in these experiments. This comparable with other results done elsewhere in [123].

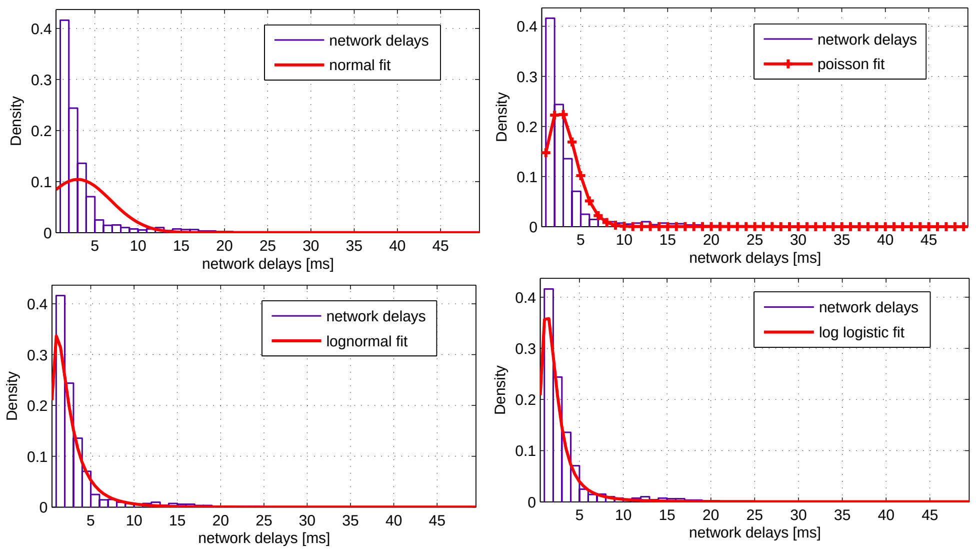
Furthermore, Table H.1 in Appendix H, page 345 shows parameter estimates for various hypothesised distributions and the maximum likelihood estimators to accompany these distribution fits.

The next set of plots in Figure 10.3 are for the bootstrap analysis of the network delays data. Figure **a)** shows the distribution of bootstrapped means for network delays and **b)** shows the corresponding QQ plot.

Table 10.2 gives numerical results for the bootstrap, showing for example confidence intervals of $(2.746, 2.958)$ at 95% confidence for the *BCa* and $(2.845, 2.956)$ at 95% for the *Bootstrap-t* type confidence intervals.

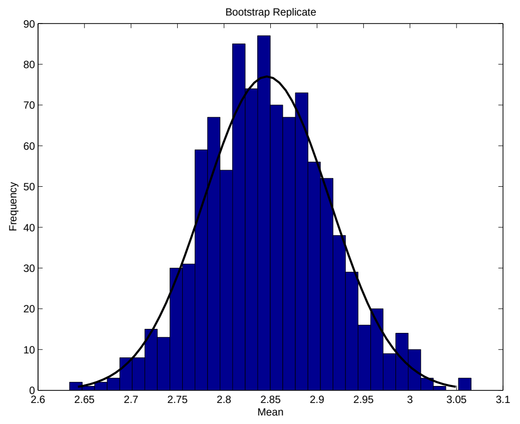


(a) time series and histogram

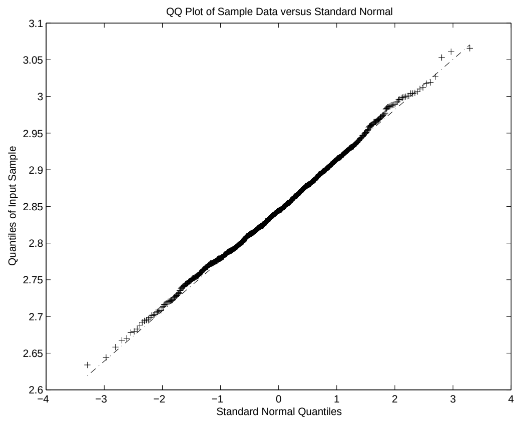


(b) probability density estimation fits

Figure 10.2: Network delays profile during experiments



(a) Density of bootstrapped Means of the network delays



(b) QQ plot

Figure 10.3: Bootstrap mean plots

The Bootstrap										
Statistic				$\hat{\theta}$	$\widehat{bias}$	$\hat{se}$				
				1660.953	0.1510	8.5083				
BCa Confidence Intervals										
α	**0.025**	**0.050**	**0.100**	**0.160**	**0.840**	**0.900**	**0.950**	**0.975**	**zo**	**ahat**
	2.719	2.746	2.771	2.787	2.913	2.935	2.958	2.988	0.01504	0.0120
Bootstrap-t Confidence Intervals										
	2.800	2.845	2.774	2.783	2.918	2.940	2.956	2.970		

Table 10.2: Results of the Bootstrap showing parameter and confidence interval estimates and errors for the network delays experiments

10.3 Results for detection delays in the distributed setting

This section presents a summary of the main results for the detection delays experiments in the distributed setting (recall the distribution architecture in Figure 6.4, page 123) and the data collection setup and scenarios discussed in the experimental design, section 6.6 in page 125.

The scheduled experiments used the experimental setup as summarised in Table 10.1, page, 189.

10.3.1 Results for scenario 1

The next set of figures and tables present results for *scenario 1* (recall that this is where all detection delays are recorded by each local controller as discussed in experimental design, section 6.6 page 125.

Figure 10.4 shows distributions of detection delays for a selection of instances of this scenario, i.e. six controllers.

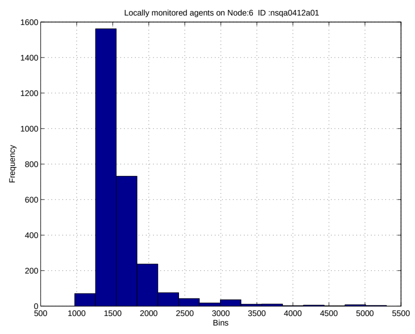
For each dataset, (i.e. data for every client node) a *nonparametric bootstrap*[6] of the mean of the detection delays was performed, and Figure 10.5 shows the distributions of the resulting bootstrap replicates for the mean of detection delays for each of the nodes. The numerical bootstrap estimates of the mean and its related *bias* and *standard* statistics are presented at the top of Table 10.3, where also results of the other five nodes can be seen.

Figure 10.6 then presents the corresponding QQ plots, showing bootstrap replicates against normal quartiles. These figures demonstrate that apart from deviations at the tails the replicates of the mean are normally distributed.

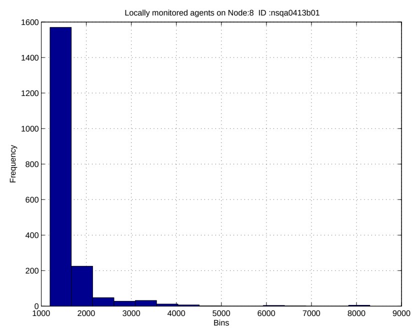
[6]No assumption is made of the underlying probability distribution.

In addition, the *bootstrap confidence intervals*, (both the *BCa* and the *Bootstrap-t*) were also computed. The numerical results are shown in the middle part of Table 10.3, where the second column shows the α value for the confidence limits. For example, to determine the 95% confidence limits, we inspect the row entry for $\alpha = 0.975$ and for $\alpha = 0.025$ giving upper and lower limits $[1677.886, 1646.089]$ *ms* for the client node *nsqa0412a01* [7] in this experiment.

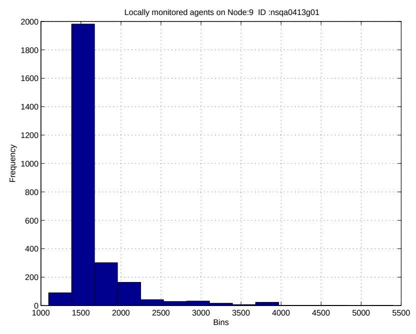
[7]Hostnames used as identifiers of machines running agent containers instead of generic labels like L_1 discussed in the data collection setup.



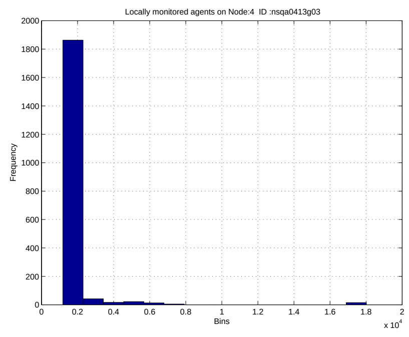
(a) Node:nsqa0412a01



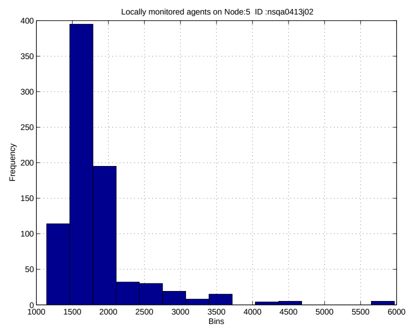
(b) Node:nsqa0413b01



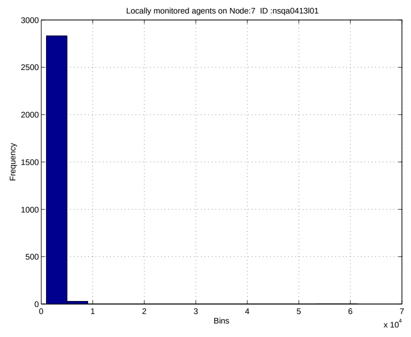
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03

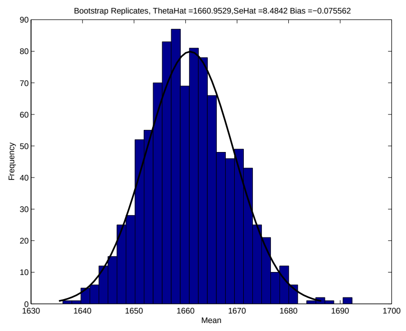


(e) Node: nsqa0412j02

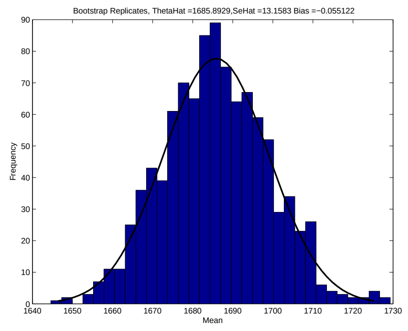


(f) Node:nsqa0412l01

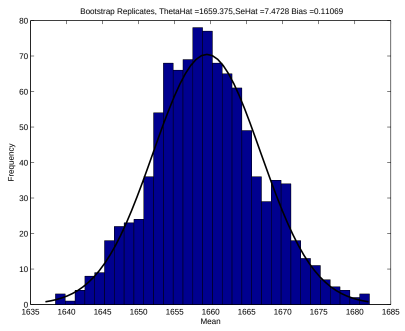
Figure 10.4: Detection delays, (in ms*) for locally monitored agents at each node. Each node has a unique identifier*



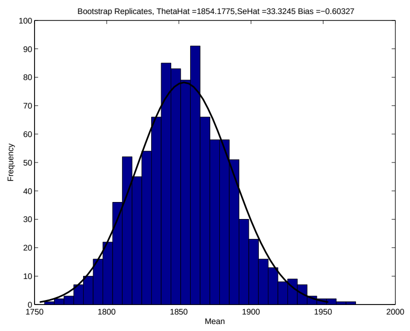
(a) Node: nsqa0412a01



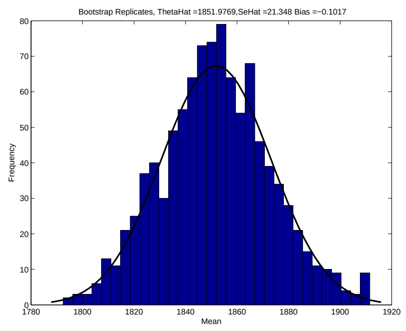
(b) Node: nsqa0413b01



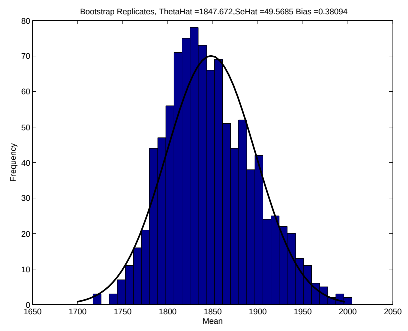
(c) Node: nsqa0412g01



(d) Node: nsqa0413g03

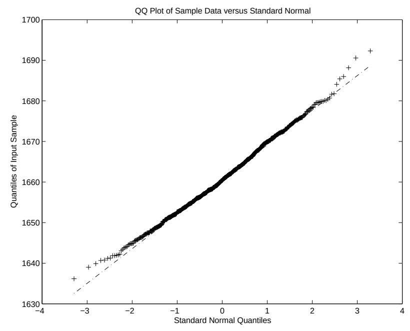


(e) Node: nsqa0412j02

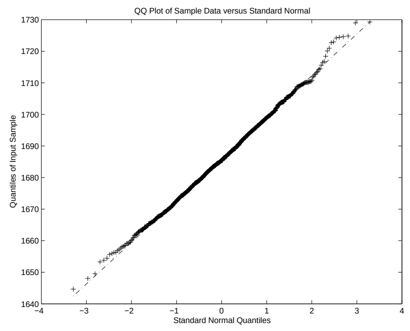


(f) Node: nsqa0412l01

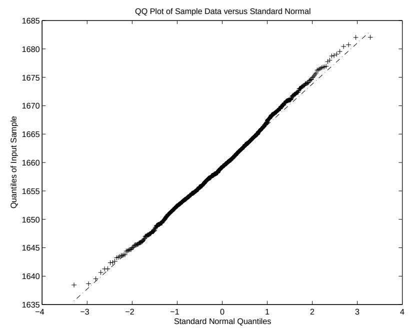
Figure 10.5: Bootstrap Replicates Density Estimation: Detection delays for locally monitored agents



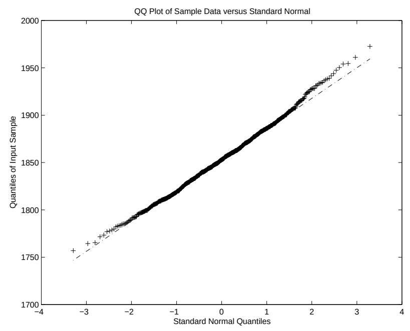
(a) Node: nsqa0412a01



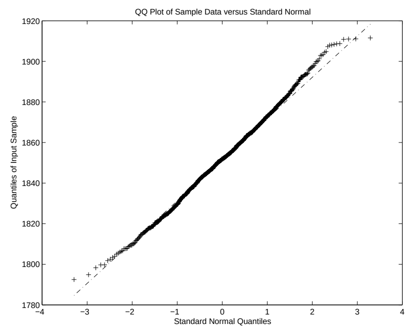
(b) Node: nsqa0413b01



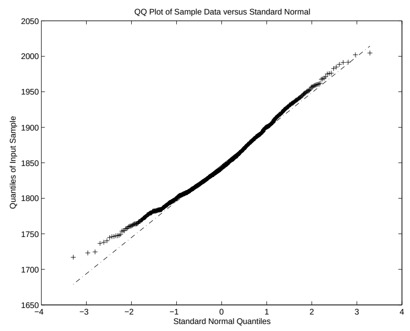
(c) Node: nsqa0412g01



(d) Node: nsqa0413g03



(e) Node: nsqa0412j02



(f) Node: nsqa0412l01

Figure 10.6: Showing the bootstrap replicates' qqplots. The figure shows that apart from some deviations at the tails, the replicates are normally *distributed.*

The Bootstrap							
		Nodes					
Statistic		nsqa0412a01	nsqa0413b01	nsqa0413g01	nsqa0413g03	nsqa0413j02	nsqa0413l01
$\hat{\theta}$		1660.953	1685.893	1659.375	1854.178	1851.977	1847.672
$bi\hat{a}s$		-0.0755	-0.0551	0.11069	-0.6033	-0.102	0.381
$\hat{se}$		8.484	13.158	7.472	33.325	21.35	49.569
Confidence Intervals							
	α						
	0.025	1646.089	1663.014	1644.897	1798.309	1812.624	1772.723
	0.050	1648.32	1666.385	1647.012	1806.002	1818.275	1782.276
	0.100	1650.648	1670.137	1649.868	1816.397	1825.107	1796.012
	0.16	1653.206	1673.422	1651.928	1825.247	1831.454	1806.606
	0.84	1669.553	1698.932	1666.977	1893.080	1872.956	1907.023
$BC_a points$	**0.9**	1672.190	1703.182	1669.099	1904.864	1879.848	1927.294
	0.95	1675.384	1708.713	1672.056	1920.293	1888.257	1956.207
	0.975	1677.886	1713.793	1675.284	1934.157	1894.522	1985.674
	z0	0.003759951	0.002506631	-0.003759951	0.05893987	-0.02506891	0.05893987
	ahat	0.01254519	0.0251825	0.01165997	0.03240697	0.01921675	0.06216931
	0.25	1655.620	1679.916	1654.137	1828.536	1838.525	1825.699

	0.5	1660.612	1687.441	1659.257	1857.283	1851.336	1857.199
	0.1	1650.106	1671.303	1650.396	1807.367	1825.623	1803.543
	0.16	1653.558	1675.754	1652.46	1818.486	1831.434	1811.687
	0.84	1670.000	1700.852	1666.666	1894.468	1873.574	1916.205
$Bootstap - t$	**0.9**	1672.985	1704.029	1669.423	1905.653	1877.411	1955.739
	0.95	1676.172	1708.785	1672.324	1921.87	1884.27	1996.275
	0.975	1679.208	1718.862	1674.188	1932.500	1891.330	2035.485

Table 10.3: Bootstrap : showing parameter and confidence interval estimates and errors for agents monitored locally at given nodes

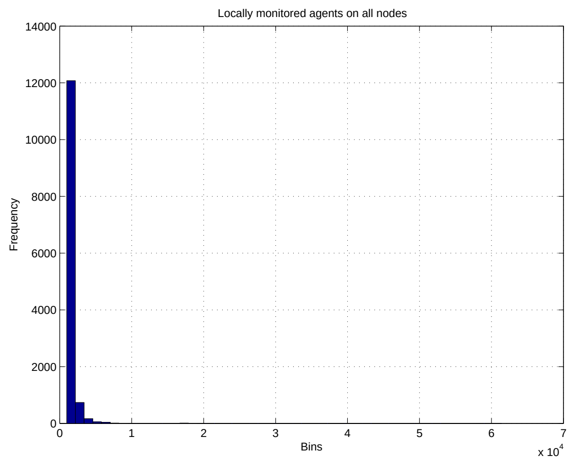
Summary Recall that these were *not* controlled experiments, i.e., local conditions (e.g. load profile) could potentially vary significantly as nodes were machines randomly chosen in the network. Inspecting Table 10.3 we can observe across the given nodes comparable detection delays in the approximate range $[1600, 1900]$ *ms* , a range also suggested by results of the *Bootstrap-t* and *BCa* confidence intervals.

Experiments on errors associated with the bootstrap show that error estimates converge and also bias estimates converge.

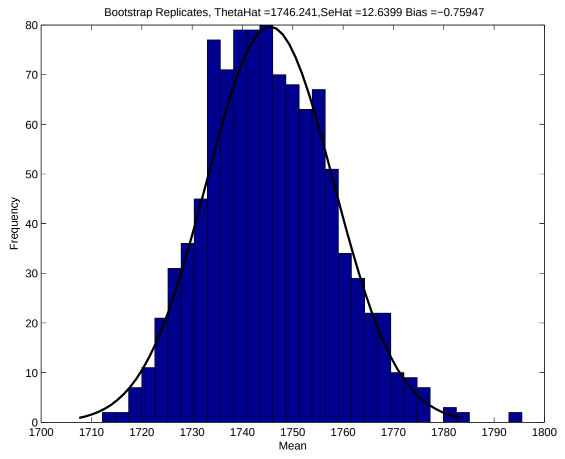
10.3.2 Results for scenario 2

The next set of figures and tables present results for this scenario (Recall that this is where all detection delays recorded by local controllers are collated into a single dataset to give a combined density distribution for all locally monitored agents as discussed in experimental design, section 6.6 page 125.

Figure 10.7 shows the distribution of detection delays and the distribution of bootstrap replicates for the mean for this dataset



(a) Distribution of detection delays.



(b) Distribution of the bootstrap replicated of the mean of detection delays

Figure 10.7: (a) This figure shows the distribution of the detection delays. As it is outliers were not filtered out hence the skewed distribution. (b) Showing the distribution of bootstrap replicates for the mean for scenario 1 (all data for locally monitored agents), showing a peak at $\approx 1750ms$ *for this dataset.*

The Bootstrap										
Statistic				$\hat{\theta}$	$bi\hat{a}s$	$\hat{se}$				
				1746.241	-0.759	12.64				
BCa Confidence Intervals										
α	**0.025**	**0.050**	**0.100**	**0.160**	**0.840**	**0.900**	**0.950**	**0.975**	**zo**	**ahat**
	1725.492	1728.414	1731.976	1734.778	1760.379	1764.624	1770.299	1775.161	0.06270678	0.04326342
Bootstrap-t Confidence Intervals										
	1738.187	1744.887	1731.876	1734.896	1761.116	1766.963	1772.215	1777.454	n/a	n/a

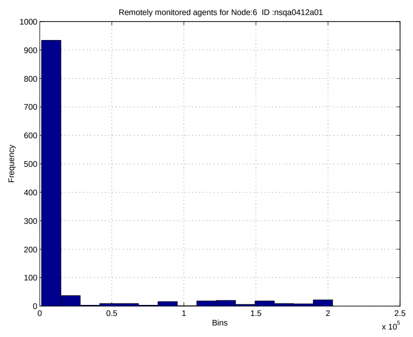
Table 10.4: Bootstrap : showing parameter and confidence interval estimates and errors for the experiments in the distributed setting for all locally monitored agents

Summary For the combined datasets for all locally monitored agents the mean detection delay is $\approx 1750ms$ and the 95% BCa confidence intervals are $[1725.49, 1775.16]ms$ and the Bootstrap-t intervals are $[1738.18, 1766.96]ms$

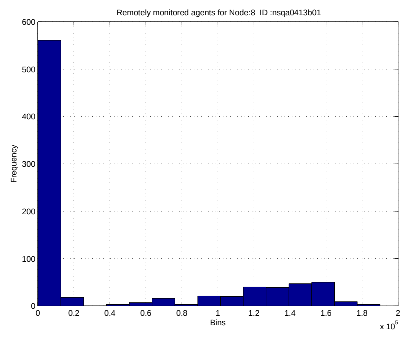
10.3.3 Scenario 3

The next set of figures and tables present results for scenario three (where detection delays were recorded by *all* remote controllers on the *cluster* for *each* given *client* node). The data collection and analysis for this scenario is again as discussed in experimental design, section 6.6 page 125.

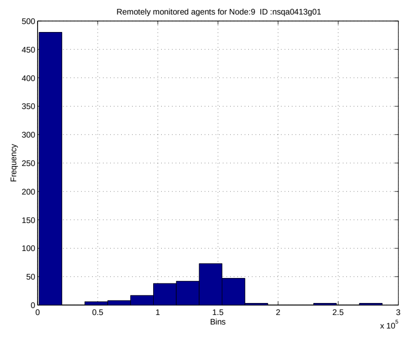
Figure 10.8 then shows the distribution of detection delays for each client node and the corresponding distributions of bootstrap replicates are shown in Figure 10.9. Table 10.5 presents the numerical results for the data analysis, showing bootstrapping results for the mean of detection delays and confidence intervals estimates



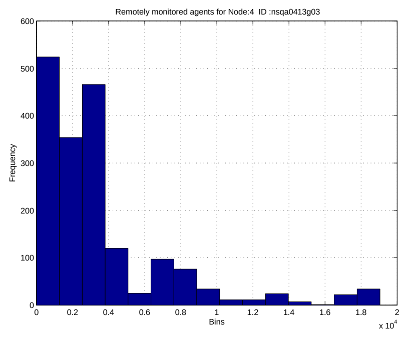
(a) Node:nsqa0412a01



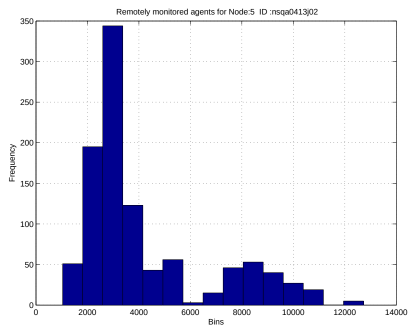
(b) Node:nsqa0413b01



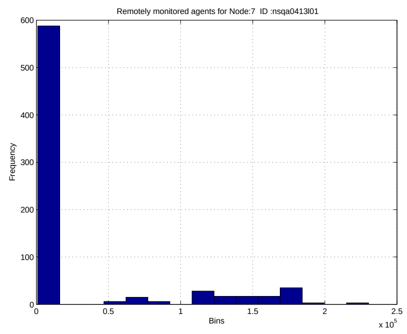
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03

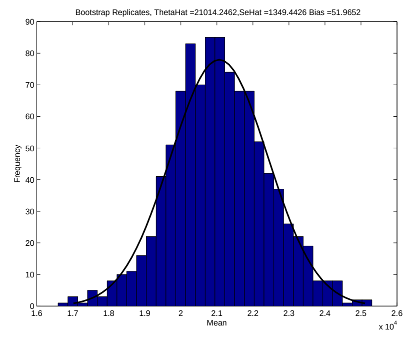


(e) Node:nsqa0412j02

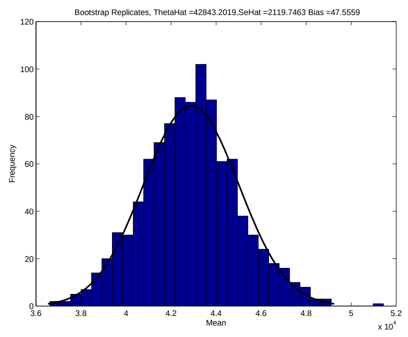


(f) Node:nsqa0412l01

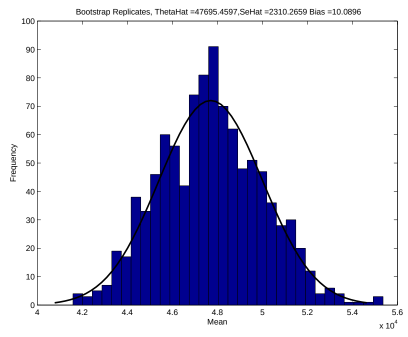
Figure 10.8: The figure shows distributions of detection delays for remotely monitored agents for each client node. Again, outliers were not filtered off.



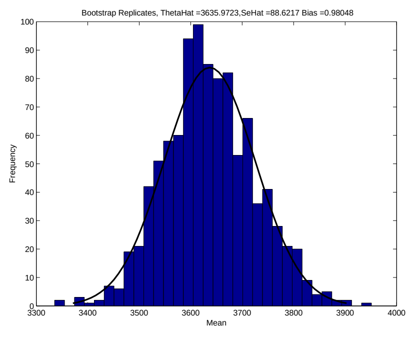
(a) Node:nsqa0412a01



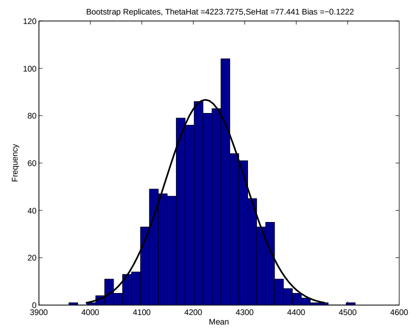
(b) Node:nsqa0413b01



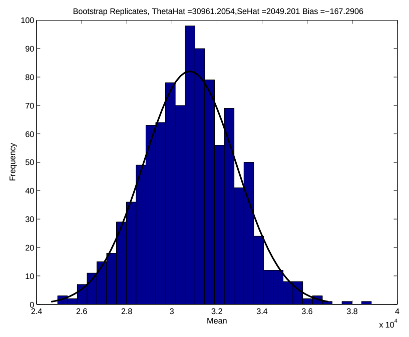
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03



(e) Node:nsqa0412j02



(f) Node:nsqa0412l01

Figure 10.9: Corresponding bootstrap replicates distribution and density estimation for detection delays for remotely monitored agents

The Bootstrap							
		Nodes					
Statistic		nsqa0412a01	nsqa0413b01	nsqa0413g01	nsqa0413g03	nsqa0413j02	nsqa0413l01
$\hat{\theta}$		21014.247	42842.209	47695.460	3635.9723	4223.7275	30961.2054
$\hat{bias}$		51.965	47.55	10.090	0.98048	-0.1222	-167.291
$\hat{se}$		1349.443	2119.74	2310.266	88.6217	77.441	2046.201
Confidence Intervals							
	α						
	0.025	18654.67	38856.86	43016.82	3452.742	4067.49	27007.01
	0.05	19039.60	39474.62	43784.95	3485.913	4090.035	27613.62
	0.1	19440.13	40260.12	44824.36	3522.030	4122.482	28338.01
	0.16	19777.15	40861.48	45395.45	3545.667	4146.643	28899.34
	0.84	22466.95	44970.21	50086.39	3724.099	4307.125	33073.9
$BC_a points$	**0.9**	22947.30	45586.77	50763.75	3756.524	4330.23	33687.00
	0.95	23515.15	46292.75	51749.55	3790.276	4361.541	34537.89
	0.975	23972.88	46877.04	52472.67	3811.732	4385.44	35077.86
	z0	0.04011681	0.02130795	-0.003759951	-0.001253314	0.01002668	0
	ahat	0.01365313	0.006024687	0.006568672	0.008712242	0.00681802	0.01123875
	0.25	20295.79	41437.27	46664.93	3576.712	4162.32	29873.07

$Bootstap - t$	**0.5**	21042.62	42874.71	48076.17	3624.043	4225.289	31285.93
	0.1	19259.52	40046.58	44874.92	3531.034	4118.288	28808.95
	0.16	19666.10	40831.58	45980.25	3553.07	4135.384	29460.45
	0.84	22579.15	45156.83	50092.87	3727.483	4298.497	33457.34
	0.9	22999.31	45685.71	50804.87	3771.143	4319.887	33913.49
	0.95	23347.34	46335.24	51781.79	3804.642	4348.613	34641.12
	0.975	23826.83	46953.61	52145.7	3854.531	4371.131	34987.83

Table 10.5: Bootstrap: showing parameter and confidence interval estimates and errors for the experiments in the distributed setting for remotely monitored agents

Summary For the datasets in this scenario , mean detection delays as shown in Table 10.5 were computed to range from $\approx 21014ms$ and 95% BCa confidence intervals $[18654, 23972]ms$ and Bootstrap-t intervals $[21042, 23826]ms$ at the lower end (for node *nsqa0412a01*), and at the extreme end (for node *nsqa0413g01*) mean detection delays was at $\approx 47695ms$ with the corresponding 95% BCa confidence intervals computed as $[43016, 52472]ms$ and Bootstrap-t intervals of $[46664, 52145]ms$.

As these were not controlled experiments, the differences could be attributed to differences in load profiles at the local nodes or cluster nodes. This assumption can be easily checked by inspecting recorded resource utilisation measures during the periods of the experiments.

10.3.4 Scenario 4

The next set of figures and tables present results for this scenario, where detection delays were recorded by *all* remote controllers on the *cluster* and the data pooled, i.e. a collective population of *all* remotely monitored agents in the cluster as discussed in experimental design, section 6.6 page 125.

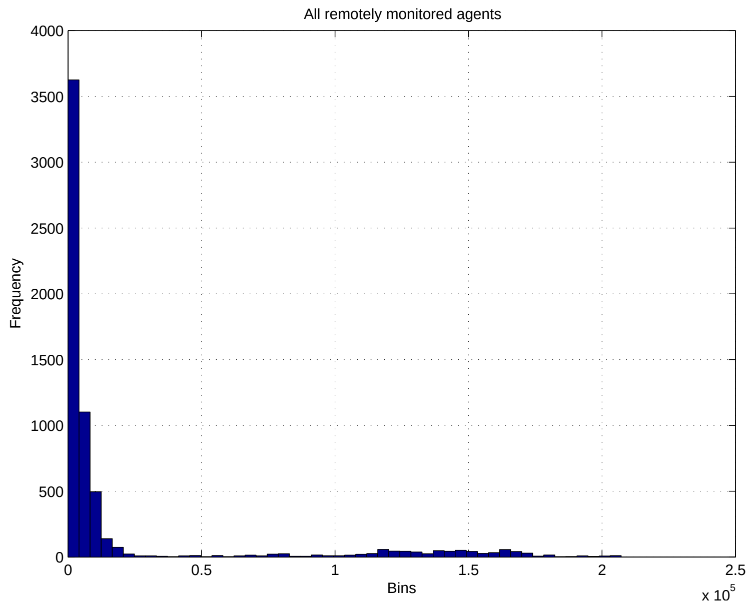


Figure 10.10: Profile of detection delays in the cluster. Showing Detection delays from all controllers in the cluster. The various peaks observed represents possibly various cluster's node mean delays

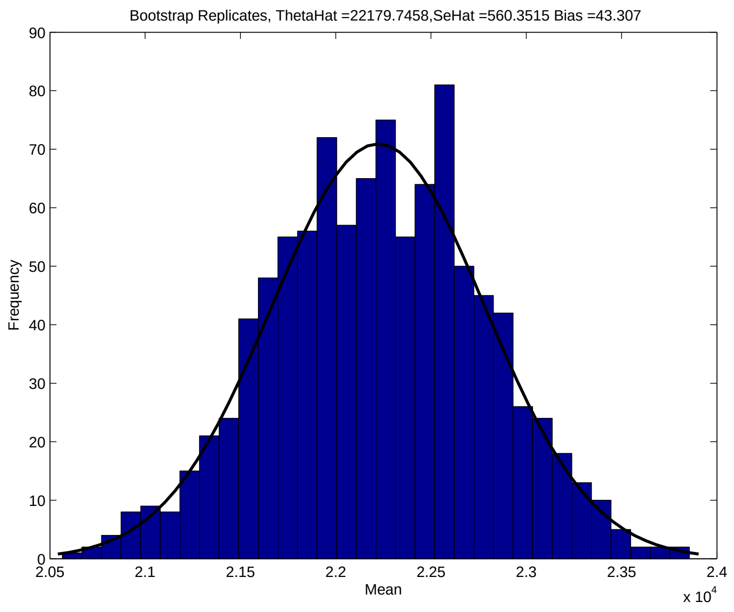


Figure 10.11: Bootstrap distribution of the mean

The Bootstrap										
Statistic				$\hat{\theta}$	$\hat{bias}$	$\hat{se}$				
				22179.746	43.307	560.3515				
BCa Confidence Intervals										
α	**0.025**	**0.050**	**0.100**	**0.160**	**0.840**	**0.900**	**0.950**	**0.975**	**zo**	**ahat**
	21138.68	21294.12	21455.25	21627.1	22808.43	22989.61	23190.45	23345.21	0.03760829	0.004951631
Bootstrap-t Confidence Intervals										
	21825.62	22214.53	21502.96	21682.53	22740.81	22894.61	23189.48	23450.35		

Table 10.6: Bootstrap: showing parameter and confidence interval estimates and errors for the experiments in the distributed setting for all cluster remotely monitored agents

Summary For the combined datasets of all cluster (remotely) monitored agents, the mean detection delay was of the order $22100ms$ as shown by the peak in Figure 10.11 and computed in Table 10.6 to be $\approx 22179ms$. The 95% BCa confidence intervals were computed to be $[21138, 123345]ms$ and the Bootstrap-t intervals to be $[21825, 23450]ms$ The results show that apart from the odd case as observed in the results in the last scenario due to local conditions, on average most remotely monitored agents experience similar detection delays when monitored by the cluster.

10.3.5 Scenario 5

Figures 10.12 and 10.14 below present a view of the results for scenario five, i.e. detection delays as logged by each cluster node. The data collection and analysis for this scenario is discussed in experimental design, section 6.6 page 125.

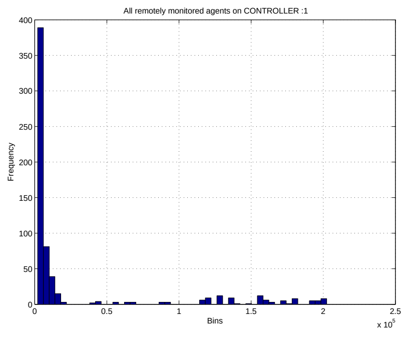
In these figures, the distributions of detection details for each *client node* on a given *cluster node*, [8] e.g. cluster node (*cruncher* and *comas*) is shown.

