UNIVERSITY OF SOUTHAMPTON

Adaptive Distributed Resource Allocation Using Cooperative Information-Sharing

by

Partha Sarathi Dutta

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering and Applied Science

School of Electronics and Computer Science

October 2005

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Partha Sarathi Dutta

In distributed systems, the individual processing units typically solve parts of the overall problem while having only localised views of the system. In such settings, the individual views could be in conflict with one another because there is no centrally controlled synchroniser (by choice of design). This problem can, in turn, create conflicting partial solutions, thereby generating sub-optimal solutions to the overall problem. Such conflicts are aggravated by dynamically changing states (which are common in such systems). Thus, coordinating the distributed components is of critical importance to allow successful task processing.

Now, distributed resource allocation (RA), which is a central problem in several important applications, exhibits all of the challenges mentioned above. Specifically, in this domain, individual processors allocate resources to partially complete tasks which are successfully solved by allocating all the necessary resources. In a shared task environment, they must act by considering the resource availabilities across the system to ensure conflict-free and successful allocations. Moreover, resource availabilities can change requiring allocations to adapt to solve tasks efficiently. Against this background, in this thesis, we focus on developing an effective solution for tasks where RA is *sequential* since they are commonly found in real-world RA systems.

Multi-agent systems (MAS) is a promising solution paradigm for such distributed RA problems. This is because, in a MAS, the agents are designed as autonomous, goal-directed, reactive programs that can adapt to environment changes to achieve the system objective. However, in line with the discussion above, such behaviour rests upon two key properties: the ability to *estimate states* and to *update these estimates* as states change so as to adapt their behaviour. We argue that such capabilities require a mechanism for sharing state information to allow the agents to acquire the necessary knowledge of the states so that they can generate non-conflicting partial solutions and successfully adapt to state changes.

Against this background, in this thesis, we develop a multi-agent information-sharing protocol to efficiently solve sequential RA problems. This protocol lets the agents cooperate with one another, by sharing information, to build reliable estimates of the system states which, in turn, generate effective adaptive behaviour. We use Q-learning, an algorithm for generating robust estimates without any prior knowledge of the environment dynamics, as the tool for estimate learning using the shared information. In so doing, our research significantly extends the state-of-the-art in cooperative MAS technology for sequential RA problems.

First, our information-sharing protocol — the post task-completion (PTC) protocol — is novel in its principle of application and is also the most effective information-sharing method for generating high-quality estimates in dynamic distributed sequential RA problems. We establish its merits using theoretical analyses. Second, we design communication heuristics based on PTC that can be used in a real sequential RA application: call routing in mesh networks. Using empirical analyses, we show that these heuristics outperform a host of benchmark algorithms used in this application domain. Third, we improve the adaptiveness of PTC in highly dynamic environments by extending the communication heuristic. Empirical analyses demonstrate that this extended protocol achieves superior responsiveness to state changes caused by network failures and successfully diagnoses failures. Finally, we extend the effectiveness of the extended PTC heuristic by using selective information-sharing. We show that sharing information using a notion of information redundancy significantly reduces the communication overhead of PTC but without sacrificing its performance advantages.

In achieving the above-mentioned contributions, we successfully establish the validity of our claim about cooperative information-sharing being able to generate effective state estimation and adaptive behaviour required for efficiently solving dynamic distributed sequential RA problems. In so doing, we also contribute towards extending the current technology for distributed network call routing by providing a highly efficient communication protocol that generates high quality solutions for the routing problem.

Contents

List of Figures ix

List of Tables xi

Acronyms xiii

Acknowledgements xv

1 Introduction 1
- 1.1 The Aims and The Research Challenges . . . 5
  - 1.1.1 State Estimation . . . 6
  - 1.1.2 Cooperative Information-Sharing . . . 6
  - 1.1.3 Adaptive Decision Making . . . 7
- 1.2 Research Contributions . . . 7
  - 1.2.1 A Novel Information-Sharing Protocol . . . 8
  - 1.2.2 Improving Performance of the Information-Sharing Protocol in Highly Dynamic Environments . . . 9
  - 1.2.3 Limiting Communication Cost of the Information-Sharing Protocol 11
- 1.3 Thesis Structure . . . 12