Figures 10.13 and 10.15 shows the corresponding bootstrap replicates for the mean for each client node agent of each cluster controller node.

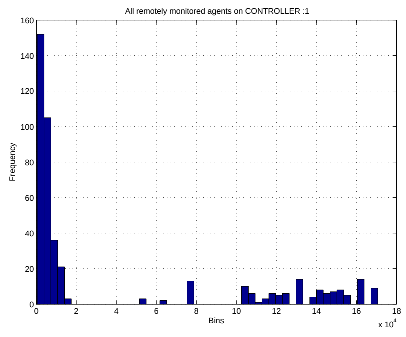
One way to interpret the figures is to say they reflect some quality of service profile for a given client node. In ideal cases that should be comparable across clients, but in reality it is affected by local load at the clients for example.Recall that these are not controlled experiments. Additionally results are affected to some extent by non-uniform network delays.

Tables 10.7 and 10.8 presents results for the computations of bootstrap mean and bootstrap confidence and associated error for each client node per cluster controller node.

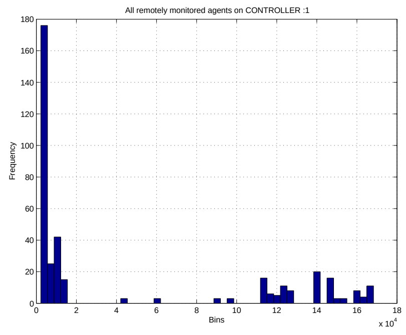
[8]Cluster nodes identified by hostname in the network instead of using generic labels like $\mathcal{C}_\infty$.



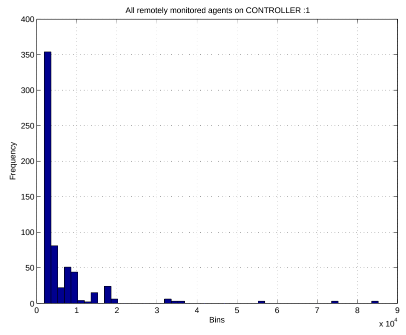
(a) Node:nsqa0412a01



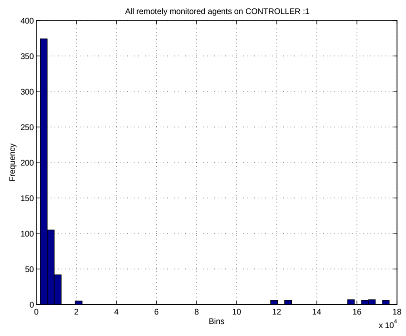
(b) Node:nsqa0413b01



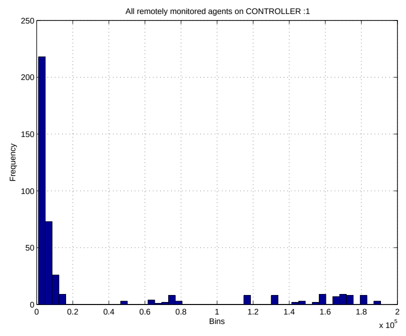
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03

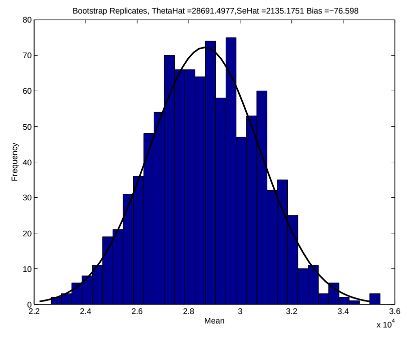


(e) Node:nsqa0412j02

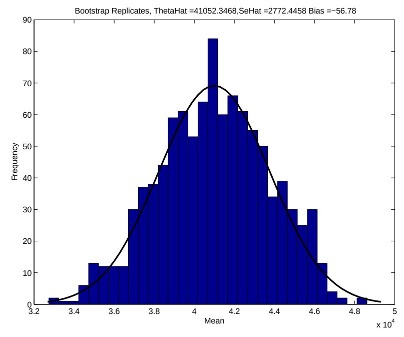


(f) Node:nsqa0412l01

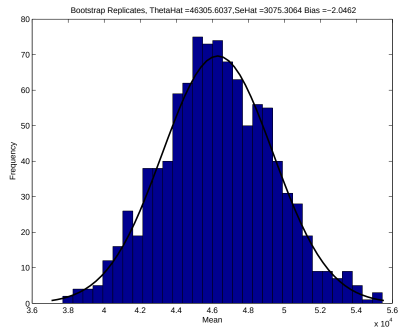
Figure 10.12: Detection delays for remotely monitored agents on cluster controller cruncher



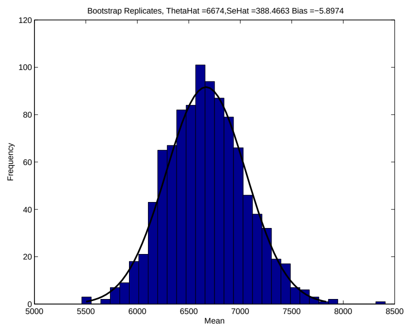
(a) Node:nsqa0412a01



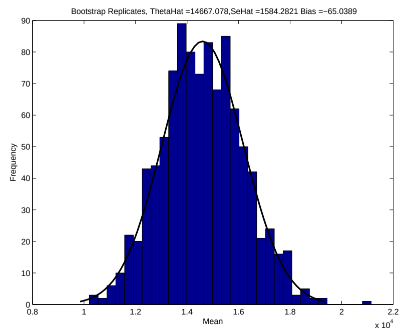
(b) Node:nsqa0413b01



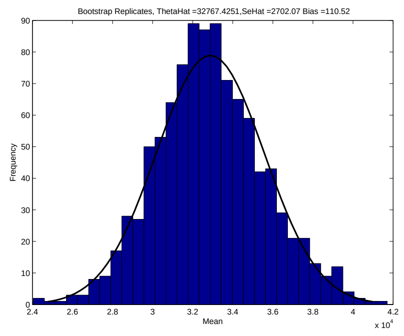
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03

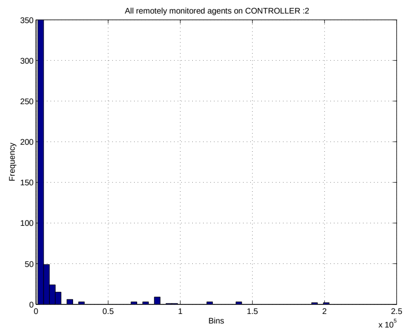


(e) Node:nsqa0412j02

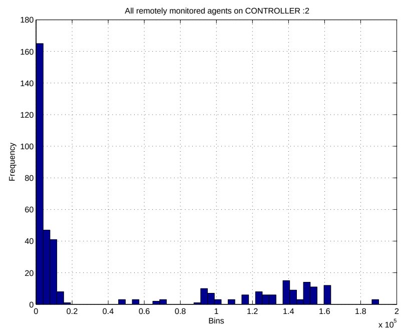


(f) Node:nsqa0412l01

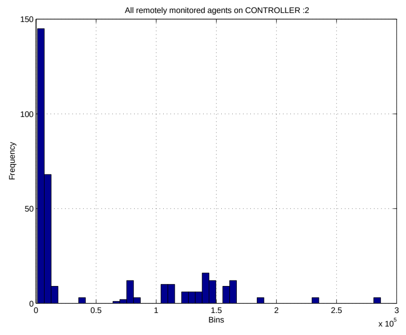
Figure 10.13: Bootstrap replicates for the mean of detection delays for remotely monitored on cluster controller cruncher



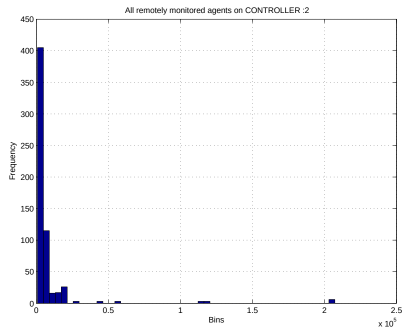
(a) Node:nsqa0412a01



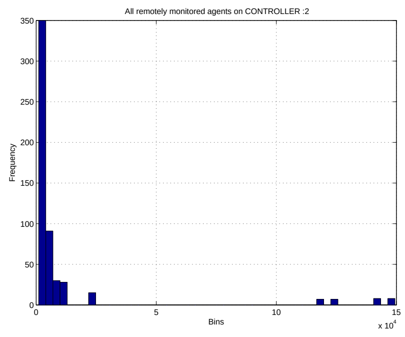
(b) Node:nsqa0413b01



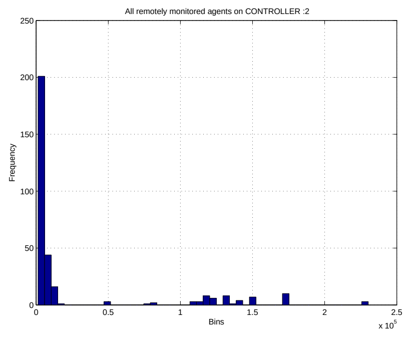
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03

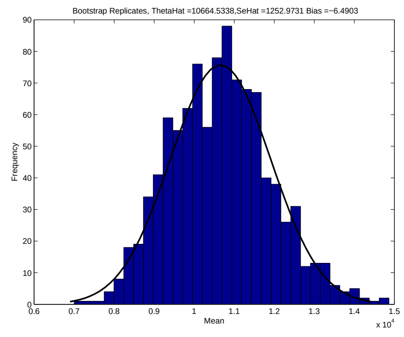


(e) Node:nsqa0412j02

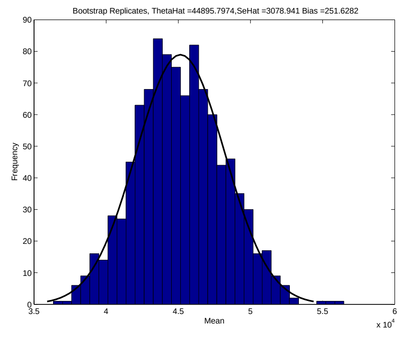


(f) Node:nsqa0412l01

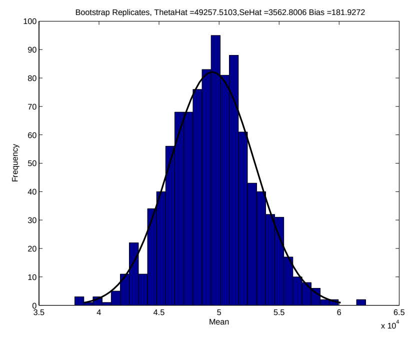
Figure 10.14: Detection delays for remotely monitored agents on cluster controller comas



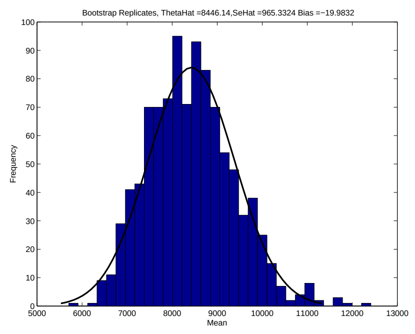
(a) Node:nsqa0412a01



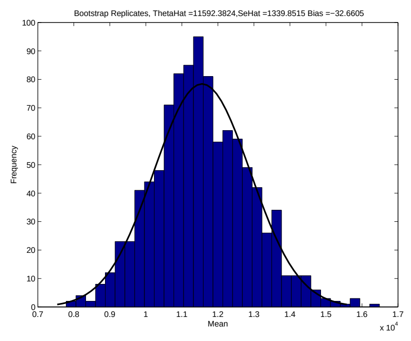
(b) Node:nsqa0413b01



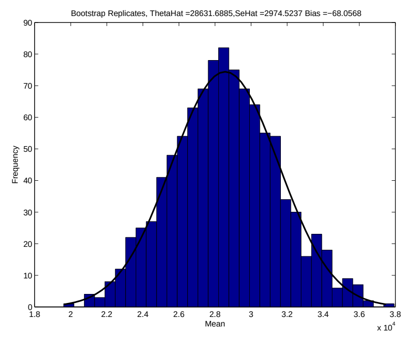
(c) Node:nsqa0412g01



(d) Node:nsqa0413g03



(e) Node:nsqa0412j02



(f) Node:nsqa0412l01

Figure 10.15: Bootstrap replicates for the mean of detection delays for remotely monitored on cluster controller comas

The Bootstrap							
		Nodes					
Statistic		nsqa0412a01	nsqa0413b01	nsqa0413g01	nsqa0413g03	nsqa0413j02	nsqa0413l01
$\hat{\theta}$		28691.497	41052.367	46305.60	6674	14667.08	32767.42
$\hat{bias}$		-76.598	-56.78	-2.046	-5.897	-65.039	110.52
$\hat{se}$		2135.18	2772.45	3075.30	388.47	1584.28	2702.07
Confidence Intervals							
	α						
	0.025	24533.41	35571.46	40389.22	5987.66	11931.89	27284.54
	0.05	25156.18	36479.31	41269.12	6087.349	12388.15	28154.51
	0.1	25899.59	37506.07	42477.02	6216.473	12768.95	29086.30
	0.16	26484.49	38307.57	43397.29	6313.37	13202.36	29819.09
	0.84	30854.5	43815.97	49517.75	7076.519	16428.66	35523.70
$BC_a points$	**0.9**	31393.01	44680.02	50375.42	7235.276	16996.99	36359.89
	0.95	32268.92	45868.88	51661.99	7422.498	17684.99	37396.93
	0.975	33043.07	46595.32	52609.35	7610.075	18248.88	38291.39
	z0	-0.01378689	0.008773312	0.01880082	0.01629380	0.05768443	-0.02256157
	ahat	0.01364055	0.008926957	0.007788226	0.03491712	0.02480227	0.01418406
	0.25	27047.47	39258.10	44379.32	6497.768	13749.38	31284.46

	0.5	28634.5	41393.6	46489.78	6742.566	14706.24	32744.7
	0.1	25651.54	37614.12	42229.45	6264.196	13128.66	29707.79
	0.16	26266.64	38238.8	42986.22	6343.522	13434.85	30363.3
	0.84	31202.68	44375.83	49889.02	7177.005	16537.25	35604.18
$Bootstap - t$	**0.9**	31854.44	45139.06	50623.25	7347.719	17166.49	36437.92
	0.95	32554.42	46078.71	51737.33	7456.396	17953.28	38372.22
	0.975	33731.72	47341.16	52921.75	7547.386	18432.07	38777.18

Table 10.7: Bootstrap results: showing parameter and confidence interval estimates and errors for the experiments in the distributed setting for remotely monitored agents on cluster controller cruncher

The Bootstrap							
		Nodes					
Statistic		nsqa0412a01	nsqa0413b01	nsqa0413g01	nsqa0413g03	nsqa0413j02	nsqa0413l01
$\hat{\theta}$		10664.53	44895.797	49257.51	8446.14	11592.38	28631.69
$\hat{bias}$		-6.490	251.62	181.93	-19.98	-32.661	68.057
$\hat{se}$		1252.97	3078.94	3562.800	965.33	1339.85	2974.52
Confidence Intervals							
	α						
$BC_a points$	**0.025**	8596.055	38955.98	42644.98	6753.435	9248.765	23231.22
	0.05	8906.755	40078.96	43709.33	7000.375	9631.675	24003.64
	0.1	9268.717	41158.53	44833.58	7302.225	10048.24	24978.87
	0.16	9575.844	41920.83	45771.72	7580.372	10331.89	25833.67
	0.84	12024.71	48078.25	52870.39	9478.053	12904.63	31767.01
	0.9	12447.32	48883.65	54008.45	9843.472	13361.22	32634.6
	0.95	12996.56	50044.43	55421.23	10187.13	13898.74	33933.07
	0.975	13601.4	51129.11	56962.32	10540.89	14461.99	34929.55
	z0	0.02256157	0.01880082	0.03384594	0.01128007	0.01504034	0.02381522
	ahat	0.03669208	0.008033102	0.01061637	0.04691313	0.02750070	0.01825531
	0.25	10071.60	42890.27	46983.35	7777.008	10952.49	26909.94

	0.5	10836.68	44675.04	49064.95	8449.172	11789.81	28878.78
	0.1	9475.183	40920.41	44845.75	7278.35	10146.07	24212.03
	0.16	9715.518	41946.14	45846.17	7547.332	10652.36	25493.22
	0.84	11974.66	47508.51	53041.9	9474.949	13000.57	31892.13
$Bootstap - t$	**0.9**	12353.53	48431.78	54796.74	9792.755	13378.07	32740.77
	0.95	13398.39	49250.12	56553.38	10105.18	13882.79	33930.96
	0.975	13941.58	50851.99	56771.18	10467.46	14534.01	34732.48

Table 10.8: Bootstrap results: showing parameter and confidence interval estimates and errors for the experiments in the distributed setting for remotely monitored agents cluster controller comas

Finally Figure 10.16 shows the detection delays distributions and corresponding distributions for bootstrap replicates of the mean for all detection delays recorded on a given cluster node. Table 10.16 then gives the corresponding numerical results for the computations of the bootstrap mean and bootstrap confidence intervals and associated estimates of errors.

Discussion Again Recall that these were *not* controlled experiments, i.e., local conditions (e.g. load profile) on each cluster controller node could potentially vary significantly as cluster nodes were server machines in the network. Nevertheless inspecting tables we can observe across the given nodes the computed mean detection delays and corresponding BCa and Bootstrap-t confidence intervals per cluster node. Also note that in the computation in these table raw data was used, i.e. outliers were not removed.

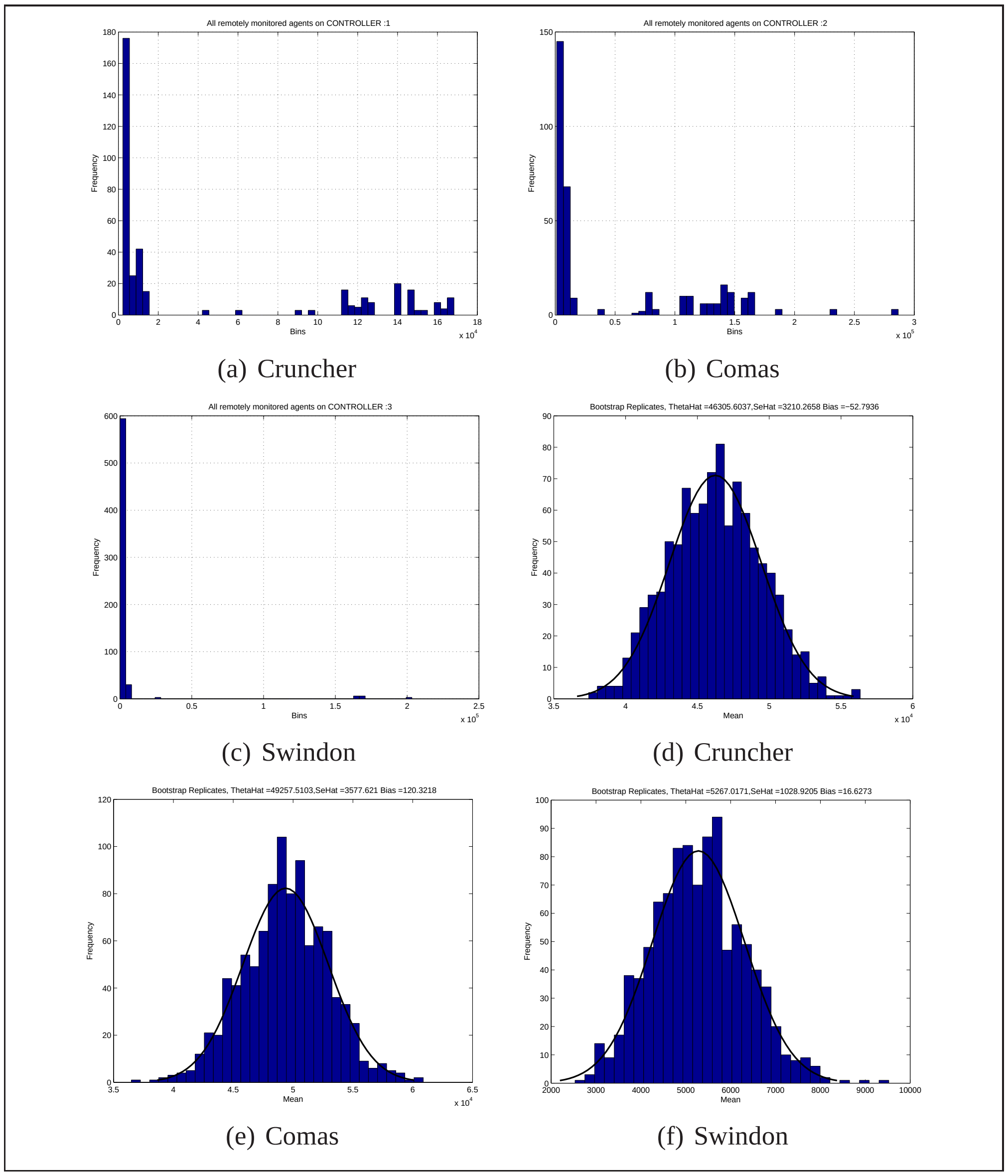


Figure 10.16: Figure (a-c) show the distribution of detection delays for remotely monitored agents on each cluster controller and figures (d-f) show the corresponding distributions for bootstrap replicates of the mean

10.4 Summary

The purpose of the chapter was to investigate the benefits of distribution of the management and monitoring scheme. The immediate benefits that were expected were those to be accrued from the notion of *distributed control*, for example in a distributed setting there is no one central point of control and a decentralised management scheme is in place and each node is independent. The particular advantages that I emphasise in this scheme are increased scalability (in that more agents can be monitored by the cluster) and redundancy of the setup, (in that a cluster node failure is not catastrophic).

CHAPTER 11

Conclusion

11.1 Discussion of other issues

Regarding the termination detection mechanism proposed, we considered the quantitative aspects in preceding chapters. We can also consider some further issue. For these, consider the following issues;

–i– In an approach where agents are given a detection protocol, and can compose their behavior with the detection protocol, it can be observed that if the detection protocol assumes correct participation by agents, a unilateral (or even strategic) deviation from the detection protocol by any agent may jeopardize the detection process. Our approach considers interactions at a protocol level, with the monitor having some awareness of the agreed protocol specification, say through a protocol library $\mathcal{P}$. One of the ways agents can put at risk the detection process is by communicating false information about protocols, or by opting not to communicate this information. One way is to enforce some norms in the society , e.g. a marketplace/auction house can stipulate and enforce rules of participation to nullify agents' incentives to deviate.

–ii– Regarding termination, other issues that need considering include detection delays, the maximum time that can elapse between termination and its detection. This has been evaluated in Chapters 7, 8, 10 for the various setups. Considerations have to be made on how this delay can be minimized for any specific detection mechanism adopted. In our framework the parameters for control waves can be adjusted for example. Other issues arise if agents execute multiple protocols. This has been considered in the conversation model given in **Chapter** 5. The setup will also need to consider the fact that agents may be executing different stages of the protocol, so that generalised society wide control waves may not useful in minimising detection delays. In future work we can consider modifying the framework to allow a given monitor to maintain agent groups and associations depending on protocols used, stages in the protocol and conversation partners for example.

–iii– Another issue relates to structuring the detection mechanisms such that there is no adverse effect on the execution of underlying protocols and that no unnecessary bottlenecks are introduced in the infrastructure of the society. The framework we describe does not require that we modify the protocols, e.g. augment or embed in the protocol some control messages. The main possible bottleneck would be the use of a single entity, the monitor, in detecting termination. We have proposed possible distribution possibilities in **Chapter** 5 and evaluation of various scenarios in **Chapter** 10 for the distributed configuration. Indeed distribution is a general problem for most services in distributed systems, services such as directory , naming services and is also well studied in distributed systems. There , there exists a number of approaches and solutions to address this concern, for example distribution, hierarchical setups e.g. as used in the Domain Name Service, DNS, group communications etc. Finally, we argue that in addition to detecting termination, with relatively simple extensions, our mechanism can also be used to provide some level visibility of the process of agent interactions which may be of value in high level management of agent societies in practical applications.

11.2 Summary

Research in multiagent systems is diverse and varied, encompassing and drawing from many fields of research. There has been some progress made in the theoretical foundations of agents and multiagent system, agent communication languages, interaction protocols, social semantics, methodologies, multiagent frameworks and others. However while progress has been made, challenges exists all round, especially in the practical aspects of development and deployment of agents and the support frameworks of management and control of agent societies say compared to work done in grid computing.

Experiments and experiences with an attempt to deploy agents for services on a global network called AgentCities [66] are recounted in [242] and [67] lists concrete challenges for this service environment, quote;

–i– Automation, i.e. management of autonomy: Understanding how to effectively automate systems in an open environment, how to control and manage deployed automated systems. This must draw on work from mathematical control theory to distributed systems and agent technology.

–ii– Interoperability, i.e. communication:- How to enable on-line software systems to interact with one another in increasingly flexible ways: configurable interaction sequences, communication about arbitrary domains.

–iii– Coordination:- Putting in place frameworks that enable automatic creation, maintenance, execution and monitoring of contracts and agreements between automated systems to fulfil their business objectives.

–iv– Knowledge acquisition (interfaces between worlds): Putting in place frameworks that enable automatic creation, maintenance, execution and monitoring of contracts and agreements between automated systems to fulfil their business objectives.

We positioned our research to make a contribution to the first point above, observing that multiagent systems are inherently distributed and are implemented on distributed systems infrastructures, and noting (as has been elsewhere [247]) that research efforts in agents infrastructure support should necessarily draw upon experiences and coordinate with distributed systems research.

11.3 Contributions

We claim to have made a number of contributions as discussed in chapter 1, page 9.

On a *theoretical level*, we have considered the distributed termination detection problem and research from distributed systems and considered it in the agent model and used this a basis for developing a class of agent control mechanisms.

1. To this end, in **Chapter** 5, section 5.6 page 110 we listed contributions towards a termination detection model for agents, where we presented definitions related to protocols and defined minimal information agents can register with interaction observers.

2. We presented an agent conversation model, defined some predicates and presented algorithms for their implementation.

3. combining all these we presented distributed a distributed protocol for termination detection and considered distribution possibilities.

On a *practical level*, we have offered a structured and systematic, methodical experimental, data collection and analysis framework, i.e.

1. In Chapter 6 we have offered a prototype implementation and used it not only to evaluate the proposed mechanism and two configurations but to explore the use of an implementation in agent middleware.
2. In Chapter 6 we defined an extensive experimental and data analysis framework that uses robust resampling methods for quantitatively evaluating a prototype in this research that can also be used evaluating future contributions.
3. The experimental work here can also set a benchmark for future work.

Aspects of these contributions have been previously published in [174] and documented [171] and is subject of papers in progress [173, 172] resulting from research discussed in this thesis.

11.4 Critical Review

The ideal definition of an agent is that of an entity that is autonomous. And, an ideal multiagent system is one with no global control. It is worth noting that the work discussed in this thesis treads on these aspects to a certain degree. We have put forward a proposal for a mechanism that contribute towards management of agents by requiring as part of society rules for participation, for agents to register partial behaviour specifications. This is not so much a problem as in multiagent agent systems, protocols are deemed public and individual agent strategies are necessarily private.

Regarding scalability, as the numbers of hosted agents increase, there are a number of issues:

–i– The size of the $c-matrix$ data structure and cost of searching it will necessarily increase. But as discussed the best solution will be in distribution and possibly exploring more efficient procedure for organising searching matrix type structures.

–ii– Regarding the control *waves* (page 106) , there maybe concerns regarding message complexity depending on the scale required and on the frequency of the generated waves.

–iii– Regarding graph algorithms for protocol graphs there will be issues with large protocol graphs as discussed in section A.4, as there is only so much graph algorithms can be improved as for graph traversal either breadth-first or depth-first search are used as a basis.

Regarding the quantitative experiments, the results and the analysis are as quoted in **Part III**, Chapters 7 to 10. In the experimental observations, there were instances of high variability and outliers in recorded data presumably due possibly to the underlying agent framework middleware and network delays, and these were considered by scheduling a number of experimental runs, using robust resampling methods (see chapter 6, page 114) that incorporates outliers and in the distributed setting, conducting experiments to estimate distributions of network delays (see chapter 10, page 191 and Appendix H, page 342) to factor into the experimental results. With the quantitative experiments we have explored and have a sense of how the mechanisms perform in an existing agent middleware, results that can also interest researchers of these agent frameworks.

11.5 Directions for future work

In addition to addressing issues raised above in the qualitative evaluation and the critical review, as discussed before, results from the area of termination detection may be used in the related area of garbage collection. We propose future work to consider a follow up and consider how to derive timely garbage collection schemes for multiagent interactions.

And evidently lurking behind is the issue how much degree of autonomy do agents really have in real applications and how much of this autonomy need be constrained when dealing with issues surrounding infrastructure support for agents, assuming that the notion of autonomy can be captured in some way. There are research efforts in the area of agents autonomy [180] and some attempts at capturing autonomy [240] that may give some directions for future work in this direction.

11.6 Related Work

The work on this thesis was inspired by a short paper [241] that discussed distributed *quiescence* detection in a multiagent negotiation and posited a solution there based on the Dijkstra and Scholten's algorithm (example of a tracing algorithm, see section 2). The work was specific to multiagent negotiation, and the algorithm there is used a basis of a quiescence detection protocol, a protocol that operates as a layer on top of an underlying *mediated* negotiation protocol.

The first contribution of that work was a formulation of the distributed quiescence detection problem in multiagent, multi-issue negotiation. The negotiation considered there is mediated, i.e. the negotiation model comprises of agents and mediators. Mediators facilitate the negotiation by managing information flow and enforcing negotiation rules [241].

The negotiation protocol comprises two general type of messages, namely OFFER (sent by agents to mediators) and NOTIFY (sent by mediators to agents). The form and content of these messages varies according to domain specific rules enforced by mediators and negotiation policies of the agents [241].

Applying the Dijkstra and Scholten's algorithm to the negotiation model involves requiring that agents augment their behaviors by passing and tracking ACK messages according to the detection protocol, i.e. their overall behavior is then a composition of their basic negotiation behavior with the transition diagram representing the algorithm [241].

As a second contribution, the work identifies and discusses circumstances under which agents may have incentives to deviate from the basic protocol and discusses a modification to the negotiation framework that is argued to present limited incentive for agents to deviate [241], this is because they compose their behavior with the detection protocol and hence unilateral deviation from the detection protocol by any agent may jeopardize the detection process.

The wider area of monitoring through *overhearing*[1] assuming petri-nets is discussed in [103, 101], where the work focuses on and explores the use of colored petri-nets to represent legal joint conversation states and messages and considers the general *overhearing* approach and provides a formalisation and the building blocks for the overhearing.

We have focused our work on termination detection and provided a runtime mechanism for making this explicit. We considered a distributed systems centric view to design a class of controllers and an architecture for termination detection and provided an extensive experimental framework to provide benchmarks for this and future work. We viewed protocols as finite state machine graphs. Finite state machine formalism is

[1]Origionally discussed by [183] within BDI frameworks and by [38].

by far the most widely used with interaction protocols, and exploring and working with graphs brings the benefits of results and techniques from algorithmic graph theory.

Termination detection is semantically related to problems like garbage collection and so we can position our future work to venture into that area further enhancing research effort in multiagent infrastructures.

References

[1] FIPA interaction protocol library specification, November 08 2000.

[2] T Agerwala. *Towards a theory for the analysis and synthesis of systems exhibiting concurrency*. PhD thesis, Johns Hopkins University,Baltimore, MD, 1975.

[3] Alfred V. Aho, J. D. Ullman, and M. Yannakakis. Modeling communications protocols by automata. In *Proceedings of the 20th Symposium on the Foundations of Computer Science*, pages 267–273, October 1979.

[4] Mukesh Singhal Ajay D. Kshemkalyani. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008.

[5] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata*, 2007.

[6] Eduardo Alonso and Esther Mondragón. Agency, learning and animal-based reinforcement learning. In *Agents and Computational Autonomy*, Lecture Notes in Computer Science, pages 1–6. Springer Verlag, 2003.

[7] Vipin Kumar Ananth Grama, George Karypis and Anshul Gupta. *Introduction to parallel computing*. Pearson Addison Wesley., 2003.

[8] T. W. Anderson. On the distribution of the two-sample cramer-von mises criterion. *Ann. Math. Statist.*, 33(3):1148–1159, 1962.

[9] T. W. Anderson and D. A. Darling. Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. *Ann. math. stat.*, 23:193–212, 1952.

[10] J.P. Ansart. Protocol specification, testing, and verification vi. In G.V. Bochman and B. Sarikaya, editors, *Proc. IFIP WG 6.1 6th Intl. Workshop on Protocol Specification, Testing and Verification*, Montreal, Canada, 10-13 June 1982.

[11] J.P. Ansart, O. Rafiq, and V. Chari pwd. Protocol description and implementation language (pdil), protocol specification, verification and testing ii. In *Proc. IFIP WG 6.1 2nd Workshop on Protocol Specification, Verification and Testing*, Idyllwild, North Holland, May 1982.

[12] P. Anthony, W. Hall, V Dung Dang, and N. R. Jennings. Autonomous agents for participating in multiple auctions. In *Proceedings IJCA1 Workshop on E-Business and the Intelligent Web*, Seattle WA, 2001.

[13] Eshrat Arjomandi. *A study of parallelism in graph theory.* PhD thesis, 1976.

[14] Alexander Artikis. Dynamic protocols for open agent systems. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 97–104, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[15] J. Ayache and J. Courtiat. A specification and implementation language for protocols, protocol specification, testing, and verification. In *Proc. IFIP WG 6.1 2nd Workshop on Protocol Specification, Verification and Testing*, North Holland, 1982.

[16] Ozalp Babaoglu and Keith Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In S. Mullender, editor, *Distributed Systems*, pages 55–96. Addison-Wesley, 1993.

[17] S. Bapat and A. Arora. Message efficient termination detection in wireless sensor networks. In *Proc. INFOCOM Computer Communications Workshops IEEE Conference on*, pages 1–6, 13–18 April 2008.

[18] Mihai Barbuceanu and Mark S. Fox. The design of a coordination language for multi-agent systems. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 341–355, London, UK, 1997. Springer-Verlag.

[19] Mihai Barbuceanu and Mark S. Fox. Integrating communicative action, conversations and decision theory to coordinate agents. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 49–58, New York, NY, USA, 1997. ACM.

[20] Mihai Barbuceanu and Wai-Kau Lo. Conversation oriented programming for agent interaction. In *Issues in Agent Communication*, pages 220–234, London, UK, 2000. Springer-Verlag.

[21] Bernhard Bauer, Jörg P. Müller, and James Odell. Agent uml: A formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):207–230, 2001.

[22] Martin Beer, Mark D'inverno, Michael Luck, Nick Jennings, Chris Preist, and Michael Schroeder. Negotiation in multi-agent systems. *Knowl. Eng. Rev.*, 14(3):285–289, 1999.

[23] F. Bellifemine, A. Poggi, and G.Rimassa. Jade a fipa-compliant agent framework. In *PAAM 99*, pages 97–108, 1999.

[24] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, NJ, April 2007.

[25] Federico Bergenti and Alessandro Ricci. Three approaches to the coordination of multiagent systems. pages 367–372, 2002.

[26] G. V. Bochmann and C. A. Sunshine. Formal methods in communication protocol design. *IEEE Transactions on Communications*, COM-28:624–631, 1980.

[27] Olivier Boissier, Julian A. Padget, Virginia Dignum, Gabriela Lindemann, Eric T. Matson, Sascha Ossowski, Jaime Simão Sichman, and Javier Vázquez-Salceda, editors. *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, AAMAS 2005 International Workshops on Agents, Norms and Institutions for Regulated Multi-Agent Systems, ANIREM 2005, and Organizations in Multi-Agent Systems, OOOP 2005, Utrecht, The Netherlands, July 25-26, 2005, Revised Selected Papers*, volume 3913 of *Lecture Notes in Computer Science*. Springer, 2006.