2 Related Work 15
- 2.1 Cooperative Multi-Agent Systems for Distributed Problem Solving . . . 16
  - 2.1.1 Functionally Accurate Cooperation . . . 16
  - 2.1.2 Organisational Structuring . . . 17
  - 2.1.3 Sophisticated Local Control . . . 18
  - 2.1.4 Teamwork Based on Joint Intentions . . . 20
  - 2.1.5 Coordination using Markov Decision Processes . . . 21
  - 2.1.6 Cooperative Social Networks . . . 22
- 2.2 Learning-Based Cooperative Multi-Agent Systems . . . 23
  - 2.2.1 Q-Routing in Dynamic Networks . . . 23
  - 2.2.2 Team Partitioned Opaque Transition Reinforcement Learning . . . 25
  - 2.2.3 Policy Gradient Search . . . 26
  - 2.2.4 Communication Decisions in Multi-Agent Coordination . . . 27
  - 2.2.5 Other Machine Learning Algorithms . . . 28
- 2.3 Summary . . . 29

3 Post Task-Completion Information-Sharing Protocol 31

3.1 Application Task Domain and Motivation for Designing PTC 32
3.1.1 Sequential Resource Allocation Tasks 32
3.1.2 Designing PTC for Sequential Resource Allocation Tasks 33
3.2 Information-Sharing Protocols for a Network Call Routing Application . . 34
3.2.1 Post Task-Completion Information-Sharing 37
3.2.2 Q-routing 38
3.2.3 TPOT-RL 38
3.3 Advantages of the PTC Protocol 39
3.3.1 Basic Assumptions and Notations 39
3.3.2 Timeliness of Information Distribution in NN 41
3.3.3 Timeliness of Information Distribution in PTC 43
3.3.4 Comparing Timeliness of Information Distribution in PTC and NN 44
3.3.5 Improved Estimation Accuracy using PTC 45
3.3.6 Non-periodic Task Environment 46
3.4 Summary 48

4 Empirical Evaluation of PTC in Telephone Network Routing 49
4.1 Domain Properties 50
4.2 Cooperative Bandwidth Allocation for Call Routing 51
4.3 PTC Information-Sharing Heuristics 55
4.3.1 PTC Inform Average Capacity (PTC-A) 56
4.3.2 PTC Inform Minimum Capacity (PTC-M) 56
4.4 Experimental Setup 56
4.4.1 Performance Measures 57
4.4.1.1 Number of Successful Calls 57
4.4.1.2 Successful Routes of Different Lengths 58
4.4.1.3 Reward Information Messages 58
4.4.2 Network Characteristics and Experimental Parameters 60
4.5 Results and Analysis 61
4.5.1 Performance — Call Success Rate 61
4.5.1.1 Constant Load. 61
4.5.1.2 Dynamically Changing Load. 65
4.5.2 Performance — Success Rate for Calls of Different Lengths 68
4.5.3 Performance — Information Message Rate 75
4.5.3.1 Constant Load. 75
4.5.3.2 Dynamically Changing Load. 76
4.6 Summary 79

5 Adaptiveness in Highly Dynamic Environments 81
5.1 The Network Failure Model 83
5.1.1 Failure Characterisation 83
5.1.2 Detection Characterisation 84
5.2 Extending PTC to Deal With Failures 85
5.3 Extended PTC Information-Sharing in Network Call Routing 86
5.4 Bandwidth Allocation Using e-PTC 87
5.5 Results and Analysis 89

5.5.1 Without Failures . . . 89
5.5.1.1 Call Success Rate. . . . 90
5.5.1.2 Message Size. . . . 90
5.5.2 With Failures . . . 92
5.5.2.1 Call Success Rate. . . . 93
5.5.2.2 Call Success Rate Recovery. . . . 96
5.5.2.3 Message Size. . . . 97
5.6 Failure Diagnosis Using e-PTC . . . 100
5.6.1 The Failure Diagnosis Process . . . 101
5.6.2 The Diagnosis Algorithm . . . 101
5.7 Results and Analysis . . . 104
5.8 Summary . . . 107