[28] T. Bolognesi and E. Brinksma. Introduction to the iso specification language lotos. In P.H.J.van Eijk, C.A.Vissers, and M.Diaz, editors, *The Formal Description Technique LOTOS*, pages 23–73. Elsevier, 1989.

[29] Alan H. Bond and Les Gasser, editors. *Distributed Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[30] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, July 2005.

[31] Wassim Bouaziz and Eric Andonoff. Dynamic execution of coordination protocols in open and distributed multi-agent systems. In *KES-AMSTA '09: Proceedings of the Third KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 609–618, Berlin, Heidelberg, 2009. Springer-Verlag.

[32] B.Pehrson. Protocol verification for osi. *Computer Networks and ISDN Systems*, 18:185–201, 1989/90.

[33] G. W. Brams. *Petri Nets: theory and practice*. Masson, 1985.

[34] D. Brand and P. Zafiropulo. Synthesis of protocols for unlimited number of processes. In *Computer Network Protocols*, pages 29–40, Gaithersberg, MD, May 1980.

[35] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[36] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. volume 8, pages 203–236, Hingham, MA, USA, 2004. Kluwer Academic Publishers.

[37] J. Brzezinski and J. Helary. Distributed termination detection : General model and algorithms. In *Proceedings of the 13th IEEE International Conference of Distributed Computing Systems*, page 21, Pittsburgh, USA, 25-28 May 1993. INRIA.

[38] Paolo Busetta, Luciano Serafini, Dhirendra Singh, and Floriano Zini. Extending multi-agent cooperation by overhearing. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 40–52, London, UK, 2001. Springer-Verlag.

[39] Lawrence Cabac and Daniel Moldt. Formal semantics for AUML agent interaction protocol diagrams. In James Odell, Paolo Giorgini, and Jörg P. Müller, editors, *The Fifth International Workshop on Agent-Oriented Software Systems (AOSE-2004). Proceedings*, pages 97–111, New York, USA, July 2004. Columbia University.

[40] Angelo Canty and Brian Ripley. *Boot: Bootstrap R Functions*, 2007. R package version 1.2-30.

[41] Brahim Chaib-Draa, Marc-André Labrie, Mathieu Bergeron, and Philippe Pasquier. Diagal: An agent communication language based on dialogue games and sustained by social commitments. *Autonomous Agents and Multi-Agent Systems*, 13(1):61–95, 2006.

[42] Walid Chainbi. Using the object paradigm to deal with the agent paradigm: capabilities and limits. pages 585–589, 2001.

[43] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis*. Wadsworth, 1983.

[44] K. M. Chandy, J. Misra, and L.M. Haas. Distributed deadlock detection. *ACM Transactions on Computer Systems*, 1:111–156, 1983.

[45] K. M. Chandy and Jayadev Misra. How processes learn. *Distrib. Comput.*, 1(1):40–52, 1986.

[46] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[47] G Chartrand. *Introductory Graph Theory*. Dover Publications, 1984.

[48] Yu chee Tseng. Detecting termination by weight-throwing in a faulty distributed system. *Journal of Parallel and Distributed Computing (JPDC*, 25:7–15, 1995.

[49] H. Chernoff and E. L. Lehmann. The use of maximum likelihood estimates in chi'2 tests for goodness of fit. *Annals of math. statistics*, pages 579–, 1954.

[50] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Coordination technologies for internet agents. *Nordic J. of Computing*, 6(3):215–240, 1999.

[51] P. R. Cohen and H. J. Levesque. Communicative actions for artificial agents. pages 65–72, June 1995.

[52] W.J Conover. *Practical Nonparametric Statistics*. Wiley, 1980.

[53] Rosaria Conte. *Social Order in Multiagent Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[54] Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. Autonomous norm acceptance. In *ATAL '98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pages 99–112, London, UK, 1999. Springer-Verlag.

[55] Thomas T. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*, chapter 14, pages 262–267. MIT Press, Cambridge, MA, USA, 1990.

[56] Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. Deriving petri nets from finite transition systems. *IEEE Trans. Comput.*, 47(8):859–882, 1998.

[57] R. Cost. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, Seattle, Washington, May 1999.

[58] R. S. Cost, T. Finin, and Y. Labrou. Jackal: A java-based tool for agent development. In *AAAI-98, Workshop on Tools for Agent Development*, Madison, WI, 1998.

[59] Rost S. Cost, Ye Chen, Tim Finin, Yannis Labrou, and Yun Peng. Using colored petri nets for conversation modeling. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 178–192. Springer-Verlag: Heidelberg, Germany, 2000.

[60] J. P. Courtiat, J. M. Ayache, and B. Algayres. Petri nets are good for protocols. *SIGCOMM Comput. Commun. Rev.*, 14(2):66–74, 1984.

[61] B. Cox, D. Tygar, and M. Sirbu. Netbill security and transaction protocol. 1995.

[62] F. Cristian, H. Aghali, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. In *Proc. 15th Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, pages 200–206, Ann Arbor, MI, USA, September 1985. IEEE Computer Society Press.

[63] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3), September 1989.

[64] Flaviu Cristian and Farnam Jahanian. A timestamp-based checkpointing protocol for long-lived distributed computations. In *Symposium on Reliability in Distributed Software*, pages 12–20, 1991.

[65] Ralph B. D'Agostino and Michael A. Stephens, editors. *Goodness-of-Fit Techniques*, volume 68 of *STATISTICS: Textbooks and Monographs*. Marcel Dekker, New York, 1986.

[66] Jonathan Dale, Bernard Burg, and Steven Willmott. The agentcities initiative: Connecting agents across the world. In Walt Truszkowski, Christopher Rouff, and Michael G. Hinchey, editors, *WRAC*, volume 2564 of *Lecture Notes in Computer Science*, pages 453–457. Springer, 2002.

[67] Jonathan Dale, Steven Willmott, and Bernard Burg. Agentcities: Challenges and deployment of next-generation service environments. In *Proc. Pacific Rim Intelligent Multi-Agent Systems*, 2002.

[68] Danthine. Protocol representation with finite state machines. *IEE transactions on communications*, 28(4):632–643, 1980.

[69] A.C Davison and D.V Hinkley. *Bootstrap Methods and Their Application*. Cambridge University Press, 1997.

[70] S. De, M. Sameeruddin, V. Sharma, N. Nandi, and H. Dutta. A new termination detection protocol for mobile distributed systems. In *Proc. 10th International Conference on Information Technology (ICIT 2007)*, pages 148–150, 17–20 Dec. 2007.

[71] Ronald F. DeMara, Yili Tseng, and Abdel Ejnioui. Tiered algorithm for distributed process quiescence and termination detection. volume 18, pages 1529–1538, November 2007.

[72] P. Dembinski and S. Budkowski. Specification language estelle. In M. Diaz, Jean-Pierre Ansart, Jean-Pierre Courtiat, P. Azema, and V. Chari, editors, *The formal description technique Estelle*, pages 35–75. North-Holland, 1989.

[73] Ralph Depke, Reiko Heckel, and Jochen Malte Küster. *Roles in Agent-Oriented Modeling*, volume 11. 2001.

[74] Thomas J. DiCiccio and Bradley Efron. Bootstrap confidence intervals. *Statistical Science*, 11(3):189–228, 1996.

[75] Frank Dignum. Agents, markets, institutions, and protocols. *Lecture Notes in Computer Science*, 1991:98–114, 2001.

[76] Frank Dignum and Mark Greaves. Issues in agent communication: An introduction. In *Issues in Agent Communication*, pages 1–16, London, UK, 2000. Springer-Verlag.

[77] Edsger W. Dijkstra, W. H. J. Feijen, and A. J. M. van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16(5):217–219, 1983.

[78] E.W. Dijkstra and C.S Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, August 1980.

[79] Dolev, Halpern, and Strong. On the possibility and impossibility of achieving clock synchronization. *JCSS: Journal of Computer and System Sciences*, 32, 1986.

[80] Edmund H. Durfee. Scaling up agent coordination strategies. *Computer*, 34(7):39–46, 2001.

[81] B. Efron. Another look at the jacknife. *Annals of Statistics*, 7(1):1–26, 1979.

[82] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1(1):54–75, 1986.

[83] B. Efron and R Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1994.

[84] Bradley Efron. Better bootstrap confidence intervals. *Journal of the American Statistical Association*, 82(397):171–185, 1987.

[85] Bradley Efron. Jackknife-after-bootstrap standard errors and influence functions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 54:83–127, 1992.

[86] Amal El Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. Protocol engineering for multi-agent interaction. *Lecture Notes in Computer Science*, 1647:89–101, 1999.

[87] Marc Esteva. *Electronic institutions. from specification to development*. PhD thesis, Universitat Politecnica de Catalunya, 2003.

[88] Shaheen S. Fatima, Michael Wooldridge, and Nicholas R. Jennings. Multi-issue negotiation under time constraints. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 143–150, New York, NY, USA, 2002. ACM.

[89] W. Feller. *Introduction to Probability Theory and Its Applications*, volume I. Wiley, New York, 3rd edition, 1968.

[90] James Filliben. Dataplot–an interactive high-level, language for graphics non-linear fitting, data analysis, and mathematics. In *Proceedings of the Third Annual Conference of the National Computer Graphics Association*, Anaheim, CA, 1982.

[91] James J. Filliben. The probability plot correlation coefficient test for normality. *Technometrics*, 17:111–117, 1975.

[92] Nissim Francez. Distributed termination. *ACM Trans. Program. Lang. Syst.*, 2(1):42–55, 1980.

[93] Les Gasser. Perspectives on organizations in multi-agent systems. *Lecture Notes in Computer Science*, 2086:1–16, 2001.

[94] Alan Gibbons and Wojciech Rytter. *Efficient parallel algorithms*. Number 0521388414. Cambridge University Press; Reprint edition, 1989.

[95] J.D Gibbons. *Nonparametric Statistical Inference*. Marcel Decker, New York, 2004.

[96] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics. North Holland, Feb 2004.

[97] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, chapter 2, pages 22–45. Annals of Discrete Mathematics. North Holland, Feb 2004.

[98] Mark Greaves, Heather Holmback, and Jeffrey Bradshaw. What is a conversation policy? pages 118–131, 2000.

[99] Zahia Guessoum and Jean-Pierre Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, July-September 1999. Special series on Actors and Agents, edited by Dennis Kafura and Jean-Pierre Briot.

[100] Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *SFCS '78: Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 8–21, Washington, DC, USA, 1978. IEEE Computer Society.

[101] Gery Gutnik. *Monitoring large-scale multi-agent systems using overhearing*. PhD thesis, Bar Ilan University, 2006.

[102] Gery Gutnik and Gal A. Kaminka. A scalable petri net representation of interaction protocols for overhearing. In *AAMAS*, pages 1246–1247. IEEE Computer Society, 2004.

[103] Gery Gutnik and Gal A. Kaminka. A scalable petri net representation of interaction protocols for overhearing. pages 1246–1247, 2004.

[104] Peter Hall. On the bootstrap and confidence intervals. *The Annals of Statistics*, 14(4):1431–1452, December 1986.

[105] Peter Hall. *The Bootstrap and Edgeworth Expansion*. Springer Verlag, New York, 1992.

[106] J.Y Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant clock synchronisation. In *Proceedings of the 3rd ACM Symposium on Princi- ples of Distributed Computin*, pages 89–102, Vancouver, August 1984. ACM Press.

[107] R. Hill and J. Arbaugh P. Cartwright, A. Nielsen. Jackknifing the bootstrap: some monte carlo evidence. *Communications in Statistics : Simulation and Computation*, 26:125–139, 1997.

[108] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[109] C.A.R. Hoare. *Communicating Sequential Processes*. Number 0-13-153271-5. Prentice Hall International Series in Computer Science, 1985.

[110] M Hollander and D. A. Wolfe. *Nonparametric Statistical Method*. Wiley, 1973.

[111] G.J Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall Englewood Cliffs, New Jersey 07632, Bell Laboratories, 1991.

[112] G.J. Holzmann. The model checker spin. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997.

[113] G.J. Holzmann. *SPIN Model Checker, The: Primer and Reference Manual*. Addison Wesley Professional, 2004.

[114] J. Hopcraft and J Ullman. *Introduction to automata theory , languages and computation*. Addison Wesley, 1979.

[115] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[116] S. T. Huang. Termination detection by using distributed snapshots. *Inf. Process. Lett.*, 32(3):113–120, 1989.

[117] Shing-Tsaan Huang. Detecting termination of distributed computations by external agents. In *Proceedings of the 9th International Conference on Distributed Computing Systems (ICDCS)*, pages 79–84, Washington, DC, 1989. IEEE Computer Society.

[118] Shing-Tsaan Huang and Pei-Wen Kao. Detecting termination of distributed computations by external agents. *Information Science and Engineering*, 7:187–201, 1991.

[119] P.J Huber. *Robust Statistics*. John Wiley and Sons, New York, 1981.

[120] Marc-Philippe Huget and Jean-Luc Koning. Interaction protocol engineering. In Marc-Philippe Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2003.

[121] Marc-Philippe Huget and Jean-Luc Koning. Requirement analysis for interaction protocols. In *CEEMAS*, pages 404–412, 2003.

[122] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[123] M. Sriram Iyengar and Mukesh Singhal. Effect of network latency on load sharing in distributed systems. *J. Parallel Distrib. Comput.*, 66(6):839–853, 2006.

[124] Ivar Jacobson, Grady Booch, and Jim Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[125] C. Jarque and A. Bera. Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Econometric Letters*, 6:255–259, 1980.

[126] N. R. Jennings. Coordination techniques for distributed artificial intelligence. pages 187–210, 1996.

[127] N. R. Jennings and M. Wooldridge. *Applications of intelligent agents*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[128] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

[129] C. Johnson. What is research in computing science, 2000.

[130] Richard Jones and Rafael Lins. *Garbage Collection:Algorithms for Automatic Dynamic Memory Management*. John Wileys and Sons, 1996.

[131] Karen Kafadar. *Testing for Normality. Henry C. Thode, Jr*, volume 98. January 2003.

[132] Elizabeth A. Kendall. Agent roles and aspects. In *ECOOP '98: Workshop ion on Object-Oriented Technology*, page 440, London, UK, 1998. Springer-Verlag.

[133] Paul Klemperer. *Auctions: Theory and Practice The Toulouse Lectures in Economics*. Princeton University Press, 2004.

[134] Donald E. Knuth. *Fundamental Algorithms*, volume 1. Addison Wesly, 3 edition, 1997.

[135] Mamadou Tadiou Kone, Akira Shimazu, and Tatsuo Nakajima. The state of the art in agent communication languages. *Knowl. Inf. Syst.*, 2(3):259–284, 2000.

[136] Jean-Luc Koning, Marc-Philippe Huget, Jun Wei, and Xu Wang. Extended modeling languages for interaction protocol design. In *Proc. of Agent-Oriented Software Engineering (AOSE) 2001, Agents 2001*, 2001.

[137] H Kopetz and W. Ochsenreiter. Clock synchronisation in distributed real-time systems. *IEEE Transactions on Computers,*, 36(8):933–940, August 1987.

[138] William H Kruskal and Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, December 1952.

[139] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings, 1993.

[140] Y. Labrou and T. Finin. Semantics for an agent communication language. In M. Wooldridge and N. R. Jennings, editors, *Fourth International Workshop on Agent Theories, Architectures, and Languages*, pages 199–203, Providence, Rhode Island, USA, 1996. Springer Verlag.

[141] Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, / 1999.

[142] Richard Lai and Ajin Jirachiefpattana. *Communication protocol specification and verification*. Springer, 1998.

[143] T. H. Lai, Y. C. Tseng, and X. Dong. A more efficient message-optimal algorithm for distributed termination detection. In *Proc. Sixth International Parallel Processing Symposium*, pages 646–649, 23–26 March 1992.

[144] Ten-Hwang Lai and Li-Fen Wu. An (n -1)-resilient algorithm for distributed termination detection. *IEEE Trans. Parallel Distrib. Syst.*, 6(1):63–78, 1995.

[145] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. In *Advances in Ultra-Dependable Distributed Systems, N. Suri, C. J. Walter, and M. M. Hugue (Eds.), IEEE Computer Society Press*. IEEE Computer Society Press, 1995.

[146] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *ommunications of the ACM*, 21(7):558–565, 1978.

[147] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.

[148] Leslie Lamport and P M Melliar Smith. Byzantine clock synchronization. *SIGOPS Oper. Syst. Rev.*, 20(3):10–16, 1986.

[149] H.W Lilliefors. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62:339–402, 1967.

[150] Michael Winikoff Lin Padgham. *Developing intelligent agent systems*. John Wiley and Sons Ltd, 2004.

[151] Fabiola López y. López and Michael Luck. Modelling norms for autonomous agents. In *ENC '03: Proceedings of the 4th Mexican International Conference on Computer Science*, page 238, Washington, DC, USA, 2003. IEEE Computer Society.

[152] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 75–88, Vancouver, August 1984.

[153] Nancy Lynch. *Distributed Algorithms*, chapter 2, pages 21–22. Morgan Kaufmann, 1996.

[154] Benedita Malheiro and Eugenio Oliveira. Argumentation as distributed belief revision: Conflict resolution in decentralised co-operative multi-agent systems. In *EPIA '01: Proceedings of the10th Portuguese Conference on Artificial Intelligence on Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving*, pages 205–218, London, UK, 2001. Springer-Verlag.

[155] H.B Mann and D.R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60, 1947.

[156] Wendy L. Martinez. *Computational Statistics Handbook with MATLAB*. Chapman & Hall/CRC Press, 2002.

[157] K.A Marzullo. *Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service*. PhD thesis, Stanford University, Department of Electrical Engineering, 1984.

[158] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. *SIGOPS Oper. Syst. Rev.*, 19(3):44–54, 1985.

[159] Jeff Matocha and Tracy Camp. A taxonomy of distributed termination detection algorithms. *J. Syst. Softw.*, 43(3):207–221, 1998.

[160] F. Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2(3):161–175, 1987.

[161] F. Mattern. Global quiescence detection based on credit distribution and recovery. *Information Processing Letters*, 30(4):195–200, 1989.

[162] N. Maudet and B. Chaib-Draa. Commitment-based and dialogue-game-based protocols: new trends in agent communication languages. *Knowl. Eng. Rev.*, 17(2):157–179, 2002.

[163] W.S. McCulloch and W. Pitts. A logical calculus of the ideas of immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[164] P. Merlin. Specification and validation of protocols. *IEEE Transactions on Communication*, 27(11):1671– 1680, 1979.

[165] D. L Mills. Network time protocol version 4 reference and implementation guide. Technical Report 06-6-1, University of Delaware, June 2006.

[166] David L Mills. *Computer Network Time Synchronization: the Network Time Protocol*. Crc, 2006.

[167] R Milner. *Communication and Concurrency*. Pentice- Hall, 1989.

[168] Jayadev Misra. Detecting termination of distributed computations using markers. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 290–294, New York, NY, USA, 1983. ACM.

[169] Neeraj Mittal, Felix C. Freiling, S. Venkatesan, and Lucia Draque Pensor. On termination detection in crash-prone distributed systems with failure detectors. *J. Parallel Distrib. Comput.*, 68(6):855–875, 2008.

[170] C.Z. Mooney and R.D. Duval. *Bootstrapping. A Nonparametric Approach to Statistical Inference*. SAGE, 1993.

[171] T. Motshegwa, M. Schroeder, R. Kloos, P. Noy, and J. Gomuloch. Control and management of agents and their services. Technical Report COMAS-2002-7-1, Department of Computing, City University, London, United Kindom, 2002.

[172] Tshiamo Motshegwa. Bootstrapping detection delays : An evaluation of a distributed termination detection protocol for agents. Forthcoming, Applied Computing.

[173] Tshiamo Motshegwa. Conversation model for distributed termination detection of multiagent interactions.

[174] Tshiamo Motshegwa and Michael Schroeder. Interaction monitoring and termination detection for agent societies: Preliminary results. In Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors, *ESAW*, volume 3071 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2003.

[175] Bernard Moulin and Brahim Chaib-draa. An overview of distributed artificial intelligence. pages 3–55, 1996.

[176] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[177] M. Naimi. Global stability detection in the asynchronous distributed computations. In *Proc. Workshop on the Future Trends of Distributed Computing Systems in the 1990s*, pages 87–92, 14–16 Sept. 1988.

[178] N. Nataran. A distributed scheme for detecting communication deadlocks. *IEEE Transactions on Software Engineering*, 12:531–537, 1986.

[179] Matthias Nickles, Michael Rovatsos, and Gerhard Weib. *Agents And Computational Autonomy: Potential, Risks, And Solutions*. Lecture Notes in Computer Science. Springer Verlag, 2004.

[180] Matthias Nickles, Michael Rovatsos, and Gerhard Weib. *Agents And Computational Autonomy: Potential, Risks, And Solutions*. Lecture Notes in Computer Science. Springer Verlag, 2004.

[181] NIST/SEMATEC. *NIST/SEMATECH eHandbook of Statistical Methods*. Nist, 2004.

[182] Bill Nitzberg and Virginia Lo. Distributed shared memory: A survey of issues and algorithms. *Computer*, 24(8):52–60, 1991.

[183] David G. Novick and Karen Ward. Mutual beliefs of multiple conversants: A computational model of collaboration in air traffic control. In *AAAI*, pages 196–201, 1993.

[184] Mariusz Nowostawski and Martin Purvis. The concept of autonomy in distributed computation and multi-agent systems. In *IAT ’07: Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 420–423, Washington, DC, USA, 2007. IEEE Computer Society.

[185] Lin Padgham, John Thangarajah, and Michael Winikoff. Auml protocols and code generation in the prometheus design tool. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *AAMAS*, page 270. IFAAMAS, 2007.

[186] G. Papp. Using petri nets for modelling of finite state machines and protocols. *Hiradastechnika*, 41(12):301–311, 1990. NewsletterInfo: 39.

[187] S. Paurobally and Jim Cunningham. Achieving common interaction protocols in open agent environments. In *Working Notes of Challenges in Open Agent Systems ’03 Workshop*, Melbourne, Australia, July 2003.

[188] J.L. Peterson. Petri nets. *ACM Surveys*, 9(3):223 – 252, 1977.

[189] C.A. Petri. *Communications with Automata*. PhD thesis, Rome Air Development Center, Rome, NY, 1966.

[190] Jeremy Pitt and Abe Mamdani. Communication protocols in multi-agent systems: A development method and reference architecture. In *Issues in Agent Communication*, pages 160–177, London, UK, 2000. Springer-Verlag.

[191] J.V. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In *Proceedings 16th International Joint Conference on Artificial Intelligence IJCAI'99*, pages 486–491, Stockholm, Sweden, 1999. Morgan-Kaufmann Publishers.

[192] R.L Plackett. Karl pearson and the chi-squared test. *International Statistical Review*, 51:59–72, 1983.

[193] J. Postel. User datagram protocol. Technical report, ISI, United States, 1980.

[194] Michael J. Quinn and Narsingh Deo. Parallel graph algorithms. *ACM Comput. Surv.*, 16(3):319–348, 1984.

[195] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, 1990.

[196] A. S. Rao and M. P. Georgeff. Bdi-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[197] Brian D. Ripley. The R project in statistical computing. *MSOR Connections. The newsletter of the LTSN Maths, Stats & OR Network.*, 1(1):23–25, February 2001.

[198] J.S Rosenschein and G.Zlotkin. *Rules of Encounter:Designing conventions for automated negotiation among Computers*. MIT Press, Boston, 1994.

[199] K Sabnani. An algorithmic technique for protocol verification. *IEEE Transactions on Communication*, 8:924–931, August 1988.

[200] Kassem Saleh. Synthesis of communications protocols: an annotated bibliography. *SIGCOMM Comput. Commun. Rev.*, 26(5):40–59, 1996.

[201] Michael Schroeder. An efficient argumentation framework for negtiating autonomous agents. In *MAAMAW '99: Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 140–149, London, UK, 1999. Springer-Verlag.

[202] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, January 1970.

[203] Sandip Sen and Gerhard Weiss. Learning in multiagent systems. pages 259–298, 1999.

[204] J. Shao and D. Tu. *The Jackknife and Bootstrap.* Springer Verlag, 1995.

[205] S.S Shapiro and M.B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52, 3,4:591–6, 1965.

[206] N. Shavit and N Franchez. A new approach to the detection of locally indicative stability. In L. Kott, editor, *Intl. Colloq. Automata, Languages, and Programming*, volume 226, pages 344–358, Washington, DC, 1986. Lecture notes Computer Science, Springer Verlag.

[207] David Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Number 1584884401. CRC Press, 2003.

[208] Kang G. Shin and P. Ramanathan. Clock synchronization of a large multiprocessor system in the presence of malicious faults. *IEEE Trans. Comput.*, 36(1):2–12, 1987.

[209] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence.*, 60(1):51–92, 1993.

[210] Yoav Shoham and Kevin Leyton-Brown. *Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[211] J. L. Simon. *Resampling: The New Statistics*. 1997.

[212] M. Singh. On the semantics of protocols among distributed intelligent agents. In *IEEE International Phoenix Conference on Computers and Communications*, pages 379–386, Phoenix, Arizona, April 1992. IEEE.

[213] Munindar P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer Verlag: Heidelberg, Germany, 2000.

[214] Mark Smith. Formal verification of communication protocol. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX: Theory, Applications, and Tools FORTE/PSTV'96: Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification*, pages 129–144, Kaiserslautern, Germany, 1996. Chapman & Hall.

[215] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *Proceedings of the 1st International Conference on Distributed Computing Systems*, pages 186–192, Washington, DC, 1979. IEEE Computer Society.

[216] P. Sprent. *Applied nonparametric statistical methods*. Chapman and Hall, London, 2nd edition, 1993.

[217] P. Sprent. *Data driven statistical methods*. Chapman and Hall, London, 1998.

[218] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. In *Symposium on Principles of Distributed Computing*, pages 71–86, 1985.

[219] R. G Staude and S. J Sheather. *Robust Estimation*. John Wiley and Sons, New York, 1990.

[220] Larry M. Stephens and Michael N. Huhns. Multiagent systems and societies of agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 79–120. The MIT Press, Cambridge, MA, USA, 1998.

[221] M.A Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, pages 730–737, 1974.

[222] MA. Stephens. *Goodness-of-Fit Techniques*, chapter Tests based on EDF statistics. Marcel Dekker, 1986.

[223] Carl A. Sunshine. Survey of protocol definition and verification techniques. *SIGCOMM Comput. Commun. Rev.*, 8(3):35–41, 1978.

[224] Milind Tambe and Hyuckchul Jung. Towards conflict resolution in agent teams via argumentation, April 22 2000.

[225] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.

[226] Gerard Tel. *Introduction To Distributed Algorithms*, chapter 13. Cambridge university Press, 2000.

[227] Gerard Tel and Friedemann Mattern. The derivation of distributed termination detection algorithms from garbage collection schemes. *ACM Transactions on Programming Languages and Systems*, 15(1):1–35, January 1993.

[228] Gerard Tel and J. van Leeuwen. The derivation of graph marking algorithms distributed termination detection protocols. *Science of Computer Programming*, 10:107–137, 1988.

[229] Thorsten Thadewald and Herbert Buumlning. Jarque-bera test and its competitors for testing normality - a power comparison. *Journal of Applied Statistics*, 34(1):87–105, 2007.

[230] John Thangarajah, Lin Padgham, and Michael Winikoff. Prometheus design tool. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 127–128, New York, NY, USA, 2005. ACM.

[231] Rodney W. Topor. Termination detection for distributed computations. *Information Processing Letters*, 18(1):33–36, 1984.

[232] T. E. Truman. *A Methodology for the Design and Implementation of Communication Protocols for Embedded Wireless Systems*. PhD thesis, UNIVERSITY OF CALIFORNIA, BERKELEY, 1998.

[233] Yu-Chee Tseng and Cheng-Chung Tan. Termination detection protocols for mobile distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):558–566, 2001.

[234] J Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[235] N. Vasanthavada and P.N. Marino. Synchronisation of fault-tolerant clocks in the presence of malicious failures. *IEEE Transactions on Computers*, 37(4):440–448, 1988.

[236] Gregor von Bochmann. Finite state description of communication protocols. *Computer Networks*, 2:361–372, 1978.

[237] Jin Wang, J. Sunil Rao, and Jun Shao. Weighted jackknife-after-bootstrap: a heuristic approach. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 240–245, Washington, DC, USA, 1997. IEEE Computer Society.

[238] Xinli Wang and Jean Mayo. A general model for detecting distributed termination in dynamic systems. *Parallel and Distributed Processing Symposium, International*, 1:84b, 2004.

[239] Gerhard Weiss. Agent orientation in software engineering. volume 16, pages 349–373, New York, NY, USA, 2001. Cambridge University Press.

[240] Gerhard Weiß, Michael Rovatsos, and Matthias Nickles. Capturing agent autonomy in roles and xml. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 105–112, New York, NY, USA, 2003. ACM.

[241] Michael Wellman William and William E. Walsh. Distributed quiescence detection in multiagent negotiation. In *ICMAS '00: Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-2000)*, page 317, Washington, DC, USA, 2000. IEEE Computer Society.

[242] Steven Willmott, Simon G. Thompson, David Bonnefoy, Patricia Charlton, Ion Constantinescu, Jonathan Dale, and Tianning Zhang. Agent based dynamic service synthesis in large-scale open environments: Experiences from the agentcities testbed. In *AAMAS*, pages 1318–1319. IEEE Computer Society, 2004.

[243] Michael Winikoff. Designing commitment-based agent interactions. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 363–370, Washington, DC, USA, 2006. IEEE Computer Society.

[244] Michael Winikoff. Implementing commitment-based interactions. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[245] Michael Winikoff, Wei Liu, and James Harland. Enhancing commitment machines. pages 198–220, 2004.

[246] Michael Wooldridge. Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000.

[247] Michael Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, 2001.

[248] Michael Wooldridge and Nicholas R. Jennings. Pitfalls of agent-oriented development. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 385–391, New York, NY, USA, 1998. ACM.

[249] Michael J. Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: A survey. pages 1–39. Springer-Verlag, 1994.

[250] Xinfeng Ye and John A. Keane. Token scheme: An algorithm for distributed termination detection and its proof of correctness. In *IFIP Congress (1)*, pages 357–364, 1992.

[251] Pinar Yolum and Munindar P. Singh. Synthesizing finite state machines for communication protocols. Technical report, Raleigh, NC, USA, 2001.

[252] Pinar Yolum and Munindar P. Singh. Commitment machines. In *ATAL 01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, pages 235–247, London, UK, 2002. Springer-Verlag.

[253] Franco Zambonelli and Andrea Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.

PART IV
APPENDICIES

APPENDIX A

Termination Detection for protocols

A.1 Illustration for the diffusing conversations algorithm

Consider Figure A.1 showing the execution trace of Algorithm 4 discussed in section 5.4 in page 93. This trace illustrates that the algorithm traverses the tree in breadth-first and at each node evaluating whether there are any descendants conversation nodes i.e. evaluating if $R_{transitive} = \emptyset$, testing for if the protocol is active using $activeProtocol$ predicate and removing that node with a procedure $remove$ if the above is true. When G eventually becomes empty, the associated flag F (defined in Definition 10 in page 84) of a conversation can be set.

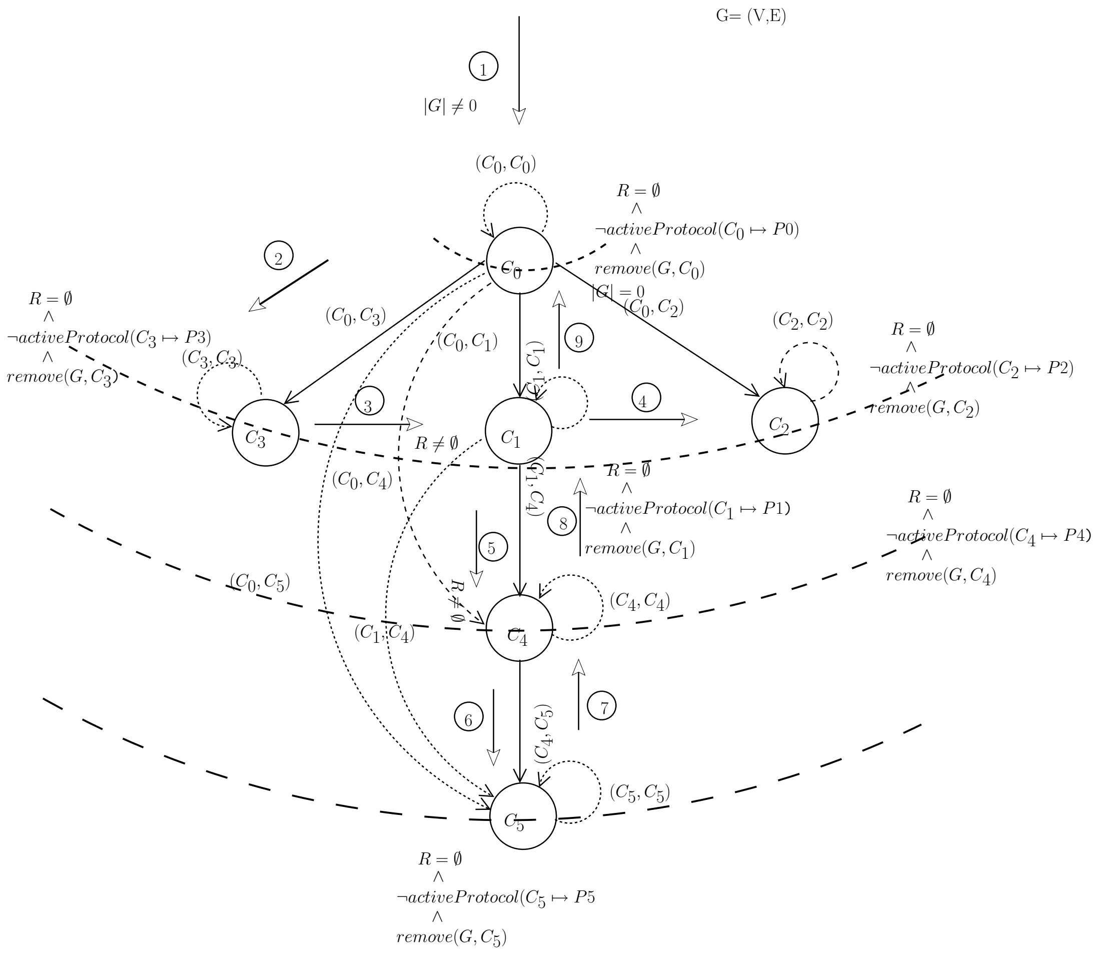

G= (V,E)
1
$|G| \neq 0$
(C_0, C_0)
$R = \emptyset \wedge \neg activeProtocol(C_0 \mapsto P0) \wedge remove(G, C_0)$
$|G| = 0$
2
C_0
(C_0, C_3)
(C_0, C_2)
(C_2, C_2)
$R = \emptyset \wedge \neg activeProtocol(C_3 \mapsto P3) \wedge remove(G, C_3)$
(C_3, C_3)
(C_0, C_1)
(C_1, C_1)
9
$R = \emptyset \wedge \neg activeProtocol(C_2 \mapsto P2) \wedge remove(G, C_2)$
3
4
C_3
C_1
C_2
$R \neq \emptyset$
(C_0, C_4)
(C_1, C_4)
$R = \emptyset \wedge \neg activeProtocol(C_1 \mapsto P1) \wedge remove(G, C_1)$
8
5
$R = \emptyset \wedge \neg activeProtocol(C_4 \mapsto P4) \wedge remove(G, C_4)$
(C_0, C_5)
$R \neq \emptyset$
(C_4, C_4)
(C_1, C_4)
C_4
6
7
(C_4, C_5)
(C_5, C_5)
C_5
$R = \emptyset \wedge \neg activeProtocol(C_5 \mapsto P5 \wedge remove(G, C_5)$

Figure A.1: Showing example trace of executing Algorithm 4 of a diffusing computation tree for a conversation $G = (V, E)$ rooted at C_0.

A.2 Other possible implementations of a global c-matrix

There are various other possibilities for logically implementing the a global $c-matrix$ $\mathbf{M}$, namely;

1. Consider a set of controllers coordinating via a *tuple space*. The tuple space functioning as a *shared memory* for global c-matrix structure allowing overlap over controllers, i.e. allowing $\bigcap_{n\in\mathbb{N}} C_n\left(\mathbf{M}_{i,j}\right) \neq \emptyset$, and providing an update protocol for the tuple space by controllers, e.g. Figure A.2.

2. Consider a combination of 2 and 3, i.e. divide $\mathbf{M}$ logically by allowing each controller to manage a separate copy of a $c - matrix$ and allow overlaps, and consider controllers coordinating via a *tuple space* that functions as a *shared cache* for the global c-matrix structure, e.g. Figure A.3.

Figures A.2 and A.3 next illustrate configurations one and two respectively as discussed above.

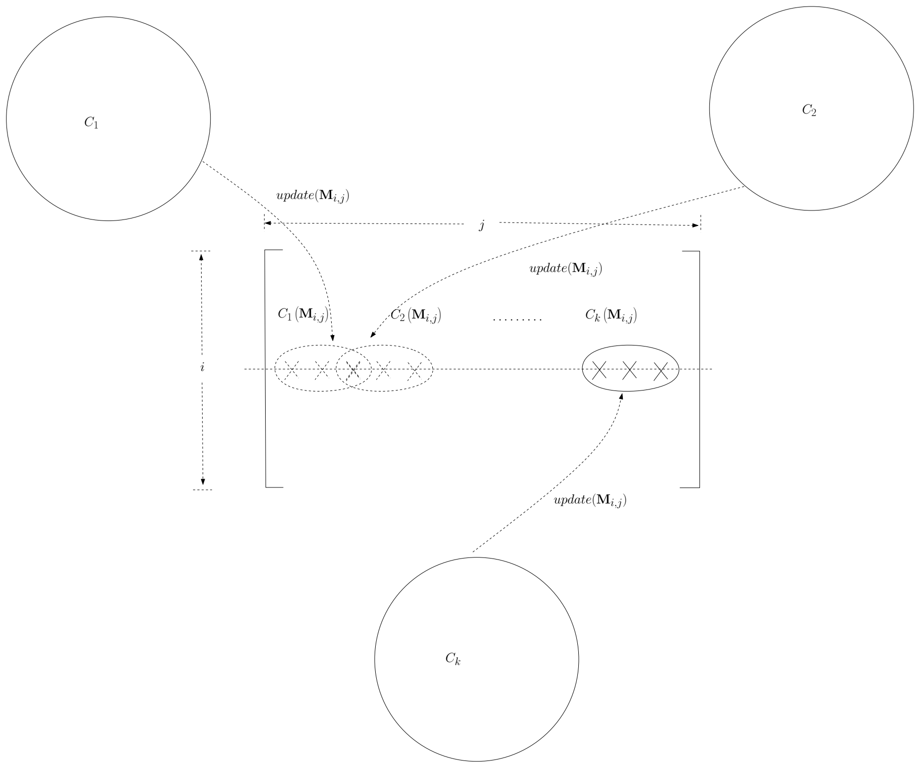


Figure A.2: Showing example global matrix allocated across controllers allowing overlaps, i.e. $\bigcap_{n\in\mathbb{N}} C_n\left(\boldsymbol{M}_{i,j}\right) \neq \emptyset$
using a tuple space

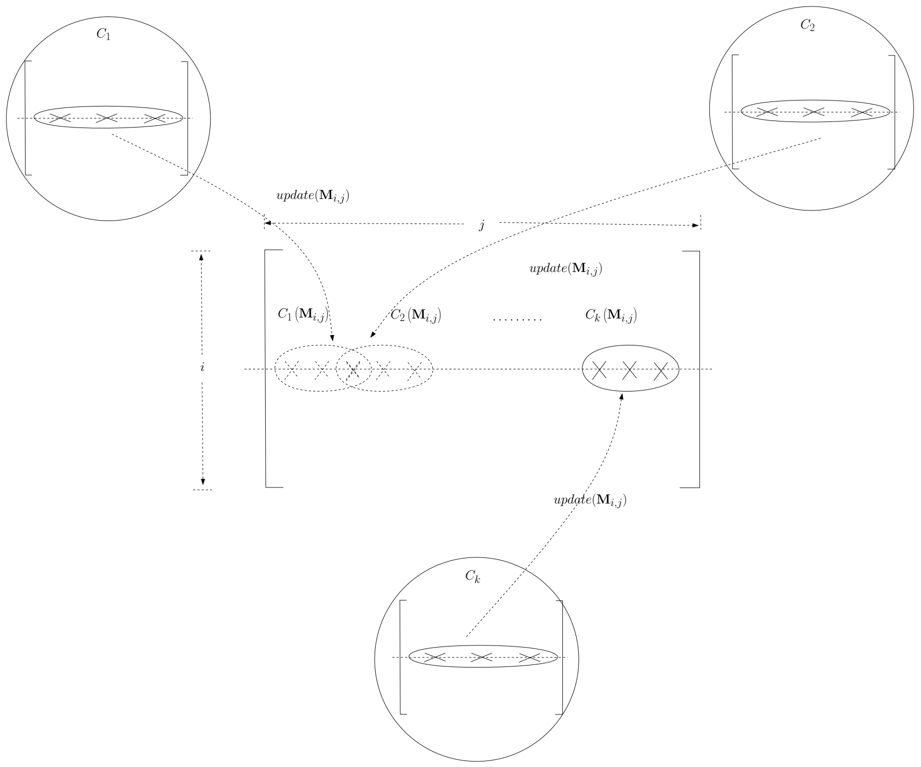


Figure A.3: Showing example global matrix allocated across controllers allowing overlaps, i.e. $\bigcap_{n\in\mathbb{N}} C_n\left(\boldsymbol{M}_{i,j}\right) \neq \emptyset$
, using local $c - matrix$ structures and tuple as cache

A.3 Algorithms

Considering setup 3, $\mathbf{M}$ in the tuple space is treated as a local matrix by controllers in $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$ with the $update$ protocol providing a concurrent[1] write mechanism to the structure. In this setup then, the procedure for global termination involves a test of the $localtermination$ predicate on $\mathbf{M}$ in the tuple space.

In setup 4, the update protocol can be extended to provide a mechanism to replicate entries of local $c-matrix$ structures held by controllers on $\mathbf{M}$ in the tuple space in a cache update[2]. Therefore in this setup the procedure for global termination involves a cache update and a test of the $localtermination$ on the cache. Consider Algorithm 8 that implements this scheme.

[1]Though concurrency control is not strictly necessary if we introduce a constraint that even though $\bigcap_{n\in\mathbb{N}} C_n \left(\mathbf{M}_{i,j}\right) \neq \emptyset$ may hold, controllers $\mathcal{C}_2$ and $\mathcal{C}_2$ say, will work with different conversations in the vector $\langle C_k \rangle$.

[2]The update can be periodic or triggered by a designated controller $\mathcal{C}_0$ with a $sync$ signal say.

Algorithm 8 Algorithm for synchronisation of cache updates across controllers in $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$, with global termination reported by controller $\mathcal{C}_0$ for agent i

```
procedure $\underline{globaltermination}$ ()

INPUT:              - $\mathbf{M}$ in the tuple space;

DATA STRUCTURES:              local $c - matrix$ $\mathbf{m}$; $\mathcal{C} = \{\mathcal{C}_n \mid n \in \mathbb{N}\}$

for all $\mathcal{C}_n \in \mathcal{C}$ do
   if $\mathcal{C}_n \neq \mathcal{C}_0$ then
      $sync(\mathcal{C}_n)$
   else
      $cacheupdate(\mathbf{M}, \mathbf{m})$
      $F \longleftarrow localterminantion\ (\mathbf{M}, i)$
   end if
end for

procedure $\underline{cacheupdate}$ ()

INPUT:              - $\mathbf{M}$ $c - matrix$ in the tuple space; $\mathbf{m}$ local $c - matrix$
for all $m_{i,j}$ do
   $(\mathbf{M}_{i,j} \mapsto F) \longleftarrow \bigwedge_{1 \leq k \leq n} inactive(\mathbf{m}_{i,j} \mapsto C_k)$
   $append(\mathbf{m}_{i,j} \mapsto \langle C_k \rangle, \mathbf{M}_{i,j} \mapsto \langle C_k \rangle)$
end for
return $(F)$
```

A.4 Discussion and complexity analysis

Algorithms discussed in Chapter 5 primarily worked on graphs and used set operations extensively. For graphs, complexity in primarily influenced by the number of nodes and branching factor.

Consider the graph traversal algorithm that performs a *reachability analysis* (Algorithm 2 in page 81) invoked by Algorithm 1, stp, page 80 which computes shortest unique termination paths on a given protocol graph.

The *update* procedure, Algorithm 3 in page 82 is of order $\mathcal{O}(n \times K \times m)$, where n is the number of elements of TP, K is the number of elements of A and m is the number of elements of A_k

But as remarked, the procedure for computing shortest unique termination paths stp can be performed off-line, and is only performed once given a protocol and can be performed only once on a set of protocols, $\mathcal{P}$, a protocol library if one exists. Furthermore, the interaction protocol graphs for agents are invariably not very large, e.g. the *contract-net* protocol given as an example in section 5.3, page 70 or consider the FIPA agent interaction protocol suite. For large protocol graphs, there exist a lot of work in the area of parallel graph algorithms [13, 194, 94] or [7] that can be explored in future work.

Regarding set operations used in these algorithms, much will depend on their implementation and data structures used. [55] explains that a binary search tree of height h can implement any of the basic dynamic set operations [3] in $\mathcal{O}(h)$ time. This is clearly reasonable for small graphs (small h), and performance maybe no better than with a linked list[55] if height h is large.

Again most of the protocol graphs are small. For large graphs, Red-Black trees [100]

[3]SEARCH, INSERT,DELETE, MAXIMUM,MINIMUM,SUCCESSOR, PREDECESSOR.

that guarantee that basic dynamic-set operations take $\mathcal{O}(log\ \ n)$ in the worst case [55] can be used. Regarding implementation, there are programming languages libraries that implement container data structures and efficient operations on them.

In section 5.4, page 85, we modelled conversations as diffusing computation trees, however, we can add another conversation scenario to the scenarios in section 5.4.1 page 83 and extend the conversation model to model some dependencies between conversations involving a particular agent i and different agents j and k. For example, see Figure A.4 where completion of $C_{0,1}$ depends on $C_{1,1,1}$, and in that case the diffusing computation for the conversation is no longer a tree but a general graph.

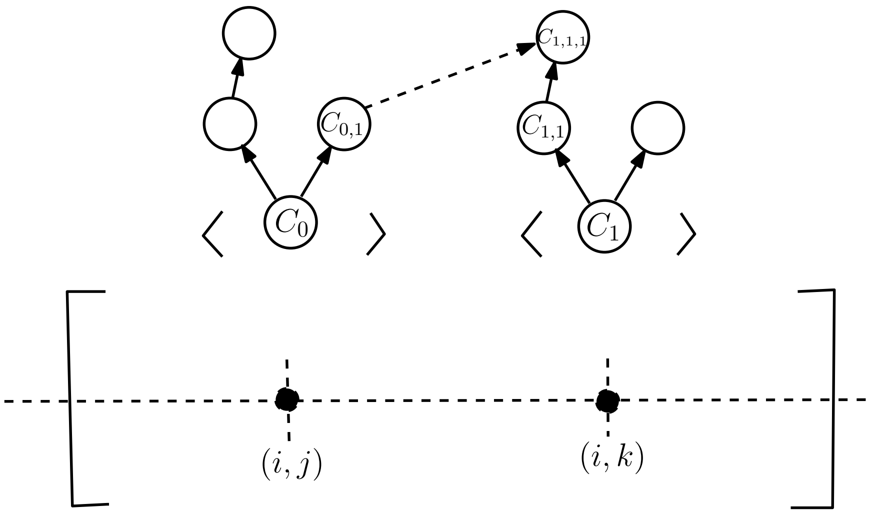


Figure A.4: Showing an extended model for conversations, with dependencies between computation trees

Computational complexities of procedures on graph structures are well known and briefly discussed next and summarised in Table A.1 below.

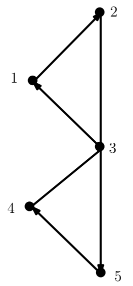
In general, a common way of representing graphs as data structures is to consider an adjacency matrix [47], and its representational data structures. An analysis of the complexity issues in algorithmic graph theory is given in [96] and summarised in Appendix J, page 372 and we cite the analysis there.

i.e. Let $G = (V, E)$ be a graph whose vertices have been (arbitrarily) ordered $v_1, v_2, \ldots v_n$. The adjacency matrix **(**M) = $(m_{i,j})$ of G is an $n \times n$ matrix with entries

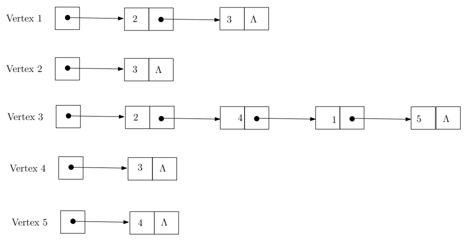
$$m_{i,j} = \begin{cases} 0 & if \quad v_i v_j \notin E \\ 1 & if \quad v_i v_j \in E \end{cases}$$

for example consider Figure A.5, then the adjacency matrix **M** is given by

$$m_{i,j} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$



(a) G



(b) adjacency lists of G

Figure A.5: G

and can be represented as a an adjacency list [4] of G given in Figure J.1 (b).

[4]Regarding adjacency lists, for each vertex v_i of G an adjacency list $adj\,(v_i)$ can be created, containing those vertices adjacent to v_j.

	Adjacency matrix stored as an array	Adjacency sets stored as lists	Adjacency sets stored sequentially
Is v_iv_j an edge?	$\mathcal{O}(1)$	$\mathcal{O}(d_i)^*$	$\mathcal{O}(d_i)^*$
Mark each vertex which is adjacent to v_i	$\mathcal{O}(n)$	$\mathcal{O}(d_i)$	$\mathcal{O}(d_i)$
Mark each edge	$\mathcal{O}(n^2)$	$\mathcal{O}(e)$	$\mathcal{O}(e)$
Add an edge v_iv_j	$\mathcal{O}(1)$	$\mathcal{O}(1)^{**}$	$\mathcal{O}(e)$
Erase an edge v_iv_j	$\mathcal{O}(1)$	$\mathcal{O}(d_i^*)$	$\mathcal{O}(e)$

Table A.1: Some typical graph operations and their complexity with respect to three data structures. If the adjacency sets are sorted then the starred entries can be reduced to $\mathcal{O}(log\ \ d_i)$ using a binary search, but the double starred entry will increase to $\mathcal{O}(d_i)$ [97]

[96] reasons that, by definition, the main diagonal of M is all zeros, and M is symmetric about the main diagonal if and only if G is an undirected graph [5] and if M is stored as a 2-dimensional array, then only one step (more precisely $\mathcal{O}(1)$ time) is required for the statements "is $v_iv_j \in E$? or "erase the edge v_iv_j. An instruction such as "mark each vertex which is adjacent to v_j" requires scanning the entire column j and hence takes n steps. Similarly, "mark each edge" takes n^2 steps. The space requirement for the array representation is $\mathcal{O}(n^2)$.

Table A.1 above discussed in [97] and in Appendix J in page 372 show some typical graph operations and their complexity with respect to three data structures, where n is the number of vertices, e is the number of edges , d_i is the degree of vertex v_i.

Regarding the $c-matrix$ data structure we introduced in section 5.4, page 84 if say for a particular applications the data structure has some properties, perhaps by the nature of the interactions type and agents relationships if any, e.g. if the matrix is sparse say, there is even more possibilities for efficient algorithms for searching it. These search

[5] But in our discussion of the diffusing computation though we extended E with some closure, e.g. reflexive so the graph is acyclic.

algorithms can useful if say the *quiescence* is required (i.e. if the objective is to check if all agents or a group of agents have terminated conversations).

Regarding the logical global $c - matrix$ $\mathbf{M}$ introduced in Definition 14 in page, 96,as an implementation point for the shared memory based alternative of the distributed termination detection protocol , the global $c - matrix$ $\mathbf{M}$ can be mapped directly to existing agent registries. This will optimise procedures such as garbage collection that may follow the termination detection procedure to avoid duplicating agent registration on registries and controllers.

APPENDIX B

Prototype Implementation

B.1 Java Agent Development Framework

The prototype and experiments were developed within the JADE agent platform [23]. In principle the ideas discussed in this research could be prototyped in other tools and platforms which supports and provide mechanisms for engineering agent interaction protocols. JADE has emerged as a popular choice and is in wide spread use in the research community as an implementation framework for java based agents and has success in large testbed project like Agentcities [66].

The details of the JADE platform and its design philosophy are discussed in detail in [24] and I summarise them here.

JADE [23], is a software framework fully implemented in Java language. Its goal is to simplify the development of multi-agent systems while ensuring standard compliance through a comprehensive set of system services and agents in compliance with the FIPA[1] specifications:, i.e. naming service and yellow-page service, message transport and parsing service, and a library of FIPA interaction protocols ready to be used.

In complying to FIPA specifications, JADE includes all those mandatory components that manage the platform, i.e. the ACC, the AMS, and the DF. All agent commu-

[1]The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems.

nication is performed through message passing, where FIPA ACL is the language to represent messages.

The agent platform can be distributed on several hosts. Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is basically a container of agents that provides a complete run-time environment for agent execution and allows several agents to concurrently execute on the same host.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, time-out and pattern matching based. The full FIPA communication model has been implemented and its components have been fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontologies and finally, transport protocols.

The transport mechanism can be adapted to each situation, by transparently choosing the best available protocol. Java RMI, event-notification, HTTP, and IIOP are currently used,but more protocols can be added. Most of the interaction protocols defined by FIPA are already available and can be instantiated after defining the application-dependent behaviour [2] of each state of the protocol.

In the jade execution model agents are implemented as one thread per agent, but agents often need to execute parallel tasks. In addition to the multi-thread solution, offered directly by the JAVA language, JADE also supports scheduling of cooperative behaviours. The run-time includes also some ready to use behaviours for the most common tasks in agent development.

[2]The computational model of an agent is multitask, where tasks (or behaviours) are executed concurrently.

B.2 Implementation

The finite state machine based interaction protocols were implemented using JADE's FSM behaviour template. In the experimental setup, a number agents executing a protocol are instantiated. Figures B.1, B.2, B.3 B.3 show various processes executed by agents and controllers. JADE provides a construct called a *behaviour* that can implement these agent processes.

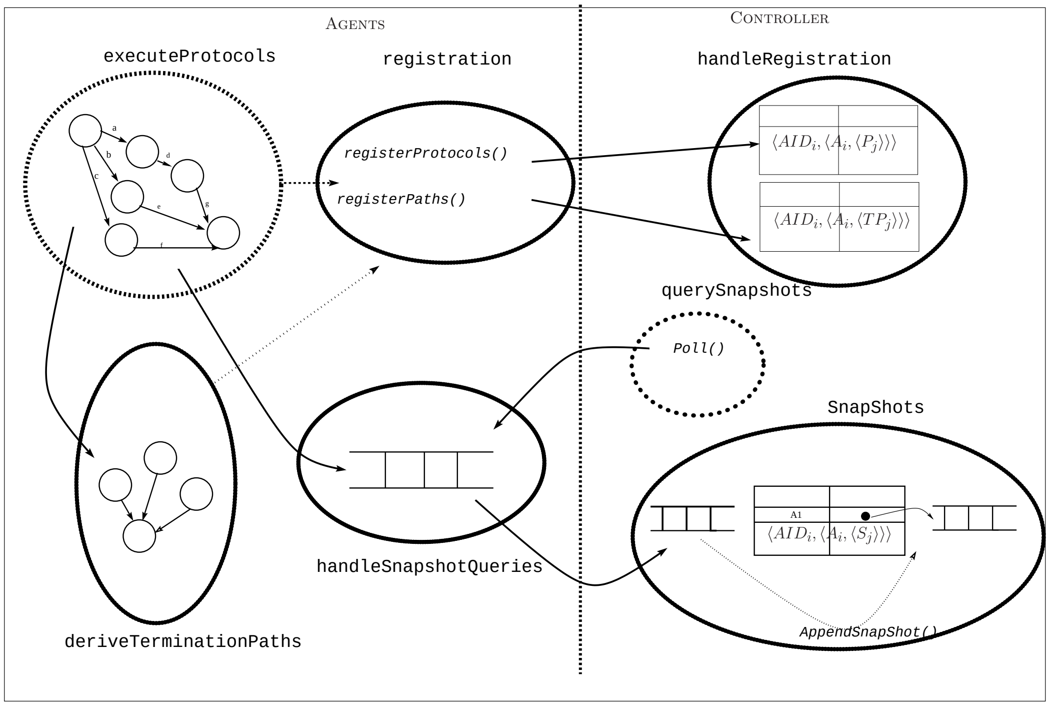

Agents
Controller
executeProtocols
registration
handleRegistration
registerProtocols()
registerPaths()
$\langle AID_i, \langle A_i, \langle P_j \rangle\rangle\rangle$
$\langle AID_i, \langle A_i, \langle TP_j \rangle\rangle\rangle$
querySnapshots
Poll()
SnapShots
A1
$\langle AID_i, \langle A_i, \langle S_j \rangle\rangle\rangle$
handleSnapshotQueries
AppendSnapShot()
deriveTerminationPaths

Figure B.1: Showing processes executed by agents and the controllers for registration, derivation of termination paths and collection of protocol execution traces

Figure B.1, shows;

I – A process for executing protocols given protocol specifications.

II – A process for registration. Depending on the experiment to be run, agents can register their full protocol specifications to the controller, or the sub-protocol comprising of *terminations paths*[3].

III – A process for deriving termination paths given protocol specifications. This can be done off line from protocol specification from the protocol library or done on agent initialisation.

IV – A process for maintaining protocol execution snapshots. This handles queries from the controllers on protocol execution.

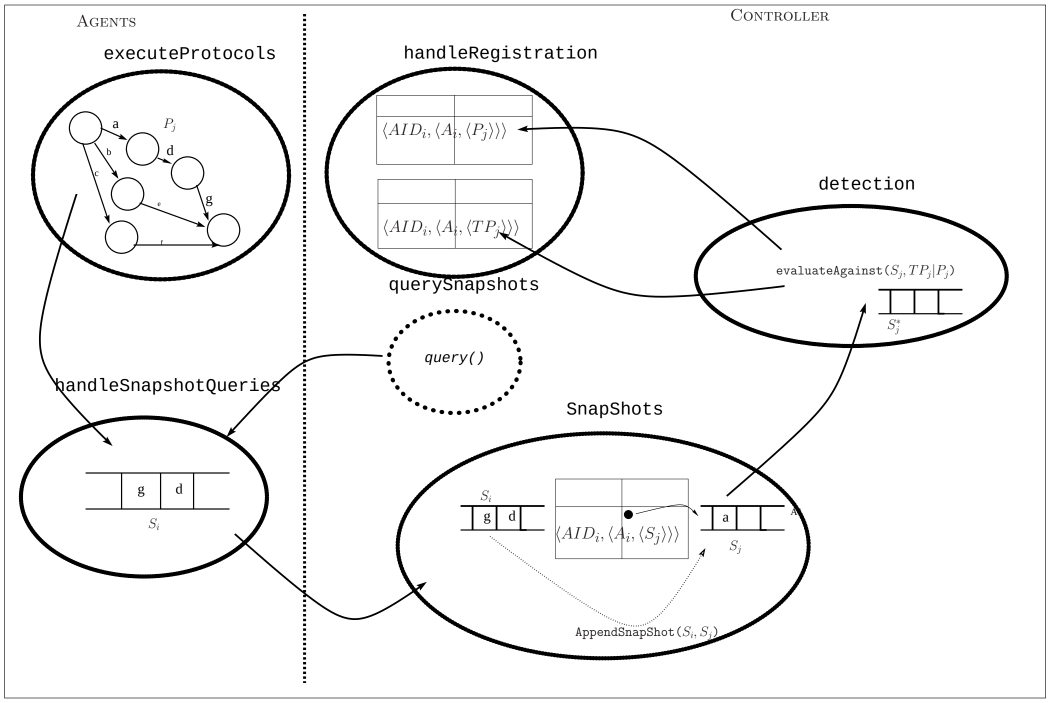
V – A process to handle protocols or sub-protocol registration. The process maintains data structures. The tuple notation $\langle AID_i, \langle A_i, \langle P_j \rangle \rangle \rangle$ for shows that an agent has an identifier, AID_i. An agent has an *agent proxy* A_i that encapsulates protocol specifications $\langle P_j \rangle$ for that agent. Similarly if only sub-protocols or termination paths were registered, $\langle AID_i, \langle A_i, \langle TP_j \rangle \rangle \rangle$, represents the tuple where $\langle TP_j \rangle$ represents the specifications of the sub-protocols or termination paths.

Figure B.2 completes Figure B.1 by showing;

VI – A process for collecting protocol execution snapshots. The process buffers protocol execution traces, and maintains data structures for organising and maintaining protocol execution snapshots for agents monitored.

VII – A process for monitoring protocols and making termination explicit given protocol or sub-protocol or termination paths specifications and protocol execution snapshots.

[3]For experimental setup purposes it does not really matter how the protocol or subprotocols are obtained, in reality there will be a protocol library that the controllers can access as proposed in Chapter 5, page 109 in Figure 5.14.



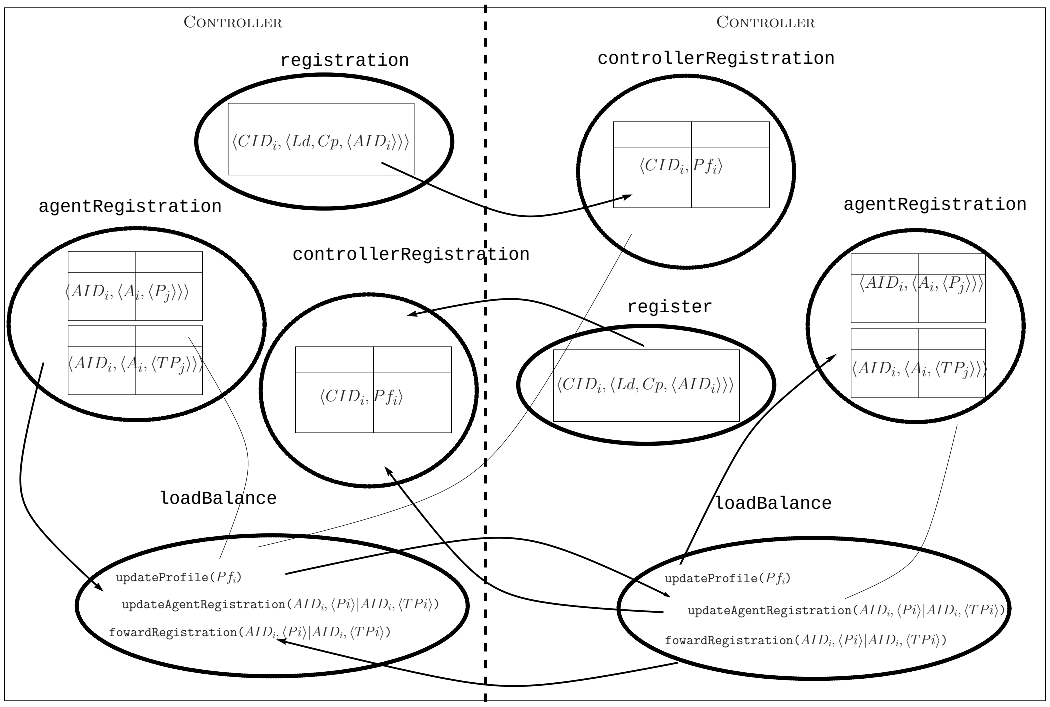
Agents
Controller
executeProtocols
handleRegistration
detection
querySnapshots
query()
handleSnapshotQueries
SnapShots
P_j
$\langle AID_i, \langle A_i, \langle P_j \rangle\rangle\rangle$
$\langle AID_i, \langle A_i, \langle TP_j \rangle\rangle\rangle$
evaluateAgainst$(S_j, TP_j|P_j)$
S_j^*
S_i
$\langle AID_i, \langle A_i, \langle S_j \rangle\rangle\rangle$
S_j
AppendSnapShot(S_i, S_j)

Figure B.2: Showing processes executed by agents and the controllers for monitoring protocol execution and termination detection.

Figure B.3 on the other hand shows processes involved in *controller-to-controller* interactions in the decentralised setup. Controllers maintain profiles and can register with each other and use a load balancing mechanism. The figure explicitly shows for each controller;

VIII – A process for registering with remote controllers. Each controller maintains a data structure to represent its *profile* Pf_ishown as a tuple $\langle CID_i, \langle Ld, Cp, \langle AID_i \rangle\rangle\rangle$ where CID_i is the controller identifier, Ld is metric representing local load, Cp represents capacity, i.e. threshold load and $\langle AID_i \rangle$ a sequence agent identifiers for registered agents registered with this controller.

IX – A process for handling registration requests by remote controllers. This hosts remote controllers's profile representations against controller identifiers as shown in the diagram as the tuple $\langle CID_{id}, Pf_i \rangle$



CONTROLLER
CONTROLLER
registration
$\langle CID_i, \langle Ld, Cp, \langle AID_i \rangle\rangle\rangle$
controllerRegistration
$\langle CID_i, Pf_i \rangle$
agentRegistration
$\langle AID_i, \langle A_i, \langle P_j \rangle\rangle\rangle$
$\langle AID_i, \langle A_i, \langle TP_j \rangle\rangle\rangle$
controllerRegistration
$\langle CID_i, Pf_i \rangle$
register
$\langle CID_i, \langle Ld, Cp, \langle AID_i \rangle\rangle\rangle$
agentRegistration
$\langle AID_i, \langle A_i, \langle P_j \rangle\rangle\rangle$
$\langle AID_i, \langle A_i, \langle TP_j \rangle\rangle\rangle$
loadBalance
updateProfile(Pf_i)
updateAgentRegistration($AID_i, \langle Pi \rangle | AID_i, \langle TPi \rangle$)
fowardRegistration($AID_i, \langle Pi \rangle | AID_i, \langle TPi \rangle$)
loadBalance
updateProfile(Pf_i)
updateAgentRegistration($AID_i, \langle Pi \rangle | AID_i, \langle TPi \rangle$)
fowardRegistration($AID_i, \langle Pi \rangle | AID_i, \langle TPi \rangle$)

Figure B.3: Showing processes for controller-controller interaction in the distributed setup. Processes are for registration, load balancing

X – A load balancing process. This for forwarding agent registrations to available remote controllers in the cluster. This is triggered if local load exceeds a defined threshold. The process also refreshes the local controller's profile stored remotely and updates agent registration data structures on receiving forwarded agents registration requests.

For experimental data Figure B.4 introduces a data collection process for agents and controllers . There are entries for agents and the protocol(s) being executed, this represented as a tuple $\langle AID_i, PID_i \rangle$. For each protocol execution a tuple $\langle PID_j, Ta_j, Tc_j, \Delta T, R, Ec \rangle$ is given, representing a protocol identifier PID_j. Each agent marks and records the start and end of protocol execution, Ta_j. The controller records Tc_j the termination time of the monitored protocol on the controller's side. ΔT then is the detection delay.

, R is a computational resource utilisation metric [4]. The experimental cycle is recorded as Ec.

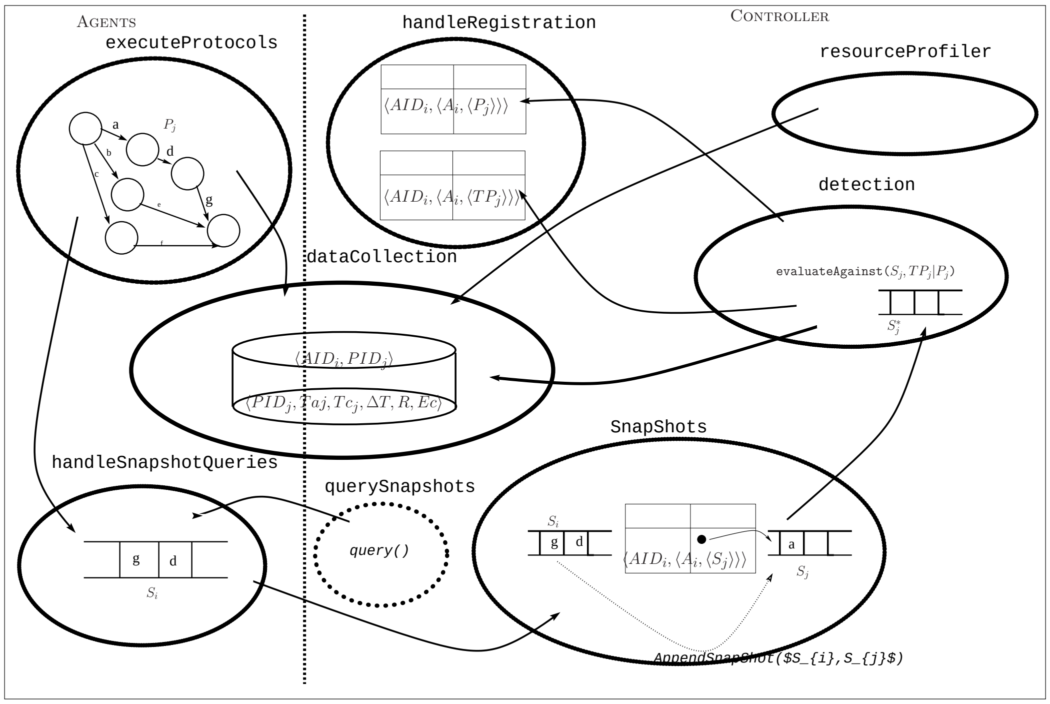

AGENTS
CONTROLLER
executeProtocols
handleRegistration
resourceProfiler
detection
dataCollection
handleSnapshotQueries
querySnapshots
SnapShots
P_j
$\langle AID_i, \langle A_i, \langle P_j \rangle \rangle \rangle$
$\langle AID_i, \langle A_i, \langle TP_j \rangle \rangle \rangle$
evaluateAgainst$(S_j, TP_j | P_j)$
S_j^*
$\langle AID_i, PID_j \rangle$
$\langle PID_j, Taj, Tc_j, \Delta T, R, Ec \rangle$
S_i
query()
$\langle AID_i, \langle A_i, \langle S_j \rangle \rangle \rangle$
S_j
AppendSnapShot(S_{i},S_{j})

Figure B.4: Showing agents-controller interaction processes for data collection and resource profiling

[4]derived from the operating system function calls to give cpu and memory utilisation, e.g. through Linux function call *top*

APPENDIX C

Experimental design and data analysis

C.1 Data collection setup

Figure C.1 gives an illustration of the data collection scenarios discussed in *Example* 12 in page 125, where we considered a set of agents $\mathcal{A}_C \subset \mathcal{A}$ that reside in the same host as some remote controller, and consider $\mathcal{A}_{Ci} \subset \mathcal{A}_C$ be agents registered with an $R_i \in \mathcal{C}$.