6 Selective Post Task-Completion Information-Sharing 109
6.1 Adaptive PTC (a-PTC) . . . 110
6.1.1 Motivation for Designing a-PTC . . . 110
6.1.2 Formal Description of a-PTC . . . 111
6.1.2.1 Transmitting State Value. . . . 112
6.1.2.2 Updating Q. . . . 113
6.1.3 Implications of Selective Information Distribution . . . 113
6.2 Experimental Evaluation . . . 114
6.2.1 Impact on Call Success Rate . . . 115
6.2.2 Impact on Message Size . . . 119
6.3 Summary . . . 122

7 Conclusions and Future Work 123
7.1 Conclusions . . . 123
7.2 Future Work . . . 125
7.2.1 Effect of Network Topology on Performance . . . 125
7.2.2 Cost of Communication . . . 126
7.2.3 Other Resource Allocation Domains . . . 126
7.2.4 Hierarchical Control . . . 127
7.2.5 Extended Representation of State Dynamics . . . 128
7.2.6 Analysis of Algorithm Stability . . . 128

A Evaluation of a Broadcast Protocol for Sharing Information 129
A-1 The Broadcast Protocol . . . 130
A-2 Effect of Message Queue Size . . . 130
A-3 Experimental Results . . . 131
A-4 Summary . . . 133

Bibliography 135

List of Figures

3.1 The call forwarding process 36
3.2 Time diagram for PTC sharing 43

4.1 Agent actions in response to various message types 53
4.2 A 36-node irregular grid topology 59
4.3 Random network topologies 60
4.4 Improvement of call success rate deviation from IZDS relative to QR and TPOT-RL — topology of figure 4.2 63
4.5 Improvement of call success rate deviation from IZDS relative to QR and TPOT-RL — topology of figure 4.3(a) 63
4.6 Improvement of call success rate deviation from IZDS relative to QR and TPOT-RL — topology of figure 4.3(b) 64
4.7 Time variation of call success rates of QR, PTC-M, and TPOT-RL with network load fluctuations — topology of figure 4.2 66
4.8 Summary statistics of call success rate differences between QR and PTC-M with network load fluctuations — topology of figure 4.2 67
4.9 Call success rates at various distances — topology of figure 4.2 71
4.10 Call success rates at various distances — topology of figure 4.3(a) 72
4.11 Call success rates at various distances — topology of figure 4.3(b) 73
4.12 Time variation of message rates of QR, PTC-M, and TPOT-RL with network load fluctuations — topology of figure 4.2 77
4.13 Summary statistics of message rate differences between QR, TPOT-RL, and PTC-M with network load fluctuations — topology of figure 4.2 . . . 78

5.1 Failure Detection Model Schematic 84
5.2 A 50-node network topology 89
5.3 A 100-node network topology 90
5.4 Percentage call success rate increase in e-PTC (no failures) on the 50-node topology of figure 5.2 91
5.5 Percentage call success rate increase in e-PTC (no failures) on the 100-node topology of figure 5.3 91
5.6 Percentage message size increase in e-PTC (no failures) on the 50-node topology of figure 5.2 92
5.7 Percentage message size increase in e-PTC (no failures) on the 100-node topology of figure 5.3 93
5.8 Call success rate when 3 nodes fail: (a) - absolute CSR, (b) - percentage CSR increase in e-PTC over PTC on the 50-node topology of figure 5.2 . 94

5.9 Call success rate when 3 nodes fail: (a) - absolute CSR, (b) - percentage CSR increase in e-PTC over PTC on the 100-node topology of figure 5.3 . 95
5.10 Call success rate with failure: e-PTC (test), PTC (test) and baseline (load 0.1) on the 50-node topology of figure 5.2 96
5.11 Call success rate with failure: e-PTC (test), PTC (test) and baseline (load 0.1) on the 100-node topology of figure 5.3 97
5.12 Call success rate deviation of e-PTC and PTC from baseline with node failure (load 0.1) on the 50-node topology of figure 5.2 98
5.13 Call success rate deviation of e-PTC and PTC from baseline with node failure (load 0.1) on the 100-node topology of figure 5.3 98
5.14 Percentage increase in e-PTC message size with 3 failures on the 50-node topology of figure 5.2 99
5.15 Percentage increase in e-PTC message size with 3 failures on the 100-node topology of figure 5.3 99
5.16 Failure diagnosis algorithm 102
5.17 Schematic of various relationships between T, $\hat{t}_s$, and $\hat{t}_{us}$ described in the diagnosis algorithm of figure 5.16 103