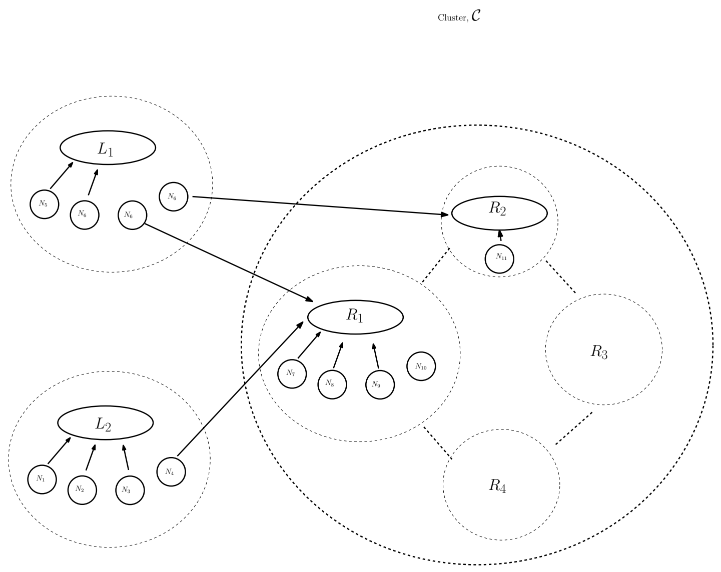
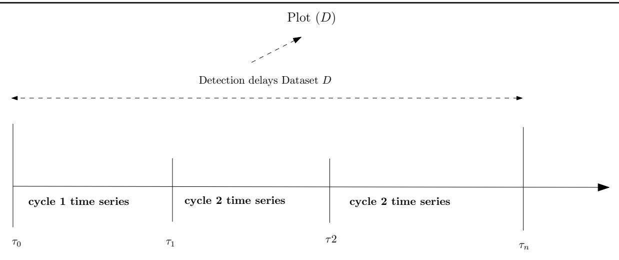


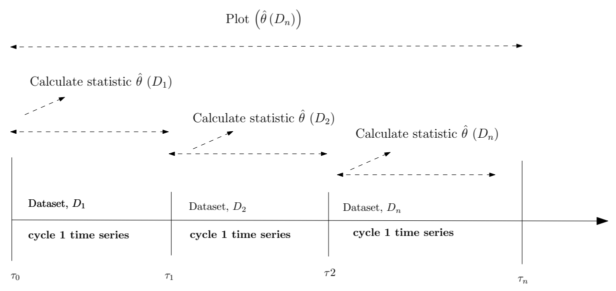
Figure C.1: Data collection setup, showing local and remote controllers in a distributed setting

C.2 Data collection during experimental cycles

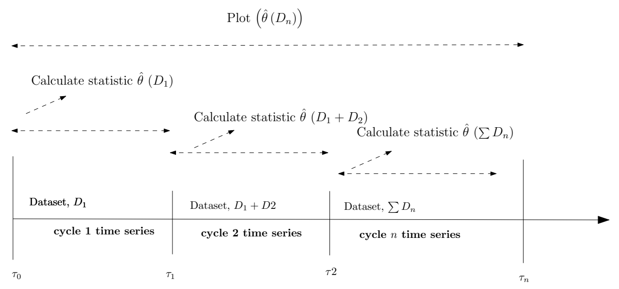
An illustration figure for experimental data collection. This illustrates various ways data collected during experimental cycles as discussed in page 175.



(a) Time series dataset


Plot $\left(\hat{\theta}\,(D_n)\right)$
Calculate statistic $\hat{\theta}$ (D_1)
Calculate statistic $\hat{\theta}$ (D_2)
Calculate statistic $\hat{\theta}$ (D_n)
Dataset, D_1
Dataset, D_2
Dataset, D_n
cycle 1 time series
cycle 1 time series
cycle 1 time series
τ_0
τ_1
$\tau 2$
τ_n

(b) Cyclic dataset


Plot $\left(\hat{\theta}\,(D_n)\right)$
Calculate statistic $\hat{\theta}$ (D_1)
Calculate statistic $\hat{\theta}$ $(D_1 + D_2)$
Calculate statistic $\hat{\theta}$ $(\sum D_n)$
Dataset, D_1
Dataset, $D_1 + D2$
Dataset, $\sum D_n$
cycle 1 time series
cycle 2 time series
cycle n time series
τ_0
τ_1
$\tau 2$
τ_n

(c) Accumulated dataset

Figure C.2: Figures a-c show various data collection scenarios for time series data over experimental cycles

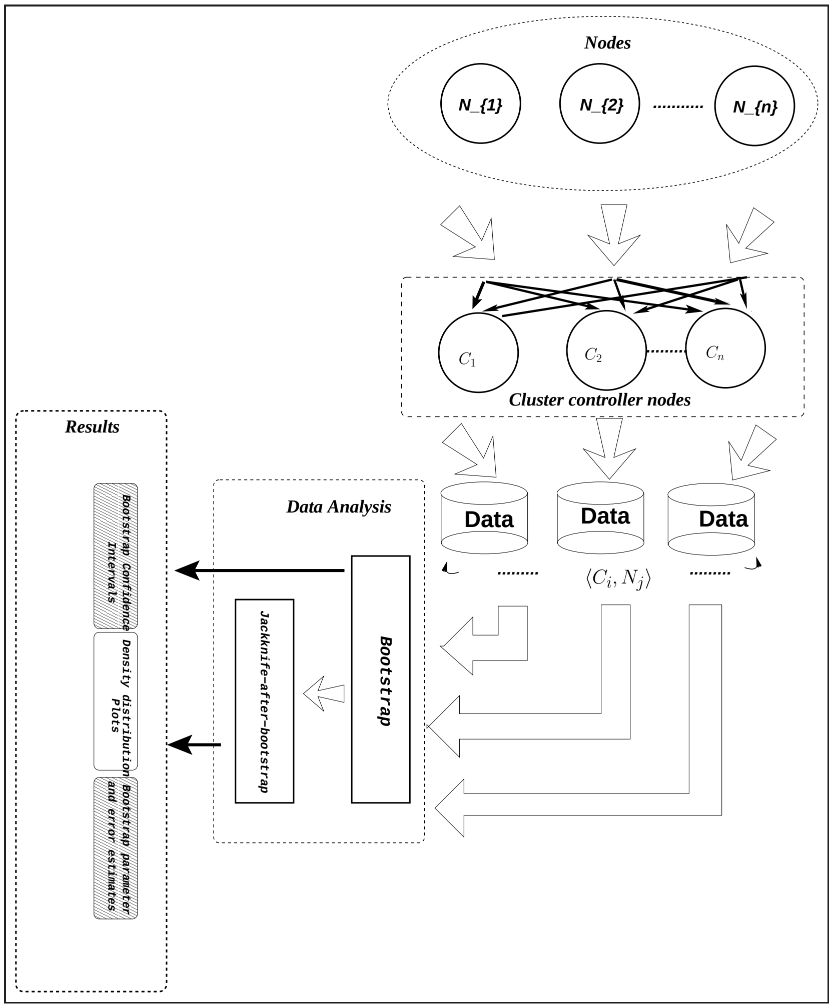

Nodes
N_{1}
N_{2}
N_{n}
C_1
C_2
C_n
Cluster controller nodes
Results
Data Analysis
Data
Data
Data
$\langle C_i, N_j \rangle$
Bootstrap
Jackknife-after-bootstrap
Bootstrap Confidence Intervals
Density distribution Plots
Bootstrap parameter and error estimates

Figure C.3: Data collection and analysis for the distributed setting

C.3 Hardware used in experiments

Regarding the hardware configuration in the distributed setup experiments, consider Figure C.1 below.

	Os	Arch & Kernel	Cpu(s)(Ghz)	Cache (KB)	Ram (GB)	Swap (GB)
Core Cluster Nodes						
Controller , R_i	GNU/Linux	32 bit i686, 2.4.20-8 SMP	4×2.4 Intel Xeon	512	1.03	2.09
Controller , R_i	GNU/Linux	32 bit i686, 2.4.22 SMP	2×0.866 Pentium III (Coppermine)	256	0.904	1.06
(Coppermine)	256	0.513	1.05			
Controller , R_i	SunOS 5.9	64 bit 4-way Superscalar SPARC V9	4×1.6 SUNW UltraSPARCIIIi	1000	8.2	37
Client Nodes						
Clients , L_i	GNU/Linux	32 bit i686, 2.4.20-8 SMP	1×3.20 Pentium IV	1024	0.512	8.03

Table C.1: The hardware specifications for machines used in the experiments for the distributed setting. The first four machines are controllers in the cluster as described in the experimental architecture

APPENDIX D

Tutorial on resampling statistical methods

D.1 The Bootstrap

The Bootstrap (see [81] [1] [69, 211, 204] for details) refers to the process of repeatedly drawing samples, with replacement, from data collected [2]. Instead of trusting theory to describe the sampling distribution of an estimator (e.g. mean), we estimate that distribution empirically. Drawing *k* bootstrap samples of size *n* (from an origional sample of also size *n*) yield *k* new estimates. The distribution of these bootstrap estimates provides an empirical basis for estimating standard errors or confidence intervals. The bootstrap essentially "simulates" repeating the experiment however many times as required.

Detailed bootstrap procedures and algorithms are described in the standard reference on the Bootstrap [83]. The general procedure for performing the bootstrap can be written as follows;

More formally [83] pp. 44,

- Consider a random sample x= (x_1,x_2,...,x_n) from an unknown probability distribution F. We wish to estimate a parameter of interest θ = $t(F)$ on the basis of x Typically for this purpose we calculate the estimate $\hat{\theta}$ = s(x).

[1] Brad Efron wrote the key paper rediscovering the bootstrap and his famous 1979 paper in the Annals of Statistics.

[2] Unlike monte carlo simulations which fabricate their data, bootstrapping works with real data.

Algorithm 9 The Bootstrap algorithm

```
Set $B > 1000$
**repeat**
    • Draw a resample with replacement from the data.
    • Calculate the resample mean.
    • Save the resample mean into a variable.
**until** B TIMES
• Make a histogram and normal quartile plot of the B means.
• Calculate the standard deviation of the B means.
```

- Bootstrap methods depend on the *bootstrap sample*. Define $\hat{F}$ to be the empirical distribution, putting the probability $1/n$ on each of the observed x_i, $i = 1, 2, ..., n$. A bootstrap sample is then defined to be a random sample of size n drawn from $\hat{F}$, say x^* =x_1^*, x_2^*,...,x_n^*, written [3]

$$\hat{F} \rightarrow (x_1^*, x_2^*, ..., x_n^*) \tag{D.1}$$

  D.1 can also be understood to mean that the bootstrap data points x_1^*, x_2^*,...,x_n^* are a random sample of size n drawn *with replacement* from the population of n objects (x_1,x_2,...,x_n).

- Corresponding to a bootstrap dataset x^* is a bootstrap replication of $\hat{\theta}$

$$\hat{\theta}^* = s(x^*) \tag{D.2}$$

  The quantity $s(x^*)$ is the result of applying the same function s(.) to x^* as was applied to x, e.g. if $s(x)$ is the sample mean $\bar{x}$ then $s(x^*)$ is the mean of the bootstrap dataset, $\bar{x}$ = $\sum_{i=1}^{n}(x_i^*/n)$

[3]The star notation indicates that x^* is not the actual data set x but rather the randomized, or resampled version of x.

- The bootstrap estimate of $se_F(\hat{\theta})$, the standard error of a statistic $\hat{\theta}$, is a plu$-$in estimate that uses the empirical distribution function $\hat{F}$ in place of the unknown distribution F. Specifically the bootstrap estimate of $se_F(\hat{\theta})$ is defined by

$$se_{\hat{F}}(\hat{\theta}^*) \tag{D.3}$$

  i.e. the bootstrap estimate of $se_F(\hat{\theta})$ is the standard error of $\hat{\theta}$ for data set of size n randomly sampled from $\hat{F}$.

- Recalling that the standard error of the mean $\bar{x}$, written $se_F(\bar{x})$ is the square root of the variance of x

$$se_F(\bar{x}) = [var_F(\bar{x})]^{1/2} = \frac{\sigma_F}{\sqrt{n}} \tag{D.4}$$

  and that apart from the mean, there exist no formulae to compute numerical values of the ideal estimates exactly. The bootstrap algorithm below is a computational way of obtaining a numerical value of $se_{\hat{F}}\left(\hat{\theta}^*\right)$

PROCEDURE - BASIC BOOTSTRAP

1. *Given a random sample,x= (x_1,x_2,...,x_n) , calculate $\hat{\theta}$.*
2. *Sample with replacement from the original sample to get x^* =x_1^*, x_2^*,...,x_n^**
3. *Calculate the same statistic using the bootstrap sample in step 2 to get, $\hat{\theta}^*$.*
4. *Repeat steps 2 through 3, B times.*
5. *Use this estimate of the distribution of $\hat{\theta}$ (i.e., the bootstrap replicates) to obtain the desired characteristic (e.g., standard error, bias or confidence interval).*

B is generally a large number, typically ≥ 1000 [4]

Bootstrap Estimate of Standard Error When our goal is to estimate the standard error of using the bootstrap method, we proceed as outlined in the previous procedure. Once we have the estimated distribution for $\hat{\theta}$, we use it to estimate the standard error for . This estimate is given by

$$S\hat{E}_B\left(\hat{\theta}\right) = \left\{\frac{1}{B-1}\sum_{b=1}^{B}(\theta^{*b} - \bar{\hat{\theta}}^{*})^2\right\}^{\frac{1}{2}} \tag{D.5}$$

where

$$\bar{\hat{\theta}} = \frac{1}{B}\sum_{b=1}^{B}\theta^{*b} \tag{D.6}$$

It is worth observing that Equation D.5 is just the sample standard deviation of the bootstrap replicates, and Equation D.6 is the sample mean of the bootstrap replicates. [83] show that the number of bootstrap replicates B should be between 50 and 200 when estimating the standard error of a statistic. Often the choice of B is dictated by the computational complexity of $\hat{\theta}$, the sample size n, and the computer resources that are available.

PROCEDURE - BOOTSTRAP ESTIMATE OF THE STANDARD ERROR

1. Given a random sample $x = x_1, \ldots, x_n$ calculate the statistic $\hat{\theta}$.

2. Sample with replacement from the original sample to get x^{*b} =x_1^{*b},...,x_n^{*b}.

[4]Even larger value if for example more accurate estimates are required, e.g. if narrower bands of confidence intervals are desirable.

3. Calculate the same statistic using the sample in step 2 to get the bootstrap replicates, $\hat{\theta^{*b}}$.

4. Repeat steps 2 through 3, B times.

5. Estimate the standard error of using Equations D.5 and D.6.

Estimates of bias The standard error of an estimate discussed above is one measure of its performance. Bias is another quantity that measures the statistical accuracy of an estimate. The bias in an estimator gives a measure of how much error we have, on average, in our estimate when we use T to estimate our parameter θ.

Bias is defined as the difference between the *expected value* of the statistic and the parameter,

$$Bias\left(T\right) = \mathbb{E}\left[T\right] - \theta. \tag{D.7}$$

clearly if the estimator is *unbiased*, then the expected value of our estimator equals the true parameter value, so $(\mathbb{E}) = \theta$. Normally in order to determine the expected value in Equation D.7, the *distribution* of the statistic T must be known, i.e. the expectation in Equation D.7 is taken with respect to the true distribution F. In these situations, the bias can be determined analytically [156]. When the distribution of the statistic is not known, then we can use methods such as the jackknife and the bootstrap discussed in this section to estimate the bias of T. To get the bootstrap estimate of bias, we use the empirical distribution as before. We resample from the empirical distribution and calculate the statistic using each bootstrap resample, yielding the bootstrap replicates $\hat{\theta^{*b}}$. We use these to estimate the bias from the following:

$$\hat{bias_B} = \bar{\hat{\theta}}^{*} - \hat{\theta} \tag{D.8}$$

where $\bar{\hat{\theta}}^*$ is given by the mean of the bootstrap replicates (Equation D.6). We are interested in the bias in order to correct for it. The bias−corrected estimator is given by

$$\widehat{\theta} = \hat{\theta} - bi\hat{a}s_B \tag{D.9}$$

and using Equations D.8 D.9 we have

$$\widehat{\theta} = 2\hat{\theta} - \bar{\hat{\theta}}^* \tag{D.10}$$

More bootstrap samples are needed to estimate the bias, than are required to estimate the standard error. [83] recommend that $B \geq 400$.

The procedure for estimating the bias is given below.

PROCEDURE - BOOTSTRAP ESTIMATE OF THE BIAS

1. Given a random sample,$x = (x_1, \ldots, x_n)$, calculate the statistic $\hat{\theta}$.
2. Sample with replacement from the original sample to get $x^{*b} = x_1^{*b}, \ldots, x_n^{*b}$.
3. Calculate the same statistic using the sample in step 2 to get the bootstrap replicates, $\hat{\theta^{*b}}$.
4. Repeat steps 2 through 3, B times.
5. Using the bootstrap replicates, calculate $\bar{\hat{\theta}}^*$.
6. Estimate the bias of using Equation D.8.

D.2 Bootstrap confidence intervals

Bootstrap allows calculation of confidence intervals in a number of ways, namely the *standard interval*, the *bootstrap-t interval* and the *percentile* method.

Bootstrap Standard Confidence Interval This is based on the parametric of the confidence interval. It can be shown that the $(1-\alpha).100\%$ confidence interval for the mean can be found using

$$\mathbf{P}\left(\bar{X} - z^{\left(1-\frac{\alpha}{2}\right)}\frac{\sigma}{\sqrt{n}} < \mu < \bar{X} - z^{\left(\frac{\alpha}{2}\right)}\frac{\sigma}{\sqrt{n}}\right) = 1-\alpha \tag{D.11}$$

Similarly, the bootstrap standard confidence interval is given by

$$\left(\hat{\theta} - z^{(1-\alpha/2)}SE_{\hat{\theta}}, \hat{\theta} - z^{(\alpha/2)}SE_{\hat{\theta}}\right) \tag{D.12}$$

where $SE_{\hat{\theta}}$ is the standard error for the statistic $\hat{\theta}$ obtained using the bootstrap [170]. The confidence interval in Equation D.12 can be used when the distribution for $\hat{\theta}$ is normally distributed or the normality assumption is plausible.

Bootstrap-*t* Confidence Interval for this type of intervals, first generate B bootstrap samples, and for each bootstrap sample the following quantity is computed:

$$z^{*b} = \frac{\hat{\theta^{*b}} - \hat{\theta}}{\hat{SE^{*B}}} \tag{D.13}$$

As before, $\hat{\theta^{*b}}$ is the bootstrap replicate of $\hat{\theta}$, but $\hat{SE^{*B}}$ is the estimated standard error of for that bootstrap.

Once we have the B bootstrapped values from Equation D.13, the next step is to estimate the quantizes needed for the endpoints of the interval. The $\alpha/2-$th quartile, denoted by $t^{\hat{\alpha/2}}$ of the z^{*b}, is estimated by

$$\alpha/2 = \frac{\#\left(z^{*b} \leq t^{\hat{\alpha/2}}\right)}{B} \tag{D.14}$$

This is then used to calculate the bootstrap$-t$ confidence interval, which as a result is given by

$$\left(\hat{\theta} - t^{(1-\alpha/2)} \cdot S\hat{E}\theta, \hat{\theta} - t^{(\alpha/2)} \cdot S\hat{E}\theta\right) \tag{D.15}$$

where $\hat{SE}$ is an estimate of the standard error of $\hat{\theta}$. The bootstrap$-t$ interval is reported to be suitable for *location statistics* such as the mean or quantizes. However, its accuracy for more general situations is questionable [83]. The procedure for determining the bootstrap-*t* intervals is outlined below:

PROCEDURE - BOOTSTRAP-T CONFIDENCE INTERVAL

1. Given a random sample,$x = (x_1, \ldots, x_n)$, calculate $\hat{\theta}$.
2. Sample with replacement from the original sample to get $x^{*b} = x_1^{*b}, \ldots, x_n^{*b}$.
3. Calculate the same statistic using the sample in step 2 to get $\hat{\theta^{*b}}$.
4. Use the bootstrap sample **x**^{*b}to get the standard error of $\hat{\theta^{*b}}$. This can be calculated using a formula or estimated by the bootstrap.
5. Calculate z^{*b} using the information found in steps 3 and 4.
6. Repeat steps 2 through 5, B times, where $B \geq 1000$.
7. Order the z^{*b}'s from smallest to largest. Find the quantizes $t^{(1-\hat{\alpha}/2)}$ and $t^{(\hat{\alpha}/2)}$.

8. Estimate the standard error $S\hat{E}\theta$ of $\hat{\theta}$ using the B bootstrap replicates of $\hat{\theta^{*b}}$ (from step 3).

9. Use Equation D.15 to get the confidence interval.

[156] observes that the number of bootstrap replicates that are needed is quite large for confidence intervals. It is recommended that $B \geq 1000$. If no formula exists for calculating the standard error of $\hat{\theta^{*b}}$, then the bootstrap method can be used. This means that there are two levels of bootstrapping: one for finding the $S\hat{E}^{*b}$ and one for finding the z^{*b}, which can greatly increase the computational burden. For example, say that $B = 1000$ and we use 50 bootstrap replicates to find $S\hat{E}^{*b}$, then this results in a total of 50,000 resamples.

Bootstrap Percentile Interval This is an improved bootstrap confidence interval based on the quantizes of the distribution of the bootstrap replicates. This technique has the benefit of being more stable than the bootstrap$-t$, and it also enjoys better theoretical coverage properties [83].

The bootstrap percentile confidence interval is given by:

$$\left(\theta_B^{*(\hat{\alpha/2})}, \theta_B^{*(1\hat{-}\alpha/2)}\right) \tag{D.16}$$

where $\theta_B^{*(\hat{\alpha/2})}$ is the $\alpha/2$ quartile in the bootstrap distribution of θ^*. For example, if $\alpha/2 = 0.025$ and $B = 1000$, then $\theta_B^{*(0\hat{.}025)}$ is the $\hat{\theta^{*b}}$ in the 25*th* position of the ordered bootstrap replicates. Similarly, $\theta_B^{*(0\hat{.}975)}$ is the replicate in position 975. The procedure is the same as the general bootstrap method, and is outlined in the steps below.

PROCEDURE - BOOTSTRAP PERCENTILE INTERVAL

1. Given a random sample,$x = (x_1, \ldots, x_n)$, calculate $\hat{\theta}$. .

2. Sample with replacement from the original sample to get $x^{*b} = x_1^{*b}, \ldots, x_n^{*b}$.

3. Calculate the same statistic using the sample in step 2 to get the bootstrap replicates, $\hat{\theta^{*b}}$.

4. Repeat steps 2 through 3, B times, where $B \geq 1000$.

5. Order the $\hat{\theta^{*b}}$ from smallest to largest.

6. Calculate and $B \cdot (\alpha/2)$ and $B \cdot (1 - \alpha/2)$.

BCa Intervals The BC_a (*bias-corrected and accelerated*) bootstrap confidence interval is an improvement on the bootstrap percentile interval and is superior to all the other intervals discussed here. As discussed above, the upper and lower endpoints of the bootstrap percentile confidence interval are given by:

$$PercentileInterval : \left(\hat{\theta_{Lo}}, \hat{\theta_{Hi}}\right) = \left(\theta_B^{*\hat{(\alpha/2)}}, \theta_B^{*(1\hat{-}\alpha/2)}\right) \tag{D.17}$$

where (if for example we are considering 90% intervals), $\hat{\theta_{Lo}}$ is the bootstrap replicate in the 5*th* position and of the ordered list of replicates. Similarly in this example, $\hat{\theta_{Hi}}$ is the bootstrap replicate in the 95*th* position.

The BC_a interval adjusts the endpoints of the interval based on two parameters, a and z_o. The $(1 - \alpha)\,.100\%$ confidence interval using the method is

$$PercentileInterval : \left(\hat{\theta_{Lo}}, \hat{\theta_{Hi}}\right) = \left(\theta_B^{*\hat{(\alpha_1)}}, \theta_B^{*\hat{(\alpha_2)}}\right) \tag{D.18}$$

where

$$\begin{aligned}\alpha_1 &= \Phi\left(\hat{z_0} + \frac{\hat{z_0} + z^{(\alpha/2)}}{1 - \hat{a}\left(\hat{z_0} + z^{(\alpha/2)}\right)}\right) \\ \alpha_2 &= \Phi\left(\hat{z_0} + \frac{\hat{z_0} + z^{(1-\alpha/2)}}{1 - \hat{a}\left(\hat{z_0} + z^{(1-\alpha/2)}\right)}\right)\end{aligned} \tag{D.19}$$

In Equation D.20 Φ denotes the *standard normal* cumulative distribution function, therefore $0 \leq \alpha_1 \leq 1$ and $0 \leq \alpha_2 \leq 1$. Therefore the end points of the interval in Equation D.18 are adjusted using the information from the distribution of the bootstrap replicates(instead of basing the endpoints on the confidence level $1 - \alpha$).

Note from Equation D.20 that if both $\hat{a}$ and $\hat{z_o}$ the BC_a is the same as the bootstrap percentile interval.

$$\alpha_1 = \Phi\left\{0 + \frac{0 + z^{(\alpha/2)}}{1 - 0\left(0 + z^{(\alpha/2)}\right)}\right\} = \Phi\left(z^{(\alpha/2)}\right) = \alpha/2$$

The factors $\hat{z_0}$ and $\hat{a}$ are bias correction and acceleration respectively [156] The bias-correction is based on the proportion of bootstrap $\hat{\theta^{*b}}$ replicates that are less than the statistic $\hat{\theta}$ calculated from the original sample. It is given by

$$\hat{z_0} = \Phi^{-1}\left(\frac{\#\left(\hat{\theta^{*b}} < \hat{\theta}\right)}{B}\right) \tag{D.20}$$

where Φ^{-1} denotes the inverse of the standard normal cumulative distribution function. The acceleration parameter $\hat{a}$ is obtained using the jackknife procedure as follows,

$$\hat{a} = \frac{\sum_{i=1}^{n} \left\{ \theta^{\bar{\hat{(j)}}} - \theta^{\hat{-i}} \right\}^{3}}{6 \left\{ \sum_{i=1}^{n} \left(\theta^{\bar{\hat{(j)}}} - \theta^{\hat{(-i)}} \right)^{2} \right\}^{3/2}} \tag{D.21}$$

where $\theta^{\hat{-i}}$ is the value of the statistic using the sample with the $i-$*th* data point removed (the $i-$*th* jackknife sample) and

$$\theta^{\bar{\hat{(j)}}} = \sum_{i=1}^{n} \theta^{\hat{(-i)}} \tag{D.22}$$

More theoretical details can be seen in see Efron and Tibshirani [1993] and Efron [1987].

D.3 The Jackknife

Jacknife (also referred to as Quenouille$-$Tukey Jackknife) is another resampling technique, developed before the Efron's bootstrap [5]. Like the bootstrap, it also aims at providing a computational procedure to estimate and compensate for bias and to derive robust estimates of standard errors and confidence intervals. Jackknife though is a less general technique than the bootstrap, and it explores the sample variation in a different way from the bootstrap. Jackknifed statistics are developed by systematically dropping out subsets of data one at a time and assessing the resulting variation in the studied parameter [170]. We discuss it here because in practise jackknife is typically used in conjunction with the bootstrap in a technique termed *Jackknife-After-Bootstrap*, which we have used in our analysis.

Regarding jackknife, suppose we wish to estimate the bias and the standard error of

[5]First introduced by Quenouille in 1949 and later developed by John W. Tukey in 1958.

$\hat{\theta}$. $\hat{\theta}$ might be the mean, the variance, the correlation coefficient or some other statistic of interest, then consider the following definition;

Definition 17. *Jackknife*
Suppose we have a sample $x = x_1, \ldots, x_n$ and an estimator $\hat{\theta} = s(x)$. The jackknife focuses on the samples that leave out one observation at a time:

$$x(i) = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \tag{D.23}$$

for $i = 1, 2, \ldots, n$ called jackknife samples. The i_{th} jackknife sample consists of the data set with the ith observation removed. let

$$\hat{\theta_{(i)}} = s\left(x_{(i)}\right) \tag{D.24}$$

be the i_{th} jackknife replication of $\hat{\theta}$.
The jackknife estimate of bias is defined by

$$\widehat{bias}_{jack} = (n-1)\left(\hat{\theta_{(.)}} - \hat{\theta}\right) \tag{D.25}$$

where

$$\hat{\theta_{(.)}} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta_{(i)}} \tag{D.26}$$

the jackknife estimate of standard error defined by

$$\widehat{se}_{jack} = \left\{\frac{n-1}{n}\sum_{i=1}^{n}\left(\hat{\theta_{(i)}} - \hat{\theta_{(.)}}\right)^2\right\}^{\frac{1}{2}} \tag{D.27}$$

Theoretical details of the jacknife especially justification of the factor $\frac{n-1}{n}$ in Equation D.27 can be seen in [83].

The procedure for the jackknife is summarised below.

PROCEDURE - JACKKNIFE

1. Leave out an observation.
2. Calculate the value of the statistic using the remaining sample points to obtain $\hat{\theta_{(i)}}$.
3. Repeat steps 1 and 2, leaving out one point at a time, until all n $\hat{\theta_{(i)}}$ are recorded.
4. Calculate the jackknife estimate of the bias of $\hat{\theta}$ using Equation D.25.
5. Calculate the jackknife estimate of the standard error of T using Equation D.27.

Comparisons between the bootstrap and jackknife are also discussed in [83] pp. 145, to advise how to chose between the two methods. What is worth noting is that the jackknife can be viewed as a simple approximation of the bootstrap for the estimation of standard errors and bias. It is also worth noting that the jackknife can fail badly if the statistic $\hat{\theta}$ is not "smooth" [6], such as is the case with the median (median is not a *diffentiable*,or smooth function of x)

As stated before it is common practise to use the jackknife in conjunction with the bootstrap in a technique termed *Jackknife-After-Bootstrap* which we discuss next.

D.3.1 Jackknife-After-Bootstrap

When using the bootstrap to get estimates of standard error and bias, the values obtained are also estimates, therefore they also have error associated with them. This error arises from two sources, one of which is the usual sampling variability because

[6]Intuitively the idea of smoothness is that small changes in the data set cause only small changes in the statistic.

we are working with the sample instead of the population. The other variability comes from the fact that we are working with a finite number B of bootstrap samples.

The so called *jacknife-after-bootstrap* technique can be used to estimate this variability. The technique allows us to obtain estimates of variation in functionals[7] of a bootstrap distribution without performing a second level bootstrap. The characteristic of the problem and the procedure is similar to that of the bootstrap , the main difference being that the resampling is done *without* replacement [156].

For example, suppose a bootstrap estimate of some statistic (e.g. estimate of standard error) has been obtained. Denote this estimate as $\hat{\gamma_B}$.

To obtain the jackknife-after-bootstrap estimate of the *variability* of $\hat{\gamma_B}$, one data point at a time is left out and using the bootstrap method calculate $\gamma_B^{(-1)}$ on the remaining data points. We continue in this way until we have the n values of $\gamma_B^{(-1)}$. An estimate of the variance of $\hat{\gamma_B}$ using the $\gamma_B^{(-1)}$ values, is as follows

$$v\hat{ar}_{jack}\left(\hat{\gamma_B}\right) = \frac{n-1}{n}\sum_{i=1}^{n}(\gamma_B^{(-i)} - \bar{\hat{\gamma}}_B)^2 \tag{D.28}$$

where $\bar{\hat{\gamma}}_B = \frac{1}{n}\sum_{i=1}^{n}\gamma_B^{(-i)}$

[156] and [83] give details and discuss an efficient way of performing the jackknife-after-bootstrap summarised in the procedure below.

PROCEDURE - JACKKNIFE-AFTER-BOOTSTRAP

1. Given a random sample x = (x_1,x_2,...,x_n), calculate a statistic of interest $\hat{\theta}$.

2. Sample with replacement from the original sample to get a bootstrap sample x^{*b} = x_1^*, x_2^*,...,x_n^*.

[7]Such as bias and standard error of a statistic.

3. Using the sample obtained in step 2, calculate the same statistic that was determined in step one and denote by $\hat{\theta^{*b}}$.

4. Repeat steps 2 through 3, B times to estimate the distribution of $\hat{\theta}$.

5. Estimate the desired feature of the distribution of (e.g., standard error, bias, etc.) by calculating the corresponding feature of the distribution of $\hat{\theta^{*b}}$. Denote this bootstrap estimated feature as $\hat{\gamma_B}$.

6. Now get the error in $\hat{\gamma_B}$. For $i = 1, ..., n$, find all samples x^{*b} = x_1^*, x_2^*,...,x_n^* that do not contain the point x_i. These are the bootstrap samples that can be used to calculate $\gamma_B^{(-i)}$.

7. Calculate the estimate of the variance of using Equation D.28.

The procedure can also provide information on the influence of each observation on the functionals [83]. Regarding the JAB, it is also worth mentioning one caveat; simulation studies have shown that, in general, jackknife after bootstrap standard error estimates tend to be too large. This especially true for where bootstrap samples was not large. The JAB estimates were inflated and performed poorly A technique called weighted jackknife after bootstrap [237] have been proposed resolve some of these difficulties. For our purpose we use large bootstrap samples $\geq$ 1000, to mitigate these effects.

D.4 Computational statistical tools support for resampling techniques

DATAPLOT, R, SPLUS™and MATLAB™and other modern tools provide computational in-built statistical functions or contributed scripts for performing the bootstrap and jacknife-after-bootstrap capabilities which collectively can be used to perform the bootstrap and the jackknife-after-bootstrap as described in the above discussion.

We have used these tools to perform analysis. Table 1-5 present a summary of the results of the analysis for various experiments.

The bootstrap plot that results from plotting the bootstrap replicates

Below we present results obtained using these tools. Appendix A presents scripts written for these purposes.

The detailed procedure and algorithms for the bootstrap is outlined in Efron and Tibshirani. For determining estimates for the standard errors,the procedure is as follows:

- What does the sampling distribution for the statistic look like?
- What is a 95 percent confidence interval for the statistic?
- Which statistic has a sampling distribution with the smallest variance? That is, which statistic generates the narrowest confidence interval?

D.5 Experiences with jacknife-after-bootstrap

Recall that as the bootstrap estimates [8] have an associated uncertainty as they are estimates. The *jackknife-after-bootstrap* method provides a mechanism for giving a measure of this uncertainty, in particular the uncertainty in their standard errors.

In addition,it is worth noting though (as also discussed earlier) , the jackknife-after-bootstrap procedure tends to over-estimate these errors, especially for small values of B (the number of bootstrap replicates chosen).

Figure D.1 below shows results of the *jackknife-after-bootstrap* experiment in which the number of bootstrap samples was varied in order to observe the effect it had on the estimate of the error. The dataset used for this example is just one of six datasets in

[8] Are by definition themselves estimates.

Table 10.3 (table has 6 columns,one for each node),page 202 therefore this experiment can be easily repeated for the other datasets. As can be observed from the figure, the estimate of the *Standard error* converges and becomes more accurate with an increase in bootstrap samples. Therefore the values quoted in the *jackknife-after-bootstrap* section of **Table** D.1 are taken from the last experiment with the largest value of B.

the jacknife-after.bootstrap procedure is described in the appendix ,where for every selected

1. $Bi\hat{a}s_B = \hat{\theta^{*b}} - \hat{\theta}$, the bootstrap estimate of bias of the mean.

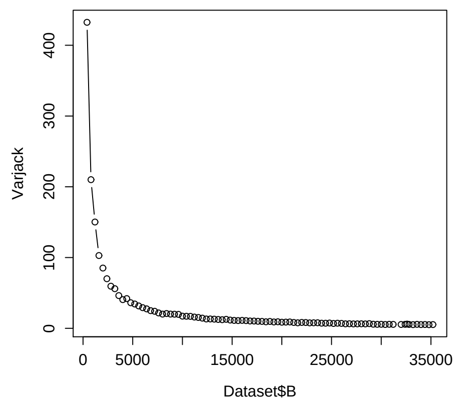
2. $\hat{\gamma_B} = Var\left(\theta\right)$ is the bootstrap estimate of the variance of the mean.

3. $SE\left(\hat{\gamma_B}\right)$ is the bootstrap estimate of the standard error in the estimate of $\hat{\gamma_B}$.