6.1 Schematic of call setup and sequence of node state transmissions 112
6.2 Time variation of average call success rates of e-PTC and a-PTC using the 50-node topology of figure 4.3(a), $p_c = 0.2$, $\Delta s = 0.1$ 116
6.3 Time variation of average call success rates of e-PTC and a-PTC using the 100-node topology of figure 4.3(b), $p_c = 0.2$, $\Delta s = 0.1$ 117
6.4 Comparing the call success rate profiles of a-PTC and e-PTC on the 50-node topology of figure 4.3(a) 117
6.5 Comparing the call success rate profiles of a-PTC and e-PTC on the 100-node topology of figure 4.3(b) 118
6.6 Time variation of average message size of e-PTC and a-PTC using the 50-node topology of figure 4.3(a), $p_c = 0.2$, $\Delta s = 0.1$ 119
6.7 Time variation of average message size of e-PTC and a-PTC using the 100-node topology of figure 4.3(b), $p_c = 0.2$, $\Delta s = 0.1$ 120
6.8 Comparing the message size profiles of a-PTC and e-PTC on the 50-node topology of figure 4.3(a) 120
6.9 Comparing the message size profiles of a-PTC and e-PTC on the 100-node topology of figure 4.3(b) 121

A-1 Average steady-state call success rates of broadcast and PTC-M using various message queue lengths, topology of figure 4.2, load 0.1 132

List of Tables

3.1 Time diagram for nearest-neighbour sharing 42

4.1 Call success rates for all strategies — topology of figure 4.2 62
4.2 Call success rates for all strategies — topology of figure 4.3(a) 62
4.3 Call success rates for all strategies — topology of figure 4.3(b) 62
4.4 Call success rates at various distances — topology of figure 4.2 68
4.5 Call success rates at various distances — topology of figure 4.3(a) 69
4.6 Call success rates at various distances — topology of figure 4.3(b) 69
4.7 Information message rates for all strategies — topology of figure 4.2 . . . 76
4.8 Information message rates for all strategies — topology of figure 4.3(a) . . 76
4.9 Information message rates for all strategies — topology of figure 4.3(b) . . 77

5.1 Summary statistics of failure diagnosis on the 50-node topology of figure 5.2106
5.2 Summary statistics of failure diagnosis on the 100-node topology of figure 5.3106

Acronyms

a-PTC	Adaptive Post Task-Completion
CSR	Call Success Rate
e-PTC	Extended Post Task-Completion
IZDS	Instantaneous Zero Delay Search
MAS	Multi-Agent Systems
MS	Message Size
NN	Nearest Neighbour
PTC	Post Task-Completion
PTC-A	**P**ost **T**ask-**C**ompletion: **A**verage capacity
PTC-M	**P**ost **T**ask-**C**ompletion: **M**inimum capacity
QR	Q-Routing
RA	Resource Allocation
RL	Reinforcement Learning
TN	Telephone Network
TPOT-RL	Team Partitioned Opaque Transition Reinforcement Learning

Acknowledgements

I owe thanks to many for helping me completing this thesis. The two people who contributed the most towards making my PhD experience a success in all aspects by virtue of their ability to encourage, criticise, test, and guide, are my supervisors, Prof. Nick Jennings and Prof. Luc Moreau. It is to them that I completely owe the bright moments of success that helped overcome any misjudgements that I made. Without them, this thesis would not have happened.

Several other people inspired me through this often arduous course. Firstly, I am thankful to the Engineering and Physical Sciences Research Council (EPSRC), who provided financial support to the project MOHICAN (to which my work was affiliated) and made my PhD a possibility to start with. I was also obliged on numerous occasions with invaluable advice from Dr. Steve Gunn, Dr. Srinandan Dasmahapatra, and Dr. Steve Braithwaite. In particular, Steve Gunn examined my MPhil to PhD transfer viva and Sri examined my nine month report. An article for the Journal of Artificial Intelligence Research was published in collaboration with Steve Gunn and Sri. Steve Braithwaite helped with important information about mesh networks that were used as a test-bed in our work.

Equally memorable were the social occasions I enjoyed with friends. The coffee breaks with Nadim Haque and Nishan Karunatillake gave the much needed punctuations (and the caffeine) to dislodge monotony. In my shared flat, the dinner-time quibbles with Arouna Wokeau, Victor Tan, Sanjay Vivek, and Vijay Dialani helped to hone my skill of thinking "out of the box". Also enriching were the meetings (mostly over rice and chicken curry) with Diptesh Kayal, Sourish and Suparna Banerjee, Dipayan Das, and Susanta Ghosh to relish some good old-fashioned Bengali culture!

While I was living life in the UK, my family enacted a crucial role from faraway Calcutta, by being there for me always for anything. I am blessed to have ma, baba, mashi, and pishimoni next to me. Finally, a very special thanks to my better half, Atreyi, for her encouragement, care, and support, that carried me forward. My success, if any, would not be without them.

To Mashi

Chapter 1

Introduction

Research in the field of distributed systems has been primarily focused towards conceptualising, designing, and implementing techniques that use autonomous problem solving entities (also, termed as *agents*) to interact in flexible, effective and efficient ways (Bradshaw, 1997; Jennings, 2001). Such "multi-agent" systems (MAS) are proving useful in a variety of real-world applications such as Grid computing (CCGRID), the Semantic Web (Lee et al., 2001), planning and scheduling in supply chains (Denkena et al., 2004), smart sensor networks (Viswanathan and Varshney, 1997), and network routing (Minar et al., 1999) among others. Now, although these applications are highly disparate, they have a number of properties that are common to complex, distributed computer systems. First, the task objective is too complex to be effectively implemented by a single, centralised problem solver. In a network routing application, for example, a centralised routing server that takes all routing and monitoring decisions for the entire network would be complex to design, difficult to scale, and a vulnerable single point of failure. Second, with multiple, distributed problem solvers that solve only parts of the overall problem, there should be effective ways to tie together their individual solutions to generate a consistent global solution: thus the activities of the distributed problem solvers should be coordinated in some way. For example, in a distributed sensor network, the separate pieces of (possibly overlapping) information (of, say, partial tracks of moving vehicles) collected by the individual sensors need appropriate integration to generate a coherent solution (e.g., a complete picture of all vehicle routes) useful to the user. Third, most of these applications are deployed in environments characterised by unpredictable and continuous changes (Hewitt, 1990). Thus, appropriate measures are needed to respond to such changes and generate solutions with some performance guarantee when building a robust solution. For example, in a transportation system owned by a delivery company, there should be provisions to cope with unforeseen situations such as hazardous weather condition or road blocks to prevent a complete breakdown of service.

Given these characteristics, a multi-agent solution is a natural way of tackling the problem. This is because agents — autonomous, goal-directed, reactive problem solvers capable of communicating and interacting with one another (Wooldridge, 2002) — possess the necessary properties to meet the above requirements. For example, the property of autonomy allows agents to act without centralised control. In addition, they are capable of adapting their behaviour in response to changing environmental conditions so that the overall task objective is met (being goal-directed and reactive). In so doing, they will invariably need to coordinate their actions with one another to minimise the occurrence of mutually conflicting partial solutions. Communication can be used as the key mechanism to achieve such coordination. Thus, a network of agents can be used in distributed applications where they individually solve, in a reactive, goal-directed manner, parts of an overall problem and share information via communication to generate the global solution. Such interactions, therefore, represent the *cooperation* of multiple distributed agents working towards solving a common goal.[1]

Now, in most distributed systems, one of the most fundamental problems that should be solved is that of *resource allocation* (RA). For example, in the Grid, the distributed computational resources such as storage and processor cycles need appropriate sharing and allocation so that the demands of the users of these resources are met optimally. In an analogous way, in supply-chain scheduling, the amount of time and the order in which processors attend different jobs have to be optimised to meet the required objectives such as timeliness and quality. Also, in sensor networks (and in network routing), the useful bandwidth of the sensors (routers) should be allocated or deallocated to ensure both bandwidth usage efficiency and effective sensing of the target (maintaining network quality of service).