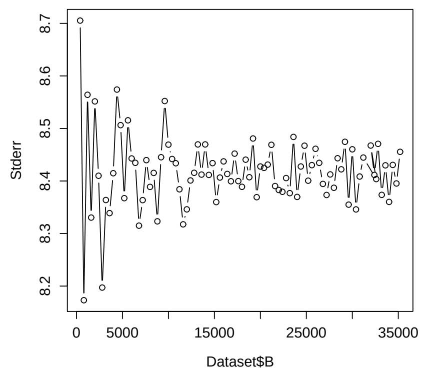
The Bootstrap							
		Nodes					
Statistic		nsqa0412a01	nsqa0413b01	nsqa0413g01	nsqa0413g03	nsqa0413j02	nsqa0413l01
$\hat{\theta}$		1660.953	1685.893	1659.375	1854.178	1851.977	1847.672
$\hat{bias}$		-0.0755	-0.0551	0.11069	-0.6033	-0.102	0.381
$\hat{se}$		8.484	13.158	7.472	33.325	21.35	49.569
Confidence Intervals							
	α						
	0.025	1646.089	1663.014	1644.897	1798.309	1812.624	1772.723
	0.050	1648.32	1666.385	1647.012	1806.002	1818.275	1782.276
	0.100	1650.648	1670.137	1649.868	1816.397	1825.107	1796.012
	0.16	1653.206	1673.422	1651.928	1825.247	1831.454	1806.606
	0.84	1669.553	1698.932	1666.977	1893.080	1872.956	1907.023
$BC_a points$	**0.9**	1672.190	1703.182	1669.099	1904.864	1879.848	1927.294
	0.95	1675.384	1708.713	1672.056	1920.293	1888.257	1956.207
	0.975	1677.886	1713.793	1675.284	1934.157	1894.522	1985.674
	z0	0.003759951	0.002506631	-0.003759951	0.05893987	-0.02506891	0.05893987
	ahat	0.01254519	0.0251825	0.01165997	0.03240697	0.01921675	0.06216931
	0.25	1655.620	1679.916	1654.137	1828.536	1838.525	1825.699

	0.5	1660.612	1687.441	1659.257	1857.283	1851.336	1857.199
	0.1	1650.106	1671.303	1650.396	1807.367	1825.623	1803.543
	0.16	1653.558	1675.754	1652.46	1818.486	1831.434	1811.687
	0.84	1670.000	1700.852	1666.666	1894.468	1873.574	1916.205
$Bootstap - t$	**0.9**	1672.985	1704.029	1669.423	1905.653	1877.411	1955.739
	0.95	1676.172	1708.785	1672.324	1921.87	1884.27	1996.275
	0.975	1679.208	1718.862	1674.188	1932.500	1891.330	2035.485

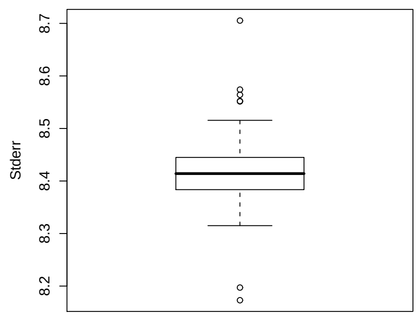
Table D.1: Bootstrap : showing parameter and confidence interval estimates and errors for agents monitored locally at given nodes



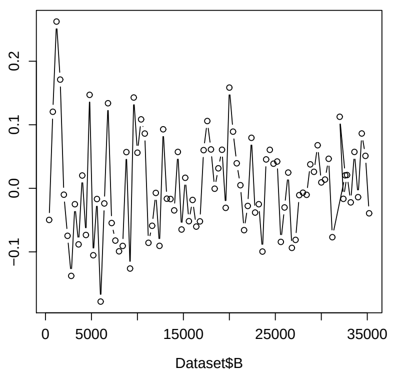
(a) $SE\left(\hat{\gamma_B}\right)$



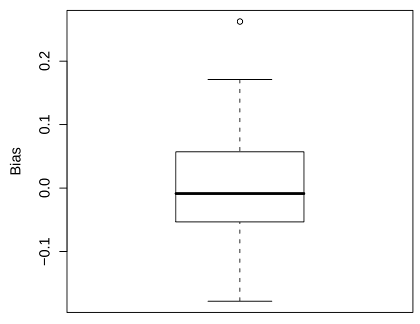
(b) $\hat{\gamma_B} = Var\left(\theta\right)$)



(c) Boxplot



(d) $Bi\hat{a}s_B = \hat{\theta^{*b}} - \hat{\theta}$



(e) Boxplot

Figure D.1: (a) Showing how the estimate of the ***standard error*** *in the bootstrap estimate of the variance of the mean,* $SE\left(\hat{\gamma_B}\right)$ *vary with an increase in number of bootstrap replicates,B. (b) The bootstrap estimate of the* ***variance*** *of the mean, (*$\hat{\gamma_B} = Var\left(\hat{\theta}\right)$*) and (d) the bootstrap estimate of the bias, (*$Bi\hat{a}s_B = \hat{\theta^{*b}} - \hat{\theta}$*) in the same experiment*

APPENDIX E

Statistical tests

E.1 Goodness of fit tests

A note on distributional measures and goodness of fit test Parametric statistics simplifies description of data, but recall that they require the assumption of normality of the data being investigated to hold. If it can be established that the data follows a normal distribution, then we can safely assume that a particular set of measurements (e.g. moments,variations) can be properly described by its mean and standard deviation for example, otherwise any conclusions drawn may be meaningless. So the first task was to ascertain this assumption of normality for the detection delays datasets.

Often transformation of the original data can be used to allow the use of parametric statistics, i.e. under a mathematical transformation (e.g. logarithm), the resulting data may be normally distributed [1]. These transformations can be viewed as entirely legitimate as they only change the scale on which the analysis is being done. And inverse transform can then be used to get to the original scales.

If normality test fails, for example in skewed or peaked distributions, other theoretical parametric models can be *fitted* for example , log-normal, gamma, logistic etc. To assess how well a particular model fits, firstly, a visual inspection of the frequency histogram with an overlaid plot of the desired distribution can be made, and secondly

[1]This data is then said to be lognormal

a goodness-of-fit test can be conducted to test the hypothesis that the data comes from a given distribution. In essence, the normality test is a special case of goodness-of-fit test.

A detailed discussion of goodness-of fit and normality can be seen in [131] [65] and [181].

For testing the normality of the detection delays data sets I used the tests below[2]. Each test defines a statistic and calculates a p-value. While one test would suffice, I considered all these test for completeness and comparative purposes. [229] discusses and compares the power of some of these standard normality tests procedures.

The lower half of table presents the results.

1. Anderson-Darling test [9], Statistic, A, is calculated as

$$A = -n - \frac{1}{n}\sum_{i=1}^{n}[2i-1][ln(p_{(i)}) + ln(1 - p_{(n-i+1)})] \tag{E.1}$$

   where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. Here, Φ is the cumulative distribution function of the standard normal distribution and $\overline{x}$ and s are mean and standard deviation of the data values. The p-value is computed from

$$Z = A(1.0 + 0.75/n + 2.25/n^2) \tag{E.2}$$

2. Shapiro-Wilks [205]. The statistic ,W , is calculated as

$$W = \Big(\sum_{i=1}^{n} a_i x(i)\Big) / \sum_{i=1}^{n}(x_i - \overline{x}) \tag{E.3}$$

   Where x_i are ordered sample values.

[2]As implemented in the R TM, Matlab TMand DATAPLOT TMsoftware packages .

3. Lillie test [149]. The statistic,D is calculated as $D = \max D^+, D^-$

$$D^+ = \max_{i=1,\dots,n} i/n - p_{(i)} \qquad D^- = \max_{i=1,\dots,n} p_{(i)} - (i-1)/n \tag{E.4}$$

where again $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. Here, Φ is the cumulative distribution function of the standard normal distribution. The p-value is computed from the distribution of the modified statistic $Z = D(\sqrt{n} - 0.01 + 0.85/\sqrt{n})$ as described in [221]

4. Jarque-Bera [125]. The statistic, JB , is calculated as

$$JB = \frac{n}{6}\Big(S^2 + \frac{(K-3)^3}{4}\Big) \tag{E.5}$$

Where S is the sample skewness , K sample kurtosis both defined as usual in terms of third and forth central moments μ_3, μ_4 i.e. as $S = \mu_3/\sigma^3$ and $K = \mu_4/\sigma^4$

5. Pearson test [49, 192]. The statistic. P, is calculated as

$$P = \sum (C_i - E_i)^2/E_i \tag{E.6}$$

Where C_i is the number of counted and E_i is the number of expected observations (under the hypothesis) in class i.

6. Cramer-von Mises test [8]. The test statistic, W , is calculated as

$$W = \frac{1}{12n} + \sum_{i=1}^{n} (p_{(i)} - \frac{2i-1}{2n}) \tag{E.7}$$

where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$, and Φ is the cumulative distribution function of the standard normal distribution.

I have also performed some data transformations and conducted normality tests on the resulting data, see appendix All test confirm that the normality assumption cannot really be made for the detection delays data.

For exploratory purposes , I have also fitted some theoretic parametric distributions to the data. The results are shown in appendices.

A note on distributional measures and goodness of fit test Parametric statistics simplifies description of data, but recall that they require the assumption of normality of the data being investigated to hold. If it can be established that the data follows a normal distribution, then we can safely assume that a particular set of measurements (e.g. moments,variations) can be properly described by its mean and standard deviation for example, otherwise any conclusions drawn may be meaningless. So the first task was to ascertain this assumption of normality for the detection delays datasets.

Often transformation of the original data can be used to allow the use of parametric statistics, i.e. under a mathematical transformation (e.g. logarithm), the resulting data may be normally distributed [3]. These transformations can be viewed as entirely legitimate as they only change the scale on which the analysis is being done. And inverse transform can then be used to get to the original scales.

If normality test fails, for example in skewed or peaked distributions, other theoretical parametric models can be *fitted* for example , log-normal, gamma, logistic etc. . To assess how well a particular model fits, firstly, a visual inspection of the frequency histogram with an overlaid plot of the desired distribution can be made, and secondly a goodness-of-fit test can be conducted to test the hypothesis that the data comes from

[3]This data is then said to be lognormal.

a given distribution. In essence, the normality test is a special case of goodness-of-fit test.

A detailed discussion of goodness-of fit and normality can be seen in [131] [65] and [181].

For testing the normality of the detection delays data sets I used the tests below[4]. Each test defines a statistic and calculates a p-value. While one test would suffice, I considered all these test for completeness and comparative purposes. [229] discusses and compares the power of some of these standard normality tests procedures.

The lower half of table presents the results.

1. Anderson-Darling test [9], Statistic, A, is calculated as

$$A = -n - \frac{1}{n}\sum_{i=1}^{n}[2i-1][ln(p_{(i)}) + ln(1-p_{(n-i+1)})] \tag{E.8}$$

   where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. Here, Φ is the cumulative distribution function of the standard normal distribution and $\overline{x}$ and s are mean and standard deviation of the data values. The p-value is computed from

$$Z = A(1.0 + 0.75/n + 2.25/n^2) \tag{E.9}$$

2. Shapiro-Wilks [205]. The statistic ,W , is calculated as

$$W = \Big(\sum_{i=1}^{n} a_i x(i)\Big) / \sum_{i=1}^{n}(x_i - \overline{x}) \tag{E.10}$$

   Where x_i are ordered sample values.

[4]As implemented in the R ™, Matlab ™and DATAPLOT ™software packages.

3. Lillie test [149]. The statistic,D is calculated as $D = \max D^+, D^-$

$$D^+ = \max_{i=1,\ldots,n} i/n - p_{(i)} \qquad D^- = \max_{i=1,\ldots,n} p_{(i)} - (i-1)/n \tag{E.11}$$

where again $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. Here, Φ is the cumulative distribution function of the standard normal distribution. The p-value is computed from the distribution of the modified statistic $Z = D(\sqrt{n} - 0.01 + 0.85/\sqrt{n})$ as described in [221]

4. Jarque-Bera [125]. The statistic, JB , is calculated as

$$JB = \frac{n}{6}\Big(S^2 + \frac{(K-3)^3}{4}\Big) \tag{E.12}$$

Where S is the sample skewness , K sample kurtosis both defined as usual in terms of third and forth central moments μ_3, μ_4 i.e. as $S = \mu_3/\sigma^3$ and $K = \mu_4/\sigma^4$

5. Pearson test [49, 192]. The statistic. P, is calculated as

$$P = \sum (C_i - E_i)^2/E_i \tag{E.13}$$

Where C_i is the number of counted and E_i is the number of expected observations (under the hypothesis) in class i.

6. Cramer-von Mises test [8]. The test statistic, W , is calculated as

$$W = \frac{1}{12n} + \sum_{i=1}^{n}(p_{(i)} - \frac{2i-1}{2n}) \tag{E.14}$$

where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$, and Φ is the cumulative distribution function of the standard normal distribution.

I have also performed some data transformations and conducted normality tests on the resulting data, see appendix All test confirm that the normality assumption cannot really be made for the detection delays data.

For exploratory purposes , I have also fitted some theoretic parametric distributions to the data. The results are shown in appendices.

DISTRIBUTIONAL NORMALITY TESTS										
ANDERSON-DARLING		$A=-n-\frac{1}{n}\sum_{i=1}^{n}[2i-1][ln(p_{(i)})+ln(1-p_{(n-i+1)})]$								
Statistic	A	26.99178	6.518426	11.87086	11.07163	4.86468	68.7629	11.86567	4.23203	43.77127
	P-value	3.160e-61	5.45e-16	1.80e-28	1.23e-26	4.85e-12	1.68e-132	1.85e-28	1.61e-10	3.00e-93
Conclusion		REJECT								
WILKSON-SHAPIRO										
Statistic	W	0.94143	0.97707	0.95221	0.96082	0.98741	0.84531	0.97045	0.98697	0.91408
	p-value	1.71e-25	6.40e-16	3.27e-23	4.30e-21	4.58e-11	5.90e-38	2.85e-18	2.57e-11	4.23e-30
Conclusion		REJECT								
SHAPIRO-FRANCIA		$W=\left(\sum_{i=1}^{n}a_ix(i)\right)/\sum_{i=1}^{n}(x_i-\overline{x})$								
Statistic	W	0.941533	0.977359	0.95234	0.96110	0.98803	0.84541	0.97064	0.98726	0.91364
	p-value	3.34e-23	1.45e-14	3.70e-21	3.17e-19	5.60e-10	2.60e-34	1.03e-16	2.21e-10	2.20e-27
Conclusion		REJECT								
LILLIE(KOLG-SMIR)		$D^{+}=\max_{i=1,\ldots,n} i/n-p_{(i)}, D^{-}=\max_{i=1,\ldots,n} p_{(i)}-(i-1)/n$								
Statistic	D	0.107311	0.040742	0.041497	0.04118	0.03484	0.14700	0.05665	0.034663	0.10399
	p-value	2.25e-53	5.61e-07	2.98e-07	3.89e-07	4.96e-05	2.10e-102	5.55e-14	5.61e-05	5.35e-50
Conclusion		REJECT								
JARQUE-BERA		$JB=\frac{n}{6}\left(S^2+\frac{(K-3)^3}{4}\right)$								

Statistic	X^2	391.5488	53.03562	453.9846	237.4310	40.43232	1920.713	72.68503	42.78078	589.0186
	p-value	0	3.04e-12	0	0	1.66e-09	0	1.11e-16	5.13e-10	0
Conclusion		REJECT								
PEARSON		$P = \sum(C_i - E_i)^2/E_i$								
Statistic	P	520.8587	231.6351	142.4349	166.3462	98.1617	625.0502	205.6223	108.9965	465.7974
	p-value	1.06e-86	5.15e-30	2.72e-14	2.53e-18	1.92e-07	6.05e-108	2.91e-25	5.07e-09	1.37e-75
Conclusion		REJECT								
CRAMER-VON MISES		$W = \frac{1}{12n} + \sum_{i=1}^{n}(p_{(i)} - \frac{2i-1}{2n})$								
Statistic	W	4.648875	0.655082	1.28574	1.29745	0.73091	10.99672	1.54436	0.58801	7.43111
	p-value	6.91e+51	1.35e-07	3.75e-10	3.70e-10	3.87e-08	Inf	6.43e-10	4.60e-07	7.29e+197
Conclusion		REJECT								

Table E.1: Showing results of a number of normality tests on all datasets for experiments where the agent numbers monitored was varied from 5 to 100. All normality tests reject the hypothesis that the data is normally distributed as evidenced by low p-values, i.e. $p - values \ll 0.05$

E.2 Hypothesis Tests

Regarding the hypothesis to be tested, we can consider a two sided test with the null hypothesis being that the *location parameters for the partial and full protocol datasets are equal*, and with the alternative hypothesis just being that *there is a difference between the location parameters* i.e.

$$H_0 : \mu_1 = \mu_2, H_1 : \mu_1 \neq \mu_2 \tag{E.15}$$

But a more relevant test is the one sided hypothesis test given that we suspect the partial protocol datasets to have lower location parameter than the full protocol datasets.

$$H_0 : \mu_1 = \mu_2, H_1 : \mu_1 \leq \mu_2 \tag{E.16}$$

i.e. the null hypothesis can be stated as: "*There is no difference in the location parameter for the partial protocol and full protocol datasets*. The corresponding alternative hypothesis can be stated as: " *The location parameter for the partial protocol dataset is less than that of the full protocol dataset*.

I ran the $\mathrm{wilcox.test}$ procedure with the data vectors from the two data sets and for each of the experiments when the agent numbers were varied from 10 through to a 100. Results are shown in Table E.2. The results show that the is a strong evidence across all experiments against the null hypothesis, as shown by the very low p-values, and by extension a strong evidence toward alternative hypothesis.

Wilcoxon rank sum test Data: Pproto.data and Fproto.data
10 agents experiments
W = 93638, p-value $< 2.2e-16$
alternative hypothesis: true location shift is less than 0
30 agents experiments
W = 26503.5, p-value $< 2.2e-16$
alternative hypothesis: true location shift is less than 0
50 agents experiments
W = 693222.5, p-value $< 2.2e-16$
alternative hypothesis: true location shift is less than 0
100 agents experiments
W = 1008896, p-value $< 2.2e-16$
alternative hypothesis: true location shift is less than 0

Table E.2: Showing results of the non parametric Wilcoxon hypothesis test for the partial and full protocol datasets for experiments with agent numbers varied from 10 to 100. The very low p-value are a strong evidence against accepting the null hypothesis and strong evidence for considering the alternative

E.3 Repeatability of experiments

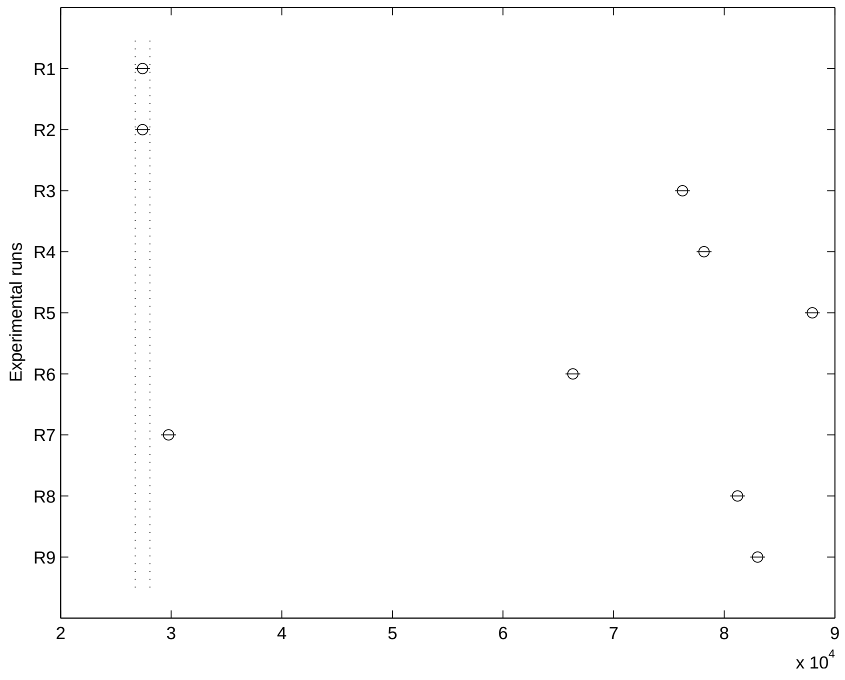
A note on repeatability of experiments and analysis of variance To check whether experiments were repeatable, for every experiment, a number of runs were scheduled. For example Figure E.1 shows the box plots of the data for experimental runs for the 10 and 70 agents experiments. The corresponding, Table E.3 shows results of the Kruskal-Wallis test (a non parametric equivalent of the ANOVA test) for these experimental runs. The related results for the multiple comparison procedure are shown in Figure E.1

KRUSKAL-WALLIS ANOVA TABLE					
SOURCE	SS	DF	MS	CHI-SQ	PROB > CHI-SQ
10 AGENTS					
Columns	2.1750e+09	1	2.1750e+09	2.2230e+03	0

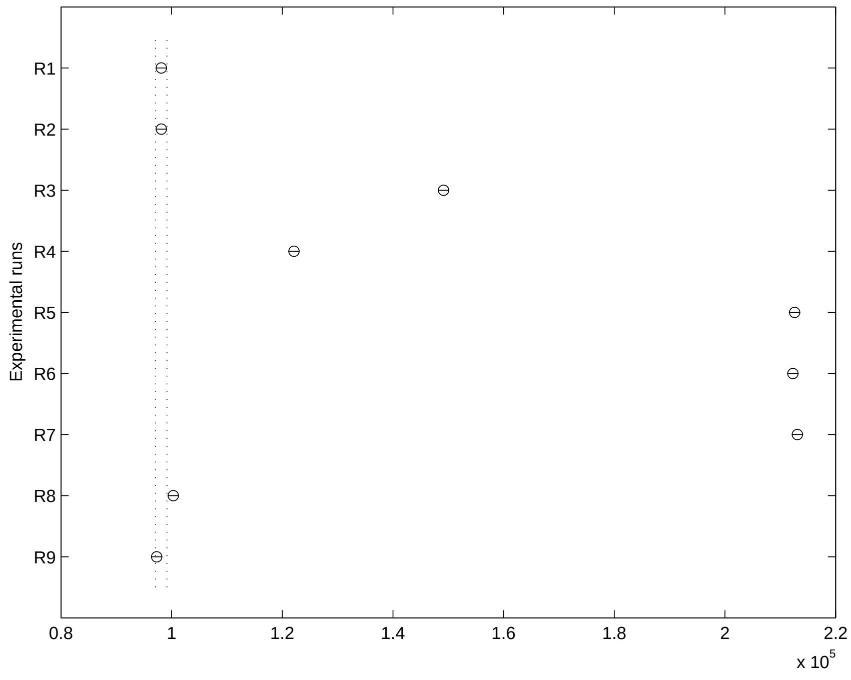
Error	1.1760e+09	3424	3.4347e+05		
Total	3.3510e+09	3425			
70 AGENTS					
Columns	2.4878e+08	1	2.4878e+08	254.2751	0
Error	3.1023e+09	3424	9.0604e+05		
Total	3.3510e+09	3425			

Table E.3: Results for the non-parametric anova using Kruskal-Wallis test

The sensitive multiple comparisons test results as shown in Figure E.1 and the kruskal-wallis tests results in Table E.3 do show differences in the location parameter across the experimental runs, but the boxplots indicate that the differences are reasonable.



(a) 10 experimental runs



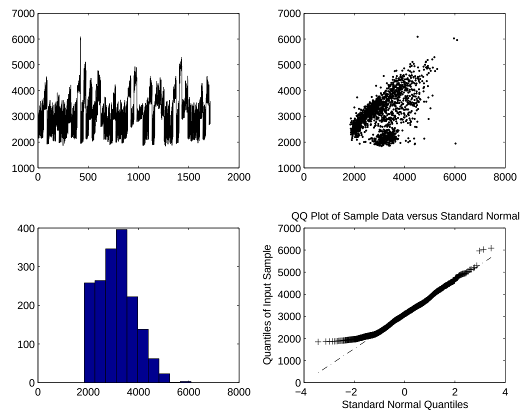
(b) 70 agent experimental runs

Figure E.1: Showing plot for the multiple comparison tests for 9 experimental runs for the 10 and 70 agent full protocol experiment

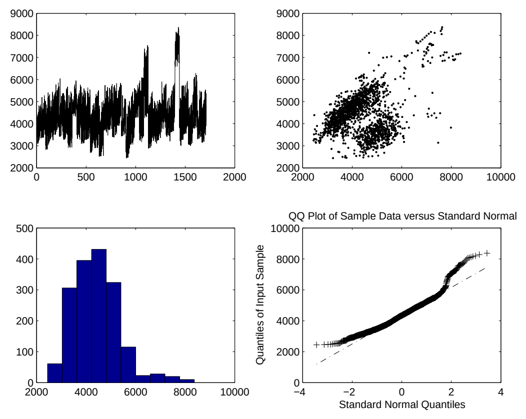
APPENDIX F

All summary statistics

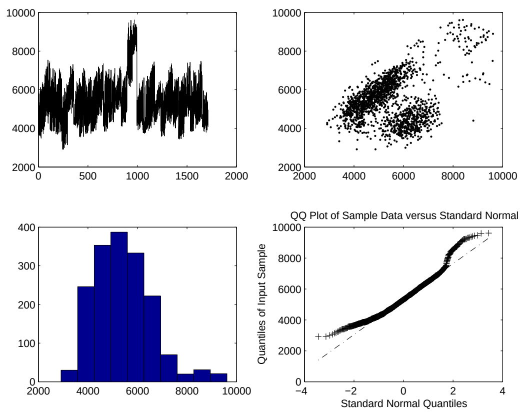
F.1 Partial Protocol data sets



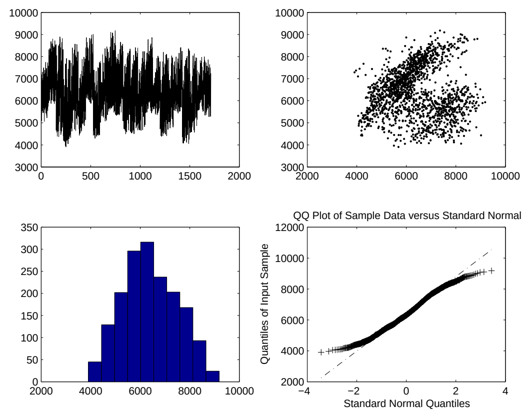
(a) 10 agent experiment



(b) 15 agent experiment

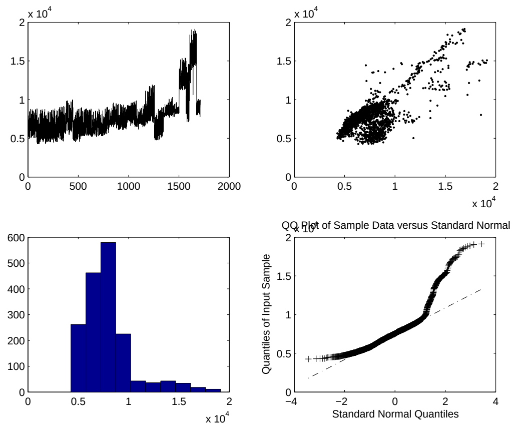


(c) 20 agent experiment

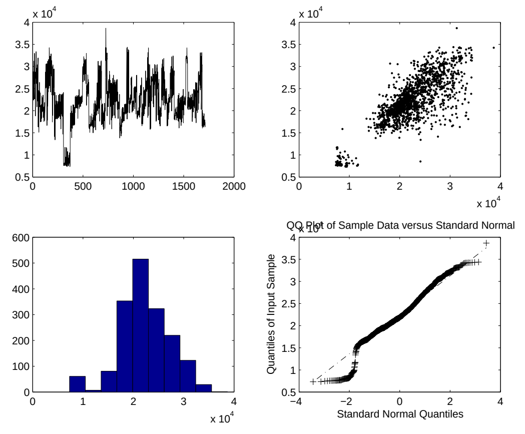


(d) 25 agent experiment

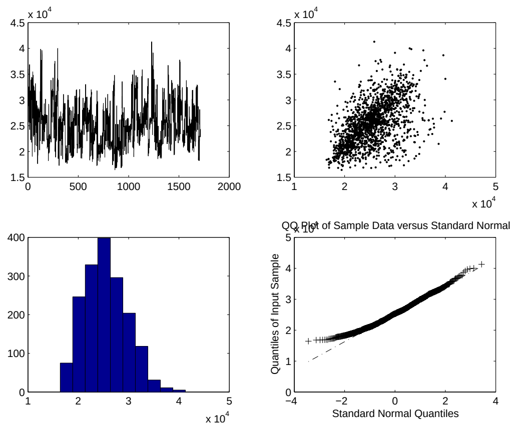
Figure F.1: Figures (a)-(d) show 4-plots for the purposes of exploratory data analysis for centralised experiments using the partial protocol scheme where agent numbers were varied from 10 through to 25



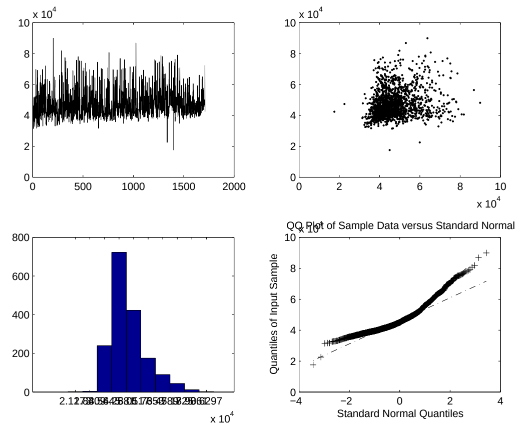
(a) 30 agent experiment



(b) 40 agent experiment



(c) 50 agent experiment



(d) 100 agent experiment

Figure F.2: Figures (a)-(d) show 4-plots for the purposes of exploratory data analysis for centralised experiments using the partial protocol scheme where agent numbers were varied from 30 through to 100

F.2 Full Protocol datasets

```
                        SUMMARY STATISTICS
                      NUMBER OF OBSERVATIONS =     5317
***********************************************************************
*        LOCATION MEASURES         *       DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.3564000E+04  *  RANGE        =   0.2470000E+04  *
*  MEAN         =   0.2921928E+04  *  STAND. DEV.  =   0.4257777E+03  *
*  MIDMEAN      =   0.2882001E+04  *  AV. AB. DEV. =   0.3228591E+03  *
*  MEDIAN       =   0.2795000E+04  *  MINIMUM      =   0.2329000E+04  *
*               =                  *  LOWER QUART. =   0.2597000E+04  *
*               =                  *  LOWER HINGE  =   0.2597000E+04  *
*               =                  *  UPPER HINGE  =   0.3158000E+04  *
*               =                  *  UPPER QUART. =   0.3158000E+04  *
*               =                  *  MAXIMUM      =   0.4799000E+04  *
***********************************************************************
*       RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =   0.4246227E+00  *  ST. 3RD MOM. =   0.1471157E+01  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.5648685E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.3380893E+03  *
*               =                  *  UNIFORM PPCC =   0.9070824E+00  *
*               =                  *  NORMAL  PPCC =   0.9325916E+00  *
*               =                  *  TUK -.5 PPCC =   0.6920604E+00  *
*               =                  *  CAUCHY  PPCC =   0.2227653E+00  *
***********************************************************************
```

(a) 5 Agent Experiment

```
                        SUMMARY STATISTICS
                      NUMBER OF OBSERVATIONS =    16910
***********************************************************************
*        LOCATION MEASURES         *       DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.7022500E+04  *  RANGE        =   0.6155000E+04  *
*  MEAN         =   0.5628141E+04  *  STAND. DEV.  =   0.9632474E+03  *
*  MIDMEAN      =   0.5644754E+04  *  AV. AB. DEV. =   0.6999647E+03  *
*  MEDIAN       =   0.5395000E+04  *  MINIMUM      =   0.3945000E+04  *
*               =                  *  LOWER QUART. =   0.4950000E+04  *
*               =                  *  LOWER HINGE  =   0.4950000E+04  *
*               =                  *  UPPER HINGE  =   0.6074000E+04  *
*               =                  *  UPPER QUART. =   0.6074000E+04  *
*               =                  *  MAXIMUM      =   0.1010000E+05  *
***********************************************************************
*       RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =   0.7945901E+00  *  ST. 3RD MOM. =   0.1405107E+01  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.5350581E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.5873189E+03  *
*               =                  *  UNIFORM PPCC =   0.9122940E+00  *
*               =                  *  NORMAL  PPCC =   0.9451001E+00  *
*               =                  *  TUK -.5 PPCC =   0.6660264E+00  *
*               =                  *  CAUCHY  PPCC =   0.1550779E+00  *
***********************************************************************
```

(b) 10 Agent Experiment

Figure F.3: Showing summary statistics for 5, 10 agent experiments

```
                       SUMMARY STATISTICS
                     NUMBER OF OBSERVATIONS =    21424
***********************************************************************
*        LOCATION MEASURES         *       DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.1336450E+05  *  RANGE        =   0.7295000E+04  *
*  MEAN         =   0.1237764E+05  *  STAND. DEV.  =   0.9852908E+03  *
*  MIDMEAN      =   0.1238323E+05  *  AV. AB. DEV. =   0.8050829E+03  *
*  MEDIAN       =   0.1214500E+05  *  MINIMUM      =   0.9717000E+04  *
*               =                  *  LOWER QUART. =   0.1161900E+05  *
*               =                  *  LOWER HINGE  =   0.1161900E+05  *
*               =                  *  UPPER HINGE  =   0.1321100E+05  *
*               =                  *  UPPER QUART. =   0.1321100E+05  *
*               =                  *  MAXIMUM      =   0.1701200E+05  *
***********************************************************************
*       RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =  -0.3532879E-01  *  ST. 3RD MOM. =   0.6062595E+00  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.3205218E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.3354115E+03  *
*               =                  *  UNIFORM PPCC =   0.9707130E+00  *
*               =                  *  NORMAL  PPCC =   0.9767389E+00  *
*               =                  *  TUK -.5 PPCC =   0.6588120E+00  *
*               =                  *  CAUCHY  PPCC =   0.1502470E+00  *
***********************************************************************
```

(a) 20 Agent Experiment

```
                       SUMMARY STATISTICS
                     NUMBER OF OBSERVATIONS =    23314
***********************************************************************
*        LOCATION MEASURES         *       DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.1715650E+05  *  RANGE        =   0.1207500E+05  *
*  MEAN         =   0.1760411E+05  *  STAND. DEV.  =   0.1411312E+04  *
*  MIDMEAN      =   0.1762307E+05  *  AV. AB. DEV. =   0.1167766E+04  *
*  MEDIAN       =   0.1712900E+05  *  MINIMUM      =   0.1111900E+05  *
*               =                  *  LOWER QUART. =   0.1651300E+05  *
*               =                  *  LOWER HINGE  =   0.1651300E+05  *
*               =                  *  UPPER HINGE  =   0.1904900E+05  *
*               =                  *  UPPER QUART. =   0.1904900E+05  *
*               =                  *  MAXIMUM      =   0.2319400E+05  *
***********************************************************************
*       RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =  -0.2015251E+00  *  ST. 3RD MOM. =   0.4725470E+00  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.2211800E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.4984297E+03  *
*               =                  *  UNIFORM PPCC =   0.9714868E+00  *
*               =                  *  NORMAL  PPCC =   0.9635677E+00  *
*               =                  *  TUK -.5 PPCC =   0.6231364E+00  *
*               =                  *  CAUCHY  PPCC =   0.1427646E+00  *
***********************************************************************
```

(b) 30 Agent Experiment

Figure F.4: Showing summary statistics for 20, 30 agent experiments

```
                      SUMMARY STATISTICS
                    NUMBER OF OBSERVATIONS =    27413
***********************************************************************
*        LOCATION MEASURES         *       DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.2415050E+05  *  RANGE        =   0.2167700E+05  *
*  MEAN         =   0.2338939E+05  *  STAND. DEV.  =   0.2435642E+04  *
*  MIDMEAN      =   0.2341896E+05  *  AV. AB. DEV. =   0.2120906E+04  *
*  MEDIAN       =   0.2287900E+05  *  MINIMUM      =   0.1331200E+05  *
*               =                  *  LOWER QUART. =   0.2128000E+05  *
*               =                  *  LOWER HINGE  =   0.2128000E+05  *
*               =                  *  UPPER HINGE  =   0.2571200E+05  *
*               =                  *  UPPER QUART. =   0.2571200E+05  *
*               =                  *  MAXIMUM      =   0.3498900E+05  *
***********************************************************************
*       RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =  -0.1445579E-01  *  ST. 3RD MOM. =   0.3875467E+00  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.2782674E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.4959662E+03  *
*               =                  *  UNIFORM PPCC =   0.9786684E+00  *
*               =                  *  NORMAL  PPCC =   0.9678799E+00  *
*               =                  *  TUK -.5 PPCC =   0.6320242E+00  *
*               =                  *  CAUCHY  PPCC =   0.1418798E+00  *
***********************************************************************
```

(a) 40 Agent Experiment

```
                      SUMMARY STATISTICS
                    NUMBER OF OBSERVATIONS =    31882
***********************************************************************
*        LOCATION MEASURES         *       DISPERSION MEASURES        *
***********************************************************************
*  MIDRANGE     =   0.3152500E+05  *  RANGE        =   0.2075400E+05  *
*  MEAN         =   0.2918180E+05  *  STAND. DEV.  =   0.2817885E+04  *
*  MIDMEAN      =   0.2919943E+05  *  AV. AB. DEV. =   0.2275700E+04  *
*  MEDIAN       =   0.2954800E+05  *  MINIMUM      =   0.2114800E+05  *
*               =                  *  LOWER QUART. =   0.2748400E+05  *
*               =                  *  LOWER HINGE  =   0.2748400E+05  *
*               =                  *  UPPER HINGE  =   0.3137600E+05  *
*               =                  *  UPPER QUART. =   0.3137600E+05  *
*               =                  *  MAXIMUM      =   0.4190200E+05  *
***********************************************************************
*       RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
***********************************************************************
*  AUTOCO COEF  =   0.1621292E+00  *  ST. 3RD MOM. =  -0.4408322E+00  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.2420063E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.3858649E+03  *
*               =                  *  UNIFORM PPCC =   0.9809257E+00  *
*               =                  *  NORMAL  PPCC =   0.9799967E+00  *
*               =                  *  TUK -.5 PPCC =   0.6121349E+00  *
*               =                  *  CAUCHY  PPCC =   0.1181198E+00  *
***********************************************************************
```

(b) 50 Agent Experiment

Figure F.5: Showing summary statistics for 40, 50 agent experiments

```
                    SUMMARY STATISTICS
                  NUMBER OF OBSERVATIONS =    31897
***********************************************************************
*       LOCATION MEASURES         *       DISPERSION MEASURES         *
***********************************************************************
*  MIDRANGE     =   0.3425250E+05  *  RANGE        =   0.2189700E+05  *
*  MEAN         =   0.3462034E+05  *  STAND. DEV.  =   0.3305890E+04  *
*  MIDMEAN      =   0.3464385E+05  *  AV. AB. DEV. =   0.2619900E+04  *
*  MEDIAN       =   0.3497700E+05  *  MINIMUM      =   0.2330400E+05  *
*               =                  *  LOWER QUART. =   0.3284000E+05  *
*               =                  *  LOWER HINGE  =   0.3284000E+05  *
*               =                  *  UPPER HINGE  =   0.3705400E+05  *
*               =                  *  UPPER QUART. =   0.3705400E+05  *
*               =                  *  MAXIMUM      =   0.4520100E+05  *
***********************************************************************
*      RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES       *
***********************************************************************
*  AUTOCO COEF  =   0.2307196E+00  *  ST. 3RD MOM. =  -0.5565903E+00  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.2742768E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.3651406E+03  *
*               =                  *  UNIFORM PPCC =   0.9734833E+00  *
*               =                  *  NORMAL  PPCC =   0.9817482E+00  *
*               =                  *  TUK -.5 PPCC =   0.6199930E+00  *
*               =                  *  CAUCHY  PPCC =   0.1137837E+00  *
***********************************************************************
```

(a) 40 Agent Experiment

```
                    SUMMARY STATISTICS
                  NUMBER OF OBSERVATIONS =    16477

***********************************************************************
*       LOCATION MEASURES         *       DISPERSION MEASURES         *
***********************************************************************
*  MIDRANGE     =   0.4061300E+05  *  RANGE        =   0.3793400E+05  *
*  MEAN         =   0.4032761E+05  *  STAND. DEV.  =   0.4198260E+04  *
*  MIDMEAN      =   0.4054351E+05  *  AV. AB. DEV. =   0.3201970E+04  *
*  MEDIAN       =   0.4019000E+05  *  MINIMUM      =   0.2164600E+05  *
*               =                  *  LOWER QUART. =   0.3798700E+05  *
*               =                  *  LOWER HINGE  =   0.3798700E+05  *
*               =                  *  UPPER HINGE  =   0.4299400E+05  *
*               =                  *  UPPER QUART. =   0.4299500E+05  *
*               =                  *  MAXIMUM      =   0.5958000E+05  *
***********************************************************************
*      RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES       *
***********************************************************************
*  AUTOCO COEF  =   0.2472570E+00  *  ST. 3RD MOM. =   0.2060684E+00  *
*               =   0.0000000E+00  *  ST. 4TH MOM. =   0.4255906E+01  *
*               =   0.0000000E+00  *  ST. WILK-SHA =  -0.1803309E+03  *
*               =                  *  UNIFORM PPCC =   0.9556414E+00  *
*               =                  *  NORMAL  PPCC =   0.9869096E+00  *
*               =                  *  TUK -.5 PPCC =   0.7210979E+00  *
*               =                  *  CAUCHY  PPCC =   0.1933749E+00  *
***********************************************************************
```

(b) 50 Agent Experiment

Figure F.6: Showing summary statistics for 40, 50 agent experiments

```
                      SUMMARY STATISTICS
                    NUMBER OF OBSERVATIONS =    20522

***********************************************************************
 *       LOCATION MEASURES         *       DISPERSION MEASURES        *
 **********************************************************************
 *  MIDRANGE     =   0.4940700E+05  *  RANGE        =   0.6254400E+05  *
 *  MEAN         =   0.5080186E+05  *  STAND. DEV.  =   0.7051735E+04  *
 *  MIDMEAN      =   0.5052490E+05  *  AV. AB. DEV. =   0.5334674E+04  *
 *  MEDIAN       =   0.5079550E+05  *  MINIMUM      =   0.1813500E+05  *
 *               =                  *  LOWER QUART. =   0.4730225E+05  *
 *               =                  *  LOWER HINGE  =   0.4730300E+05  *
 *               =                  *  UPPER HINGE  =   0.5537600E+05  *
 *               =                  *  UPPER QUART. =   0.5537675E+05  *
 *               =                  *  MAXIMUM      =   0.8067900E+05  *
 **********************************************************************
 *      RANDOMNESS MEASURES        *     DISTRIBUTIONAL MEASURES      *
 **********************************************************************
 *  AUTOCO COEF  =   0.3237467E+00  *  ST. 3RD MOM. =  -0.5456704E+00  *
 *               =   0.0000000E+00  *  ST. 4TH MOM. =   0.3907562E+01  *
 *               =   0.0000000E+00  *  ST. WILK-SHA =  -0.2048621E+03  *
 *               =                  *  UNIFORM PPCC =   0.9512460E+00  *
 *               =                  *  NORMAL  PPCC =   0.9875441E+00  *
 *               =                  *  TUK -.5 PPCC =   0.7004446E+00  *
 *               =                  *  CAUCHY  PPCC =   0.1686039E+00  *
 **********************************************************************
```

Figure F.7: Showing detection delays summary statistics for the 5 agent experiment

APPENDIX G

Partial Protocol experimental runs results

To check repeatability, for each number of agents several experimental runs were made. Tables below presents results for example for the 70 agents experiments.

The Bootstrap										
Statistic		Experimental Runs								
		1	2	3	4	5	6	7	8	9
$\hat{\theta}$		40434.99	40436.47	45425.46	42701.33	52765.26	52295.44	52322.59	40503.44	40186.41
Confidence Intervals										
BC_a	**blo**	40385.82	40391.19	45338.68	42613.78	52667.94	52210.67	52234.37	40446.68	40135.91
	bhi	40482.52	40480.54	45517.59	42778.27	52876.83	52388.56	52410.85	40559.74	40237.42
	zo	-0.01	0.02	0.03	-0.01	-0.01	0.00	0.00	-0.03	0.01
	ahat	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00
Percentile	**phi**	40365.76	40387.85	45316.20	42643.30	52668.22	52218.43	52228.84	40457.01	40127.81
	plo	40501.04	40486.53	45551.85	42757.99	52855.40	52373.52	52410.31	40552.70	40240.10
Percentile-t	**p-lo**	40384.81	40389.05	45333.98	42622.16	52665.52	52206.22	52237.56	40448.84	40136.19
	pthi	40482.82	40482.14	45516.86	42780.11	52862.93	52382.95	52402.87	40565.09	40237.86
Hybrid	**hblo**	40390.66	40393.33	45335.93	42620.95	52668.58	52207.82	52238.48	40442.07	40133.11
	hbhi	40481.26	40482.77	45521.42	42771.81	52867.27	52380.03	52404.50	40557.23	40236.27

Table G.1: Bootstrap : showing parameter, confidence interval estimates,errors for 70 agent experimental runs, where θ is the sample mean

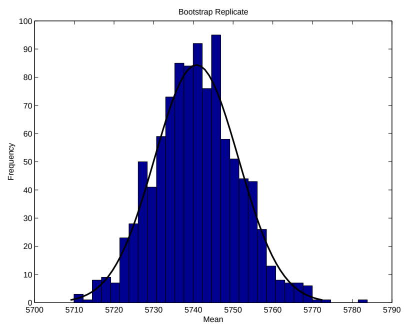
The Bootstrap										
Statistic		Experimental Runs								
		1	2	3	4	5	6	7	8	9
$\hat{\theta}$		29200.66	29201.13	29629.02	29121.28	29008.43	29342.12	37147.58	37146.98	36085.56
Confidence Intervals										
BC_a	**blo**	29163.56	29163.67	29580.78	29078.53	28970.18	29298.02	37094.76	37095.10	36016.50
	bhi	29242.54	29242.42	29680.17	29157.78	29047.69	29384.65	37196.58	37196.17	36162.14
	zo	0.04	0.03	-0.01	-0.03	-0.02	-0.01	-0.05	-0.04	0.04
	ahat	0.00	0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00
Percentile	**phi**	29156.38	29162.64	29588.56	29092.97	28969.07	29299.46	37079.24	37083.89	36004.01
	plo	29249.72	29241.01	29673.55	29149.41	29046.98	29388.92	37214.09	37212.18	36162.69
Percentile-t	**p-lo**	29163.44	29160.02	29578.14	29080.10	28968.12	29295.25	37094.16	37094.55	36017.61
	pthi	29239.90	29239.09	29676.95	29160.74	29047.30	29385.12	37197.35	37197.19	36149.10
Hybrid	**hblo**	29163.65	29161.73	29577.71	29080.22	28964.72	29298.59	37095.36	37094.35	36012.22
	hbhi	29242.35	29242.69	29679.13	29161.21	29044.75	29383.65	37199.09	37196.30	36150.14

Table G.2: Bootstrap : showing parameter and confidence interval estimates and errors for 50 agent experimenta1 runs

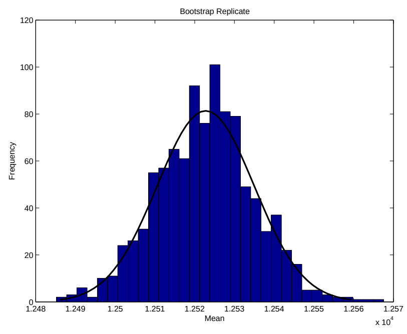
Figure G.3 below show density distributions of the bootstrap replicate of the men for the 9 experiments for the 70 agent centralised experiments. QQ plots of the replicates are also presented.

Figure G.2 shows QQ plot of the replicate for the 9 experiments for the 70 agent centralised experiments. these plots suggests that the replicates are normally distributed [1]

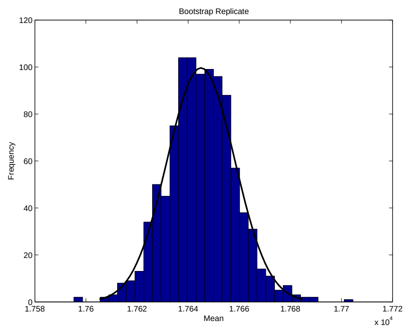
[1] This is not surprising given the central limit theorem.



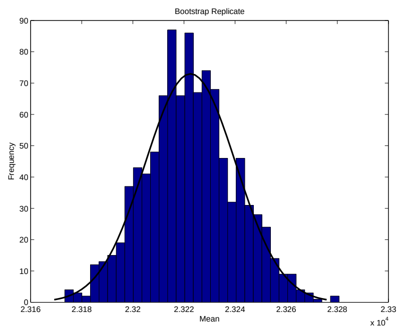
(a) RUN1



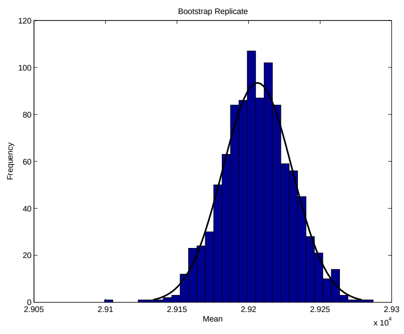
(b) RUN2



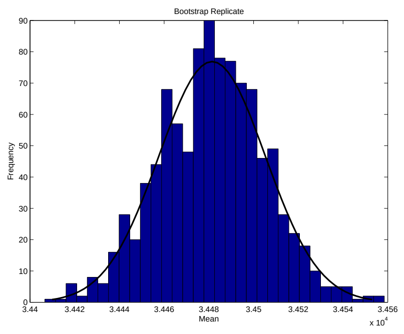
(c) RUN3



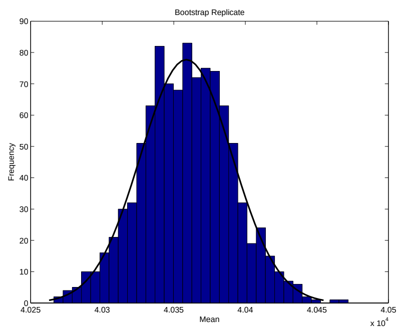
(d) RUN4



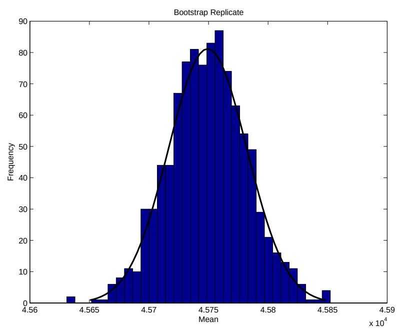
(e) RUN5



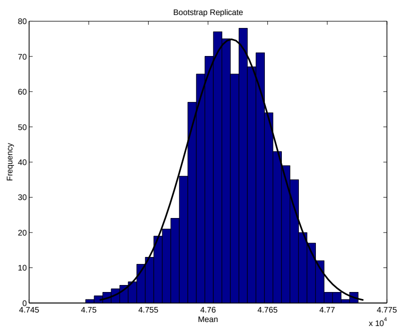
(f) RUN6



(g) RUN7

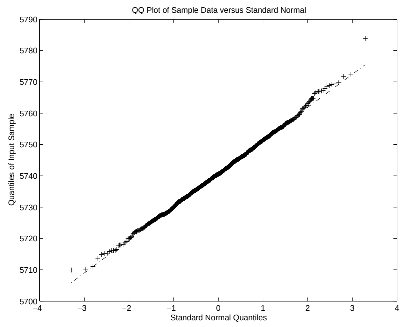


(h) RUN8

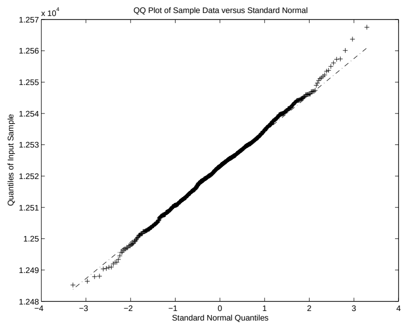


(i) RUN9

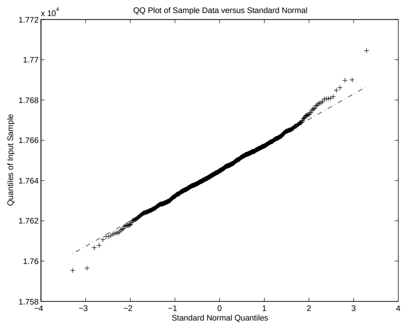
Figure G.1: Bootstrap replicates for each of the 9 70 agents experiments: Showing distribution of the mean



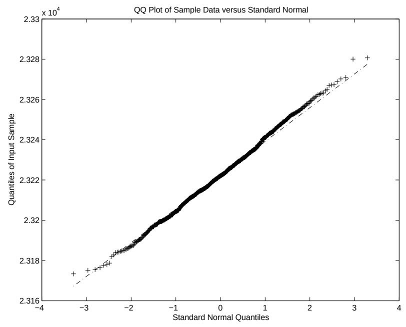
(a) RUN1



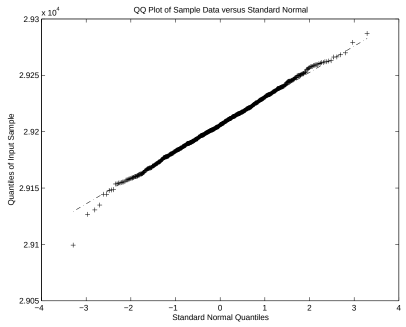
(b) RUN2



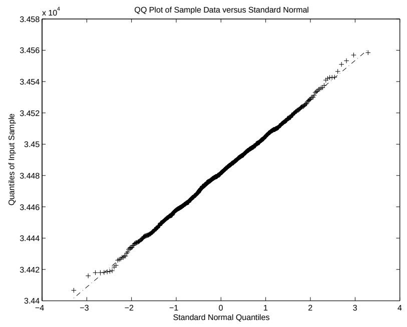
(c) RUN3



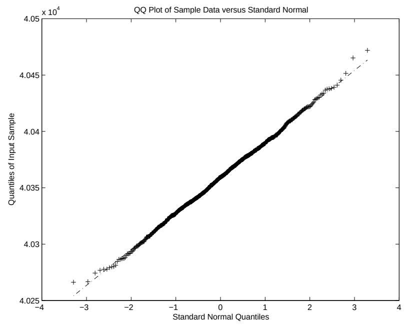
(d) RUN4



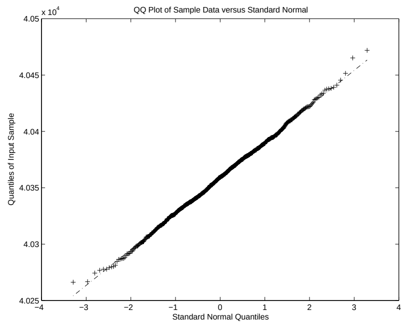
(e) RUN5



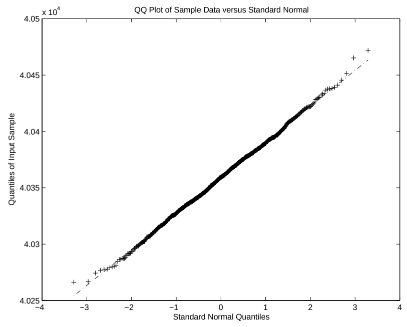
(f) RUN6



(g) RUN7



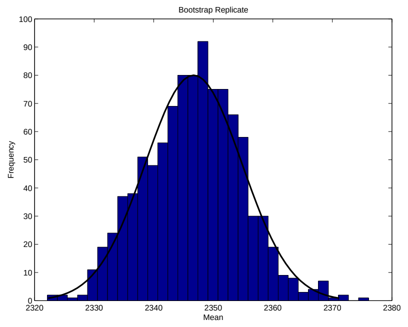
(h) RUN8



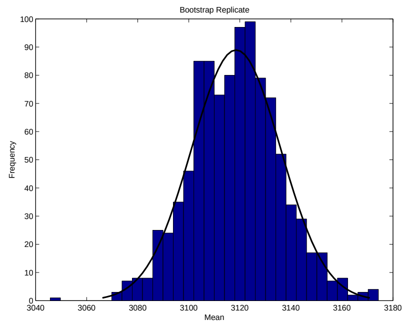
(i) RUN9

Figure G.2: Bootstrap replicates for each of the 9 70 agents experiments: Showing qqplots

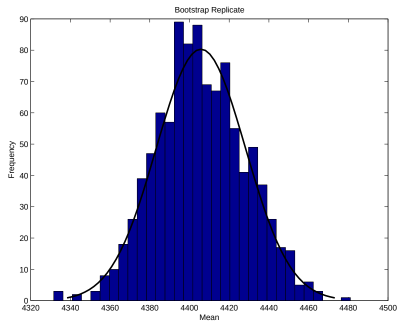
G.1 Resampling, Bootstrap confidence intervals



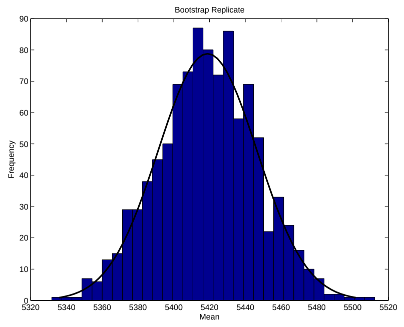
(a) RUN1



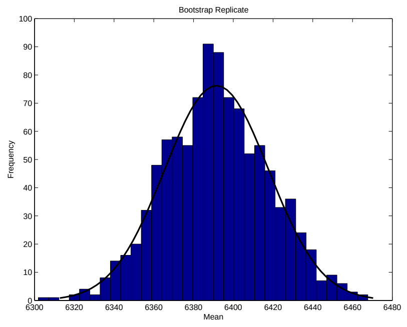
(b) RUN2



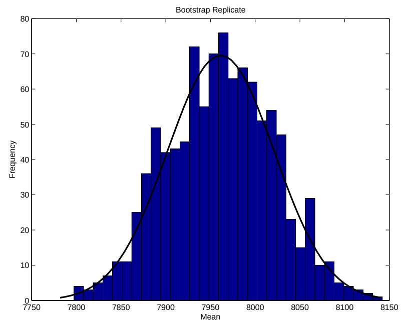
(c) RUN3



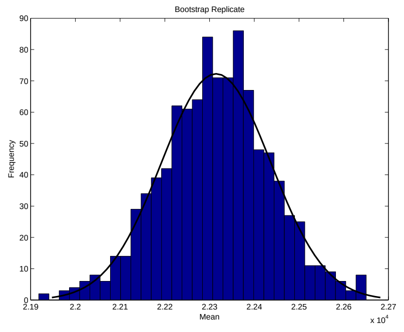
(d) RUN4



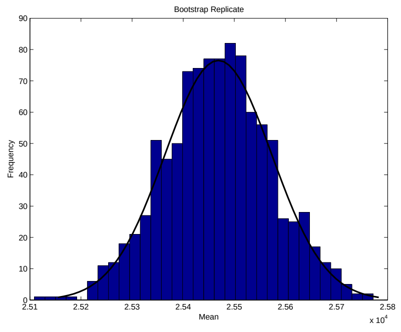
(e) RUN5



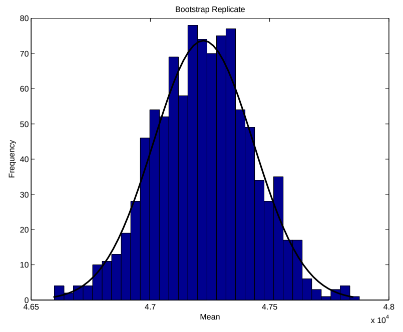
(f) RUN6



(g) RUN7

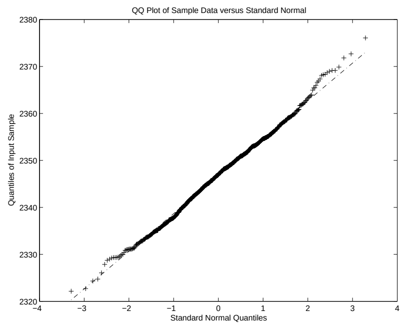


(h) RUN8

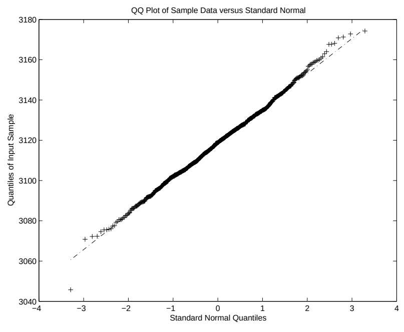


(i) RUN9

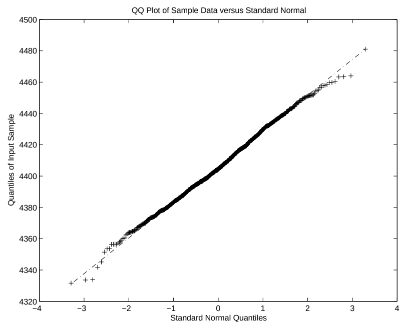
Figure G.3: Bootstrap replicates for each of the 9 70 agents experiments: Showing distribution of the mean



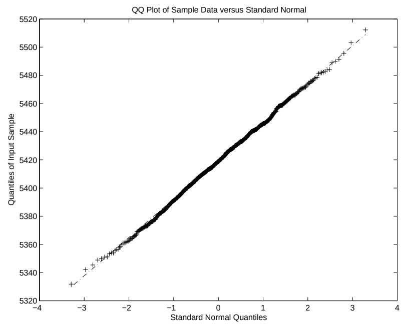
(a) RUN1



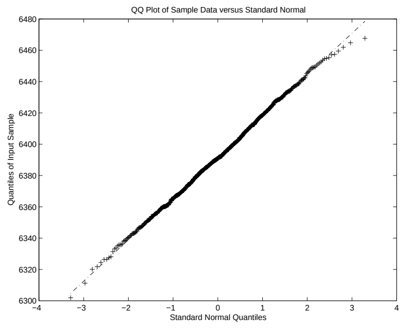
(b) RUN2



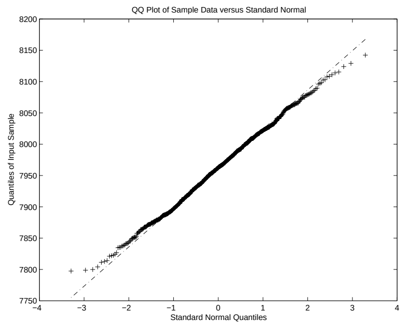
(c) RUN3



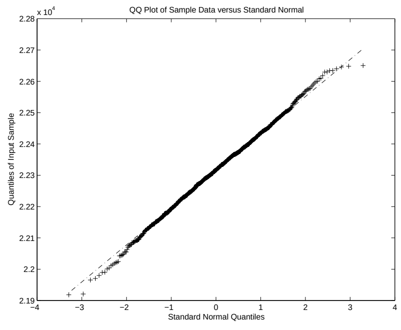
(d) RUN4



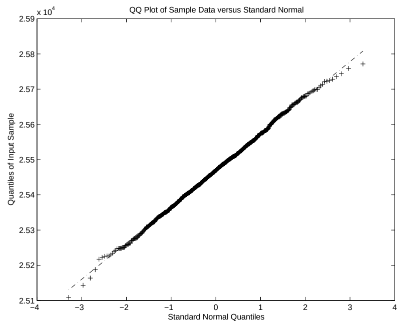
(e) RUN5



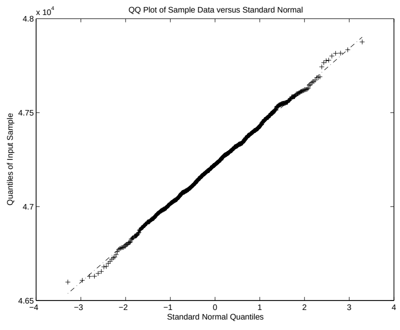
(f) RUN6



(g) RUN7



(h) RUN8



(i) RUN9

Figure G.4: Bootstrap replicates for each of the 9 70 agents experiments: Showing qqplots of the mean

APPENDIX H

Network latency and Synchronisation

The notion of time in distributed systems and the resulting inherent limitations of a distributed system relating in particular to the absence of a common (global) clock and the difficulty in reasoning about *temporal ordering* of events are well studied in distributed systems research with early work published in [146].

The area of clock synchronisation is also well researched with early notable work done by [147] [148, 62, 63, 157].

To address these problems, a number of algorithms for clock synchronisation have also been published in [106, 152, 218, 137, 235],

Extended surveys of these algorithms and synchronisation protocols can be seen in [208] and [195].

In the paragraphs that follow I only give a brief overview of these topics, sufficient only in providing context for this research when dealing with time and synchronisation related issues while conducting experiments in the distributed setting. Full details can be seen in the original papers and surveys mentioned above.

Overview In *general* when dealing with time in a distributed system;

- We may need to know the time some *event* happened on a specific node. For this, one approach is to synchronise that node's clock with some external *au-*

thoritative source of time. The main issue with this approach is to consider how difficult it is to achieve this synchronisation.

- We may need to know the time interval, or *relative order*, between two events that happened on different nodes.
  - The observation here is that, if their clocks are synchronised to some *known* degree of accuracy, we can measure time relative to each local clock. The issue here is whether this (accuracy) is always consistent.
- We cannot ignore the network's *unpredictability*.

To relate all these issues to the experiments I conducted in the distributed setting , I was specifically interested in determining *detection delays* as discussed throughout this thesis, but this time with the observation delegated to a remote node, the controller. The *event*, (termination), occurs *locally* and is detected *remotely*.

Generally dealing with time in distributed systems requires consideration of mainly **three** aspects; *physical clocks*, *coordinated universal time* and *synchronisation*. Since there is no common clock in this case, one standard approach used is to employ atomic clocks to minimise clock drift and synchronise with time servers that have *coordinated universal time*, UTC receivers, to try to compensate for unpredictable network delays.

H.1 Clock synchronisation algorithms

A large number of algorithms for clock synchronisation have been proposed in the literature. In general, because of the variable and unknown communication delays between processors, there are limits imposed on the extent to which processor local clocks can be synchronised. In addition, there is need to consider failures, and also complications do arise especially when arbitrary failures are considered. Therefore there has been work done in these areas. In particular a few theoretical results are known that study the limitations of clock synchronisation under different system models, e.g. [152, 79, 147].

These clock synchronisation algorithms discussed in the literature differ from each other in their assumptions about the *clock hardware*, *network topology*, and *failure models*. The algorithms take either a software or a hardware centric view of clock synchronisation [195]. All these algorithms provide internal clock synchronisation. External synchronisation is provided if one of the clocks is considered as the external, real-time reference.

The basic idea of software synchronisation algorithms is that each processor periodically *corrects* its local clock value according to the values of other clocks it receives through *message passing*. [195] discusses various classes of software based synchronisation protocols, namely, *convergence function with averaging* exemplified by [147, 152], *convergence function without averaging* [218], *consistency-based* [145].

For the purposes of experiments described in this chapter regarding clock synchronisation I used a software based protocol, NTP, (discussed below) widely implemented and available in UNIX based operating systems.

Network Time Protocol, NTP [166]is example of a *distributed algorithm* for time synchronisation is the famous

NTP has been implemented as an internet protocol [1] [2]. The protocol can be used to synchronise clocks on a *packet switched* network with variable latency. It uses the User Datagram Protocol, UDP [193], at the transport layer and has been designed

[1]NTP is documented in the standard internet protocol request for comments (RFC) document RFC1305)

[2]And also refers to a program (ntp daemon with utilities.) that implements the protocol and controls the computer clock.

particularly to resist the effects of variable latency.

NTP utilises a hierarchical network of servers, with *primary servers* connected directly to a time source, and *secondary servers* connected to the primary servers in a hierarchy. Servers higher up are presumed to be more accurate than at lower levels.

NTP uses Marzullo's algorithm [158, 157] with the UTC [3] time scale. The more recent implementation of NTP, NTPv4 [165] is reported to be capable of maintaining time to within 10 ms over the public internet, and can achieve accuracies of 200 μs or better in local area networks under ideal conditions [166].

NTP provides several synchronisation modes, namely; *multicast mode*, used mainly in local area networks; *procedure call mode*, which provide high accuracy and mainly used in file servers, and a *symmetric mode* where there is an exchange of detailed messages and history is maintained.

H.2 Network latency experiments

As discussed, the physical clocks on the cluster nodes were synchronised using the NTP protocol as described above [4], hence we assumed common time and negligible drift. To determine network latency experienced by control messages in the distributed setting, an experiment was also conducted.

Figure 10.2 presents an event diagram for the procedure followed when determining the distribution of the network latency and the clock drift. The experimental setup and an algorithm as suggested by the diagram is as follows; At various points in the running of overall experiments, a local time t_{11} is recorded and timer started and a *ping*

[3]High-precision atomic time standard.

[4]And the experiments conducted in the distributed setting did not span exceedingly longs period to warrant concern about clock drifts.

message sent from node $N1$ to $N2$. On arrival, local time t_{22} [5] is recorded and a *ping* message sent back to node $N1$. On arrival at node $N1$ local time t_{13} is recorded.

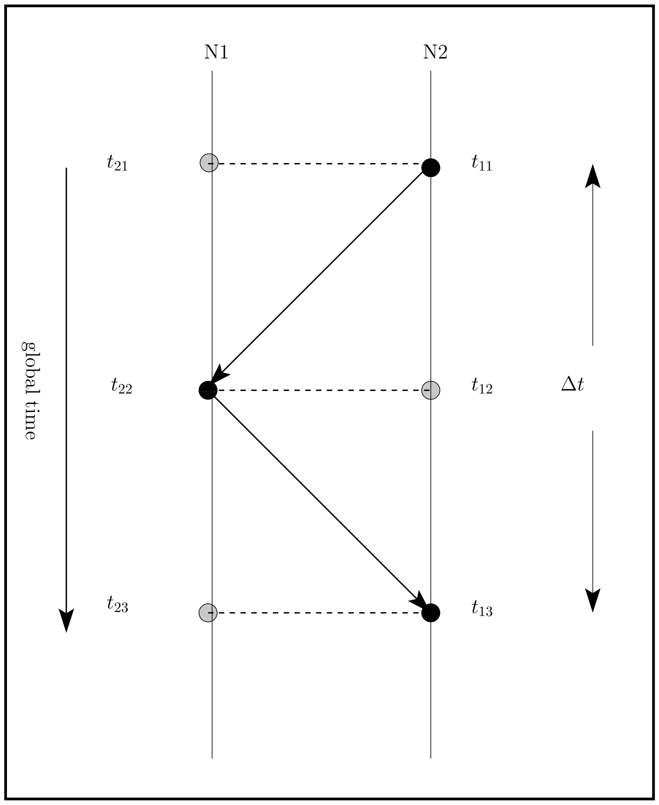


Figure H.1: Event diagram for network delays

To determine the estimate of t_{23}, time on node $N1$ corresponding to t_{13} we use equation H.1

$$t_{23} \approx t_{22} + \frac{1}{2}\Delta t \tag{H.1}$$

where clearly

$$\Delta t \approx t_{13} - t_{11} \tag{H.2}$$

[5]As determined by java *System.currentTimeMillis()* system call.

is the *network latency* incurred by messages between nodes.

Probability Density Function Parameter Estimation								
Distribution	Prob.Plot Estimate	MLE	MLE Err	MLE CL	MLE CU	Est.cov. of parameter est.		
	μ	3.01696	52.637	25739	25945		μ	σ
Normal	σ	14.6407	37.2252	3753	3899	μ	2770.65	1.13476E-12
	Log Likelihood	-4231.87				σ	1.13476E-12	1385.72
	μ	10.1491	0.00200282	10.1452	10.1530		μ	σ
Poisson	σ	0.145532	0.00141641	0.1428	0.1484	μ	4.01127e-06	3.0558e-19
	Log Likelihood	-50902.2				σ	3.0558e-19	2.00621e-06
	a	47.0423	0.912333	45.2877	48.8649		μ	σ
Lognormal	b	549.33	10.7105	528.7335	570.7281	a	0.832352	-9.71967
	Log Likelihood	-50931.1				b	-9.71967	114.715
	μ	25652.5	51.361	25552	255753		μ	σ
Log Logistic	σ	2142.23	24.5086	2095	2191	μ	2637.95	39.1888
	Log Likelihood	-51021.3				σ	39.1888	600.673

Table H.1: Showing parameter estimate and Maximum Likelihood estimates

APPENDIX I

Distributed Termination detection

I.1 Tracing Algorithms

Dijkstra-Scholten algorithm A formalisation of this algorithm in given in ([226]), and proceeds a follows; Consider the state Z_p of a process partitioned into two subsets, *passive* and *active*. Also consider a set $\mathbb{P}$ of all processes an set E of all message channels between any given pair of processes.
Then consider a predicate $term$ given in theorem 1

Theorem 1. *Theorem*

$$term \Longleftrightarrow (\forall p \in \mathbb{P} : state_p = passive)$$
$$\bigwedge(\forall p \in E : M_{pq} \textit{ does not contain } \langle mes \rangle \textit{ a message.)}$$

Proof. If all processes are passive, no internal or send event is applicable. If, more over no channel contains a $\langle mes \rangle$ message, no receive event is applicable, hence no basic event is applicable at all. If some process is active, a send or internal event is possible in that process and if some channel contains a $\langle mes \rangle$ message the receipt of a message is applicable. □

During the distributed computation, consider that there is a special node,an initiator ,p_o and that the detection algorithm maintains a computation tree $T = (V_T, E_T)$ with the following properties:

1. Either T is empty, or T is a directed tree with the root p_0 as the initiator

2. The set V_T includes all processes and all basic messages (messages sent by the underlying computation) in transit.

That is, the vertices are nodes of the network and messages in transit. Steps of the computation trigger updates.

The initiator, P_0, calls *Announce* when $P_0 \notin \mathrm{V}_T$; By the first property, T is empty in this case, and by the second property, $term$ holds.
To preserve the properties of the computation tree when the basic computation evolves, T must be expanded when a basic message is sent or when a process, not in the tree, becomes active.

When a process p sends a basic message $\langle mes \rangle$, $\langle mes \rangle$ is inserted into a tree and the father of $\langle mes \rangle$ is p.
When a process p, not in the tree, becomes active by the receipt of a message from some process q, q becomes the father of p.
To represent the sender of a message explicitly, a basic message $\langle mes \rangle$ sent by q will be denoted as $\langle mes, q \rangle$ [226].

The removal of nodes from T is also necessary, for two reasons. First a basic message is deleted when it is received. Second, to ensure progress of the detection algorithm the tree must collapse within a finite number of steps after termination. Messages are the leaves of T; processes maintain a variable that counts the number of their sons in T. The deletion of a son of process p occurs in a different process q; it is either the receipt of a son message, or the deletion of a son process q. To prevent corruption of

p's son count, a signal message $\langle sig, p\rangle$ can be sent to p when a son of p is deleted. This message replaces the deleted son of p, and its deletion, i.e., its receipt, occurs in process p and p decrements its son count when its receives a signal.

The Dijkstra-Scholten algorithm achieves an attractive balance between the control communication and the basic communication; for each basic message sent from p to q the algorithm sends exactly one control message from p to q [226].

The Dijkstra-Scholten algorithm was generalised to decentralized basic computations by [206] to give a Shavit-Franchez algorithm. In that algorithm, the computation graph is a *forest*[1] of which each tree is rooted at an initiator of the basic computation. The tree rooted at p is denoted T_p. The algorithm maintains a graph F = (V$_F$, E$_F$) such that

1. either F is empty or F is a forest of which each tree is rooted in an initiator; and

2. V$_F$ includes all active processes and all basic messages.

As in the Dijkstra-Scholten algorithm, termination is detected when the graph becomes empty. Unfortunately, in the case of a forest it is not trivial to see whether the graph is empty. Details of this algorithm can be seen in [206, 226]

I.2 Some Selected Algorithms, some optimisations and robustness considerations

To expand on the various schemes for detecting termination, below is a discussion of a selection of algorithms.

1) An (N -1)-Resilient Algorithm for Distributed Termination Detection

[144] presents a fault-tolerant termination detection algorithm based on a previous

[1]A forest is a disjoint union of trees.

fault-sensitive scheme by [78]. The proposed algorithm can tolerate any number of crash failures. It runs as efficiently as its non fault-tolerant predecessor if no process actually fails during the computation, and otherwise incurs only a small amount of cost for each actual failure. It is assumed that the underlying communication network provides such services are reliable end-to-end communication, failure detection, and fail flush.

2) Detecting Termination of Distributed Computations By External Agents [118] presents two algorithms for detecting termination of distributed computations monitored by an external controlling agent. The first algorithm is based on the *weighted throw counting* scheme [118]. Weights are assigned to each active process and to each message in transit. The agent has a weight, too. The algorithm maintains an invariant that the sum of all the weights equals one. The agent concludes the termination when its weight equals one. A space-efficient encoding of the weights is also proposed.

The second algorithm adopts the distributed snapshots scheme. When a process becomes idle, it takes a local snapshot and sends the snapshot to the agent. The agent puts the local snapshots together to form a global snapshot and determines the termination by checking the recorded state in the global snapshot.

[117] observes that by comparison, the first one is better if storage space is the major consideration, while the second is more suitable for real-time systems, because no waiting is employed on the processes due to termination detection. The second algorithm is also optimal in minimizing the message complexity: only one additional message carrying the local snapshot is needed per idleness [117]

3) A Distributed Termination Detection Scheme [250] proposes a fully distributed scheme for detecting the termination of distributed computations. The scheme does not require a pre-defined detector, and takes into account problems such as network delay and the non-order-preserving arrival of messages. It is claimed that the scheme can be applied to any kind of connection topology. The correctness of the scheme is presented in terms of showing that the global stable condition holds when the scheme declares the termination of the computation. The upper bound of the number of the messages which are used to detect termination is also discussed [250].

I.3 Static and dynamic termination algorithms

[37] then discusses details of the static and dynamic termination algorithms following the above termination definition (*static and dynamic*).

In the static termination case a control process C_i called a controller is associated with each application process P_i. The role of C_i is to observe the behaviour of P_i and to cooperate with other controllers C_j to detect occurrence of the predicate *Sterm*. In order to detect static termination, a controller, e.g. C_a, initiates detection by sending a control message query to all controllers,including itself. A controller say C_i responds with a message $(\textit{ld}_i)$ where *ld*$_i$ is a *boolean* value. C_a then combines all the boolean values received in reply messages to compute $td := \bigwedge_{1 \leq i \leq n} (\textit{ld}_i)$.

If td is true, C_a concludes that termination has occurred, otherwise it sends new query messages. The basic sequence of sending of query messages followed by the reception of associated reply messages is called a *wave*.

In the static termination algorithm, to ensure safety when the controller C_i computes the value ld_i sent back in a reply message, the values $ld_1, ..., ld_n$ must be such that

$$\bigwedge_{1\leq i\leq n} \quad ld_i \Rightarrow Sterm$$
$$\Rightarrow \forall P_i \in P : passive_i \wedge (NE_i = \varnothing) \wedge \neg fulfilled_i(ARR_i)$$

A controller C_i delays a response to a control *query* message as long as the following predicate that can be evaluated locally is false;

$$passive_i \wedge (noack_i = \varnothing) \wedge \neg fulfilled_i(ARR_i)$$

When this predicate is false, the static termination cannot be guaranteed. Regarding correctness, the values reported by the wave must no miss the activity of processes in the wake of the wave. [4] proposes that this could be accomplished in the following manner; Each controller C_i maintains a boolean variable, cp_i, initialised to true $iff\ P_i$ is initially passive in the following way

- When P_i becomes active, cp_i is set to false.
- When C_i sends a reply message to C_α it sends the current value of cp_i with this message, and then sets cp_i to $true$

Thus if a reply message carries value true from C_i to C_α, it means that P_i has been continuously passive since the previous wave and the messages arrived and not yet consumed are not suffice to activate P_i and all output channels of P_i are empty. Figure graphically illustrates this algorithm. Furthermore, presented below is a sequence of statements executed by controllers. The statements are labelled S1 to S6, with S5 only executed by C_α. In these statements *message* refers to any message of the underlying computation, whilst *queries* and *replies* are control messages.

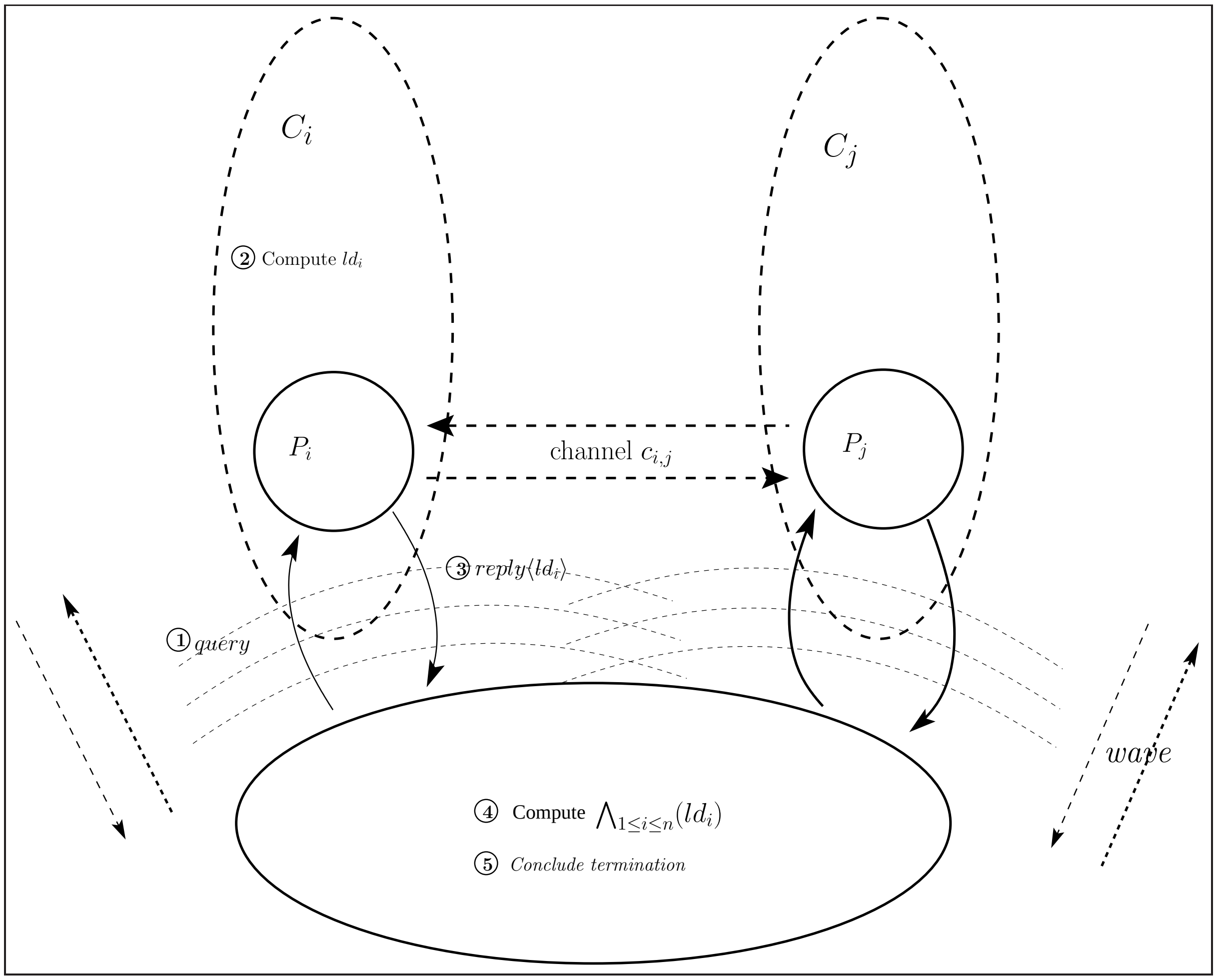


Figure I.1: An algorithm for static termination

S1: when P_i sends a message to P_j

$notack_i := notack_i + 1$

S2: when a message from P_j arrives P_i

send ack to C_j

S3: when C_i receives ack from C_j

$notack_i := notack_i - 1$

```
S4: when $P_i$ becomes active
$cp_i := false$

S5: When $C_i$ receives query from $C_\alpha$
(S5 is only executed by $C_\alpha$) wait until
$((passive_i \wedge (notack_i = \varnothing)) \wedge \neg fulfilled_i(ARR_i)$;
$ld_i := cp_i$;
$cp_i := true$;
send $reply(ld_i)$ to $C_\alpha$

S6: When controller $C_\alpha$ decides to detect static termination

repeat send query to all $C_i$;
       receive $reply(ld_i)$ from all $C_i$;
       $td := \bigwedge_{1 \le i \le n}(\textbf{\textit{ld}}_i)$
until td
claim static termination
```

Regarding dynamic termination, recall that dynamic termination can occur *before* all messages computation has arrived, because of this, termination of a computation can be detected sooner than in static termination. For the dynamic termination algorithm, consider C_α to denote the controller that launches the waves.

In addition to cp_i each controller C_i,has two vector variables, say denoted s_i r_i, that count messages respectively sent to and received from every other process, i.e. represent

- $s_i[j]$ denotes the number of messages sent by P_i to P_j;

- $r_i[j]$ denotes the number of messages received by P_i to P_j;

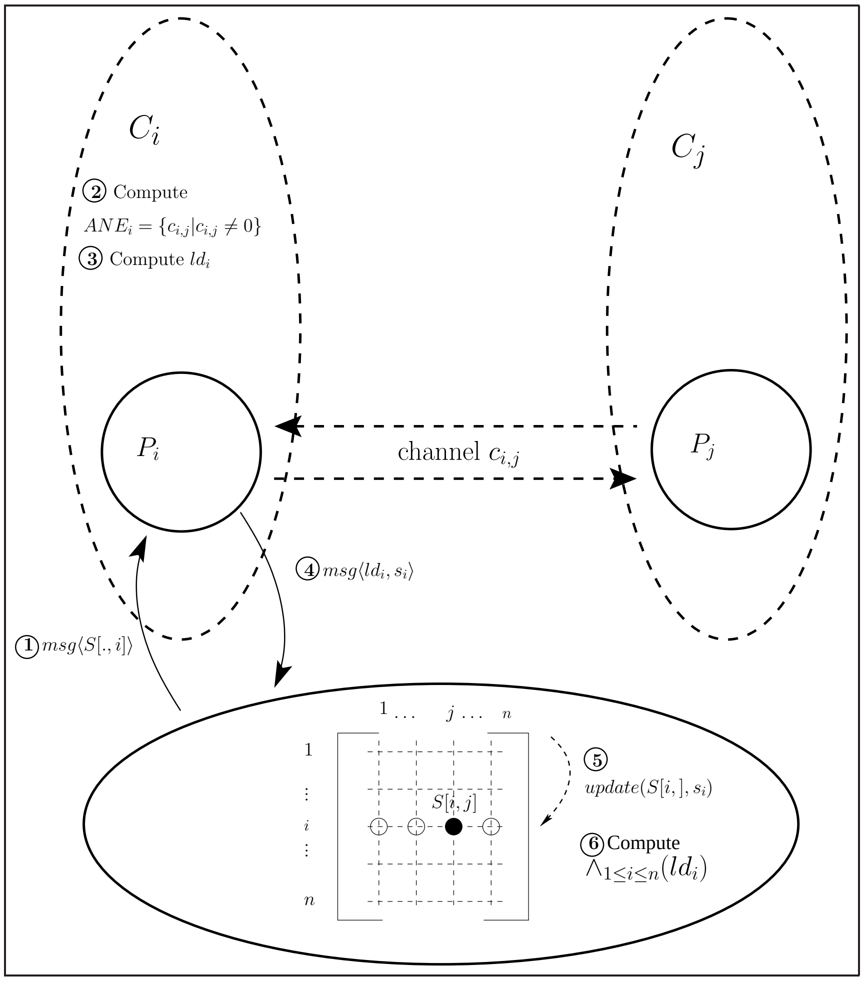

C_i
② Compute
$ANE_i = \{c_{i,j} | c_{i,j} \neq 0\}$
③ Compute ld_i
C_j
P_i
channel $c_{i,j}$
P_j
④ $msg\langle ld_i, s_i \rangle$
① $msg\langle S[.,i] \rangle$
$1 \ldots \quad j \ldots \quad n$
$S[i,j]$
⑤ $update(S[i,], s_i)$
⑥ Compute $\bigwedge_{1 \leq i \leq n}(ld_i)$

Figure I.2: An algorithm for dynamic termination

First, C_α sends to each C_i a query message containing the vector $(S[1,i],..,S[n,i])$, denoted by $S[.,i]$. Upon receiving this query message, C_i computes the set ANE_i of its *non-empty* channels. This is an approximate knowledge but is sufficient to ensure correctness.

Then C_i computes ld_i, which is true *if and only if* P_i has been continuously passive since the previous wave and its requirement cannot be fulfilled by all the messages arrived and not yet consumed (ARR_i) and all messages potentially in its input channels

(ANE_i). C_i sends to C_α a reply message carrying the values ld_i and vector s_i. Vector s_i is used by C_α to update row $S[i,]$ and thus gain more accurate knowledge.

Vector variables s_i and r_i allow C_α to update its (approximate) global knowledge about messages sent by each P_i to each P_j and get an approximate knowledge of the set of non-empty input channels. I have used figure do depict the main aspect of this algorithm and steps.

Also consider the formalisation of the algorithm below, where all the controllers C_i execute statements $S1$ to $S4$ as defined below and where only C_α executes $S5$.

```
S1: when $P_i$ sends a message to $P_j$
$s_i[j] := s_i[j] + 1$

S2: when a message from $P_j$ arrives at $P_i$
$r_i[j] := r_i[j] + 1$
S3: when $P_i$ becomes active
$cp_i := false$

S3: when $C_i$ receives query($V[1..n]$) from $C_\alpha$ (where
$V[1..n] = S[1..n, i]$ is the ith column of S)
$ANE_i := \{P_j : V[j] > r_i[j]\}$ ;
$ld_i := cp_i \wedge \neg fulfilled_i(ARR_i \cup NE_i)$
$cp_i := (state_i = passive)$; send $reply(ld_i, s_i) to C_\alpha$
```

```
S5: when controller $C_\alpha$ decides to detect dynamic termination
repeat for each $C_i$
send $query(S[1..n, i])$ to $C_i$ (i.e. the $ith$ column of S sent to $C_i$)
receive $reply(ld_i, s_i)$ from all $C_i$;
$\forall i \in [1..n] : S[i, .] := s_i$;
$td := \bigwedge_{1 \leq i \leq n}(\boldsymbol{ld}_i)$

until td;
claim dynamic termination
```

I.4 Contemporary taxonomy for algorithms

Recent research activity Research in the field was very active in the 1980's 1990's with vast number of algorithms proposed. Research output has since slowed down with one or two algorithms proposed per year in recent times. In the recent research work;

1. [169] (Mittal & Vankatesan, 2008), presents a transformation that can be used to convert any fault-sensitive termination detection algorithm (for a fully connected network topology) into fault-tolerant termination detection algorithm capable of coping with process crashes. The transformation assumes a perfect failure detector. It is also shown there that under the assumptions made the scheme is optimal in terms of message complexity

2. [17] (Bapat & Arora, 2008) proposes a message efficient termination detection in a wireless sensor network ,WSN. The topology assumed there is that of a multi-hop network of WSN nodes each with a unique identifier. The wireless communication links between the nodes is bidirectional and the reliability either

way is not necessarily the same. The algorithm assumes a special role for a base station node and assume unique identifiers for nodes hence it is asymmetric. The algorithm is no message optimal but claimed to be message efficient as it detects termination from reports of only a subset of nodes in the network. The discussion of the proposed algorithm does not cover fault tolerance, but evidently the algorithm suffers the same fault tolerance issues applicable to schemes with central entities.

3. [71], (DeMara et al) presents a tiered algorithm claimed to be time-efficient and message-efficient for process termination. The algorithm uses a global invariant of equality between process production and consumption at each level of process nesting to detect termination regardless of execution interleaving order or network transit time. Then correctness of the algorithm is validated for arbitrary process launching hierarchies. Regarding performance, the algorithm is compared to existing schemes including credit termination algorithms.

4. [70], (De et al, 2007) proposes an application layer based modified weight-throwing protocol for the distributed termination detection problem. The protocol is proposed for a purely mobile distributed environment with no static hosts. The mobile hosts are considered with limited functionality. The discussion of the effect of mobility on the proposed algorithm and is given.

5. [238] (Wang & Mayo, 2004) proposes a symmetric algorithm, assuming asynchronous communication. The algorithm assumes a more general network topology of a combination of a logical ring for the initial processes and a number of computation trees Efficiency gains are made by circulating controlling messages at most once around the ring. The algorithm assumes there are not faulty processes, but that processes can be created and accepts external processes during the computation.

In the tables that follow, I update Matocha's taxonomy with these and other algorithms that have since been published to incorporate current trends. For example a large proportion of recent algorithms have been in mobile and wireless networks area and are flexible when it comes to topology assumptions. Most algorithms are asynchronous and have no restrictions when considering message arrival.

The taxonomy and its element are depicted in Figure 2.1.

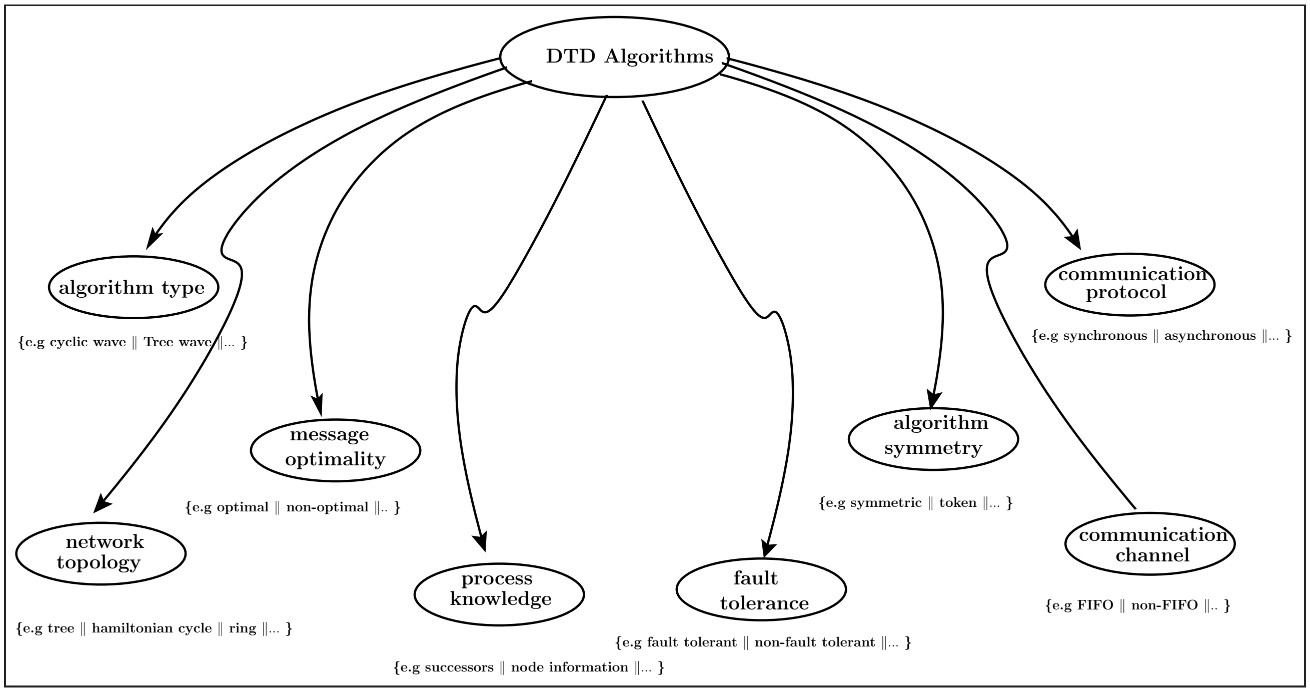

DTD Algorithms
algorithm type
{e.g cyclic wave ‖ Tree wave ‖... }
communication protocol
{e.g synchronous ‖ asynchronous ‖... }
message optimality
{e.g optimal ‖ non-optimal ‖.. }
algorithm symmetry
{e.g symmetric ‖ token ‖... }
network topology
{e.g tree ‖ hamiltonian cycle ‖ ring ‖... }
process knowledge
{e.g successors ‖ node information ‖... }
fault tolerance
{e.g fault tolerant ‖ non-fault tolerant ‖... }
communication channel
{e.g FIFO ‖ non-FIFO ‖.. }

Figure I.3: A taxonomy for distributed termination detection suggested

Tables I.1 through to I.7 show the classification for each of categories of the taxonomy.

Algorithm	Cyclic wave	Tree wave	General wave	Non-repetitive wave	Parental responsibility	Credit recovery	Other	
(Francez,1980)		✓						
(Dijkstra & Scholten,1980)								
(Francez et. al, 1981)	✓							
(Misra & Chandy, 1982)					✓			
(Chandy &Misra, 1985)			✓					
(Szymaski et. al, 1985)		✓						
(Mattern 1987)	✓							
(Muller, 1987)	✓							
(Huang, 1988)				✓				
(Mattern, 1989)							✓	
(Vankatesan, 1989)							✓	
(Lai et al.,1992)								✓
(Wang and Mayo, 2004)						✓		
(De et al, 2007)								✓(not clear!)
(Mittal & Vankatesan, 2008)								✓(not clear!)

continued on next page ...

TableI.1 ... continued from previous page

Algorithm	Cyclic wave	Tree wave	General wave	Non-repetitive wave	Parental responsibility	Credit recovery	Other
(Bapat & Arora, 2008)	✓(not clear!)						

Table I.1: DTD algorithms and their associated type, adapted from [159]

Algorithm	Hamiltonian cycle	Computation tree	Spanning tree	No requirement	Other
(Francez,1980)		✓			
(Dijkstra & Scholten,1980)		✓			
(Francez et. al, 1981)	✓				
(Misra & Chandy, 1982)		✓			
(Dijkstra et. al.„1983)	✓				
(Kumar, 1985)	✓				
(Chandy &Misra, 1985)					✓
(Szymaski et. al, 1985)			✓		
(Mattern 1987)	✓	✓			
(Muller, 1987)	✓				
(Huang, 1988)				✓	
(Mattern, 1989)					
(Vankatesan, 1989)				✓	
(Lai et al.,1992)		✓			
(Wang and Mayo, 2004)		✓			(logical ring)
(De et al, 2007)					(mobile hosts)

continued on next page ...

TableI.2 ... continued from previous page

Algorithm	Hamiltonian cycle	Computation tree	Spanning tree	No requirement	Other
(Mittal & Vankatesan, 2008)					(fully connected network)
(Bapat & Arora, 2008)					(multihop network)

Table I.2: DTD algorithms and their necessary topology , adapted from [159]

Algorithm	Specialized	p_0 only	p_0 at run time	Token	Symmetric
(Francez,1980)	✓				
(Dijkstra & Scholten,1980)		✓			
(Francez et. al, 1981)		✓			
(Misra & Chandy, 1982)		✓			
(Dijkstra et. al.,,1983)		✓			
(Kumar, 1985)				✓	
(Chandy &Misra, 1985)				✓	
(Szymaski et. al, 1985)					✓
(Mattern 1987)					✓
(Muller, 1987)					✓
(Huang, 1988)		(central entity)			
(Mattern, 1989)		✓			
(Vankatesan, 1989)			✓		
(Lai et al.,1992)			✓		
(Wang and Mayo,2004)					✓
(Mittal & Vankatesan, 2008)	(failure detector)				

continued on next page ...

TableI.3 ... continued from previous page

Algorithm	Specialized	p_0 only	p_0 at run time	Token	Symmetric
(Bapat & Arora, 2008)	✓(unique ids)				

Table I.3: DTD algorithms and their process symmetry, adapted from [159]

Algorithm	Successors	Node information	Upper bound on net diameter	Other	None
(Francez,1980)		✓			
(Dijkstra & Scholten,1980)		✓			
(Francez et. al, 1981)	✓				
(Misra & Chandy, 1982)		✓			
(Dijkstra et. al.,,1983)	✓				
(Rana,1983)	✓			logical clocks	
(Arora and Sharma,1983)	✓			distance function	
(Kumar, 1985)	✓				
(Chandy &Misra, 1985)	✓				
(Szymaski et. al, 1985)			✓		
(Shavit and Francez, 1986)					✓
(Mattern 1987)	✓				
(Muller, 1987)	✓				
(Huang, 1988)				(list of $p_i s$)	
(Huang, 1989)				(central entity)	
(Mattern, 1989)					✓

continued on next page ...

TableI.4 ... continued from previous page

Algorithm	Successors	Node information	Upper bound on net diameter	Other	None
(Vankatesan, 1989)		✓			
(Lai et al.,1992)		✓			
(Mayo and Kearns,1994)	✓			logical clocks	
(Wang and Mayo,2004)	✓(in ring)			process can't leave or be destroyed before termination	
(Mittal & Vankatesan, 2008)	✓			✓	
(Bapat & Arora, 2008)				✓(base station)	

Table I.4: DTD algorithms and their process knowledge, adapted from [159]

Algorithm	Synchronous Communication	Asynchronous Communication
(Francez,1980)	✓(CSP)	
(Dijkstra & Scholten,1980)		✓
(Francez et. al, 1981)	✓(CSP)	
(Misra & Chandy, 1982)	✓(CSP)	
(Dijkstra et. al.„1983)	✓	
(Rana,1983)	✓(CSP)	"can be modified for"
(Arora and Sharma,1983)	✓	
(Misra, 1983)		✓
(Kumar, 1985)		✓
(Chandy &Misra, 1985)		✓(though in CSP)
(Szymaski et. al, 1985)	✓	
(Shavit and Francez, 1986)		✓
(Mattern 1987)		✓
(Muller, 1987)		✓
(Huang, 1988)		✓
(Mattern, 1989)		✓
(Vankatesan, 1989)		✓
(Lai et al.,1992)		✓
(Mayo and Kearns,1994)	✓	
(Wang and Mayo, 2004)		✓
(De et al, 2007)		✓
(Mittal & Vankatesan, 2008)		✓
(Bapat & Arora, 2008)		✓

Table I.5: DTD algorithms and their communication protocol , adapted and extended [159]

Algorithm	FIFO	No restriction
(Francez,1980)	✓	
(Dijkstra & Scholten,1980)	✓	
(Francez et. al, 1981)	✓	
(Misra & Chandy, 1982)	✓	
(Dijkstra et. al.,,1983)	✓	
(Rana,1983)	✓	
(Arora and Sharma,1983)	✓	
(Misra, 1983)	✓	
(Kumar, 1985)		✓
(Chandy &Misra, 1985)	✓	
(Szymaski et. al, 1985)	✓	
(Shavit and Francez, 1986)		✓
(Mattern 1987)		✓
(Muller, 1987)		✓
(Huang, 1988)		✓
(Mattern, 1989)		✓
(Vankatesan, 1989)	✓	
(Lai et al.,1992)		✓
(Mayo and Kearns,1994)	✓	
(Wang and Mayo, 2004)	✓	
(De et al, 2007)	✓	
(Mittal & Vankatesan, 2008)		✓
(Bapat & Arora, 2008)		✓(not stated explicitly)

Table I.6: DTD algorithms and their restrictions on message arrival , adapted and extended [159]

Algorithm	Fault tolerant	Not fault tolerant
(Francez,1980)		✓
(Dijkstra & Scholten,1980)		✓
(Francez et. al, 1981)		✓
(Misra & Chandy, 1982)		✓
(Dijkstra et. al.,,1983)		✓
(Rana,1983)		✓
(Arora and Sharma,1983)		✓
(Misra, 1983)	✓	
(Kumar, 1985)		✓
(Chandy &Misra, 1985)		✓
(Szymaski et. al, 1985)		✓
(Shavit and Francez, 1986)		✓
(Mattern 1987)		✓
(Muller, 1987)		✓
(Huang, 1988)		✓
(Mattern, 1989)		✓
(Vankatesan, 1989)	✓	
(Lai et al.,1992)		✓
(Mayo and Kearns,1994)		✓
(Wang and Mayo, 2004)		✓
(De et al, 2007)		✓
(Mittal & Vankatesan, 2008)	✓(assumes perfect failure detector)	
(Bapat & Arora, 2008)		✓

Table I.7: DTD algorithms and their fault tolerance, adapted and extended [159]

Algorithm	Optimal	Not optimal
(Francez,1980)		✓
(Dijkstra & Scholten,1980)	✓	
(Francez et. al, 1981)		✓
(Misra & Chandy, 1982)		✓
(Dijkstra et. al.,,1983)		✓
(Rana,1983)		✓
(Arora and Sharma,1983)		✓
(Misra, 1983)		✓
(Kumar, 1985)		✓
(Chandy &Misra, 1985)		✓
(Szymaski et. al, 1985)		✓
(Shavit and Francez, 1986)		✓
(Mattern 1987)	✓(If star or complete graph)	✓(Otherwise)
(Muller, 1987)		✓
(Huang, 1988)		✓
(Mattern, 1989)	✓	
(Vankatesan, 1989)	✓If constant number of failures)	✓(Otherwise)
(Lai et al.,1992)	✓	
(Mayo and Kearns,1994)		✓
(Wang and Mayo, 2004)		✓(“close to“)
(De et al, 2007)		✓
(Mittal & Vankatesan, 2008)	✓	
(Bapat & Arora, 2008)		✓(but efficient)

Table I.8: DTD algorithms and their message optimality, adapted and extended [159]

APPENDIX J

Graphs, representation and complexity of algorithms

In general a common way of representing graphs as data structures is to consider an adjacency matrix [47], and its representational data structures. An analysis of the complexity issues is given in [96] and summarised here.

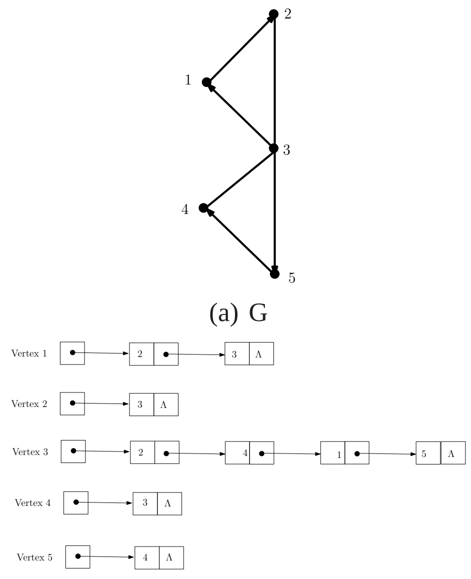
i.e. Let $G = (V, E)$ be a graph whose vertices have been (arbitrarily) ordered $v_1, v_2, \ldots v_n$. The adjacency matrix **(**M) = $(m_{i,j})$ of G is an $n \times n$ matrix with entries

$$m_{i,j} = \begin{cases} 0 & if \quad v_i v_j \notin E \\ 1 & if \quad v_i v_j \in E \end{cases}$$

for example consider Figure J.1, the adjacency matrix **M** is given by

$$m_{i,j} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and can be represented as a an adjacency list of G given in Figure J.1 (b). [96] reasons that, by definition, the main diagonal of M is all zeros, and M is symmetric about the main diagonal if and only if G is an undirected graph. If M is stored as a 2-dimensional array, then only one step (more precisely $\mathcal{O}\left(1\right)$ time) is required for the statements "Is $v_i v_j \in E$ or "Erase the edge $v_i v_j$ An instruction such as "mark each vertex which is



(a) G

(b) adjacency lists of G

Figure J.1: G

adjacent to v_j requires scanning the entire column j and hence takes n steps. Similarly, "mark each edge" takes n^2 steps. The space requirement for the array representation is $\mathcal{O}\left(n^2\right)$.

Some of the performance figures above can be improved upon when the density of M is low. We use the term sparse to indicate that $\|E\| \ll n^2$, i.e., the number of edges is much less than n^2. One of the most talked about classes of sparse graphs are the planar graphs [1] for which Euler proved that $\|E\| < 3n - 6$.

Regarding adjacency lists, for each vertex v_i of G an adjacency list $adj\left(v_i\right)$ can be created, containing those vertices adjacent to v_j. The adjacency lists are not necessarily sorted although one might wish them to be (see Figure J.1). The space requirement for the adjacency list representation of a graph with n vertices and e edges is

$$\mathcal{O}\left(\sum[1 + d_i]\right) = \mathcal{O}\left(n + e\right)$$

[1] A graph that can be embedded on a plane.

where d_i denotes the degree of i Thus, from storage considerations, it is usually more advantageous to use adjacency lists than the adjacency matrix to store a sparse graph. Often, it is also advantageous from time considerations to store a sparse graph using adjacency lists. For example, the instruction "mark each vertex which is adjacent to v_j requires scanning the list $Adj\,(v_i)$ and hence takes dj steps. Similarly, "mark each edge" takes $\mathcal{O}\,(e)$ steps using adjacency lists, a substantial saving over the adjacency matrix for a sparse graph. However, erasing an edge is more complex with lists than with the matrix as shown in Table 2.2. Thus there is no representation of a graph that is best for all operations and processes. Since the selection of a particular data structure can noticeably affect the speed and efficiency of an algorithm, decisions about the representation must incorporate a knowledge of the algorithms to be applied. Conversely, the choice of an algorithm may depend on how the data is initially given. For example, an algorithm to set up the adjacency lists of a sparse graph will take longer if we are initially given its adjacency matrix as an $n \times n$ array rather than as a collection of ordered pairs representing the edges. A graph problem is said to be linear in the size of the graph, or simply linear, if it has an algorithm which can be implemented to run in $\mathcal{O}\,(n+e)$0 steps on a graph with n vertices and e edges. This is usually the best that one could expect for a graph problem. By a careful choice of algorithm and data structure a number of simple problems can be solved in linear time; these include testing for connectivity, biconnectivity , and planarity [96]