To this end, this research specifically focuses on building a cooperative MAS solution for distributed RA. However, *resource allocation* applies to a variety of tasks and different distributed systems are designed for handling tasks that require different modalities of RA. The following examples illustrate some representative cases.

Allocation of Computational Resources in the Grid: One of the aims of Grid computing is to allocate appropriate resources (CPU, storage, etc.) for the execution of heterogeneous distributed applications (Foster and Kesselman, 1998). For example, a bioinformatician, investigating the characteristics of a chemical compound, can record his research methodology, search for logs of similar experiments conducted by fellow scientists, run experiments, and store the experimental results for future use by the community in a Grid infrastructure. Thus, in such a scenario, the resource allocation and scheduling module allocates the computational resources among the jobs that are submitted by the Grid users.

[1] An orthogonal approach of designing MAS solutions is by considering agents that act as selfish individuals trying to maximise their own utility. Such *non-cooperative* systems are discussed in chapter 2.

Allocation of Satellite Images: In the system PLEIADES (CNES, 2001), an earth observation satellite, jointly funded by organisations across different countries, is used for acquiring terrestrial images (Lemaitre et al., 2003). The users of this application are the funding bodies who periodically send image requests with specifications such as the image size, quality, and priority, among others. The satellite "agent" has the task of allocating its available resources (the images) to match the incoming requests in a way that the allocation is both efficient (allowing judicious exploitation of the resources) and equitable (allowing each requester to obtain results proportional to their contribution towards maintaining the satellite). Thus, in this application, utility-based decision mechanisms are used to allocate resources among multiple users.

Allocation of Spectrum Licenses: The Federal Communications Commission (FCC) in the United States use auctions to distribute licenses for radio frequency spectra. Thus, the entire set of licenses represent the resource pool that is owned by the FCC. It then has the task of allocating these licenses for different bandwidth segments (used for different services such as mobile phone, radio broadcast, or taxi calling service) and covering different geographical regions, among competing applicants. Typically, multiple rounds of simultaneous auctions are conducted to allocate the different licenses. Thus, in this system, an auction protocol is used to simultaneously allocate resources to multiple agents (the bidders).

In the context of our research, however, a "task" is used to refer to an activity extended in time that requires the participation (actions) of multiple agents in any order for completion. Specifically, the participation of the agents involve allocating resources that are available to them such that the task gets successfully accomplished. This is a more appropriate representation of *distributed* task processing than some of the applications listed above where the task is simply an (often, centrally controlled) allocation of resources among multiple agents. Now, we identify that in many instances of distributed task processing, the agents need to take *sequential* decisions to accomplish tasks. This implies that in order to complete such a task, the actions of different agents need to be chained together from the "start state" (when the task processing is initiated) to the "finish state" (when the task processing ends). We focus specifically on those types of tasks in which resources are allocated by distributed agents in a sequential fashion. Several examples of this type of applications exist in practice. A few are listed in the following:

Manufacturing Assembly Line: In several manufacturing industries (e.g., automobile, computer hardware, soft drinks, and so on) the various parts of the manufactured unit (e.g., a car, a computer part, a bottle of drink) are assembled by different automated assemblers in a fixed sequence. Each assembler devotes a certain amount

of time doing a particular action to put together a specific part of the unit. Hence, each assembler represents the resources of the assembly line which are to be appropriately allocated in sequence to successfully complete the assembling task.

Bandwidth Allocation for Call Routing: In circuit switched networks, in order to connect a call from its source to destination, nodes should allocate parts of their available bandwidth so that a continuous circuit is formed. Thus, the nodes transmitting the call need to carry out the allocation in sequence starting from the source node. The bandwidth reserved by each node represents the resource allocated for the completion of the task: to successfully connect the call.

Distributed Vehicle Monitoring: In this application, distributed sensors perform the task of tracking vehicle movements in their surrounding environment. Each sensor can sense vehicles when they come within its reception range. To track each such target, a sensor allocates a part of its radio bandwidth for the time the target remains within range. So, the sensing acts of individual sensors need to be integrated (be in sequence) to derive the complete track of a vehicle. Thus, allocation of resources (assigning bandwidth) proceeds sequentially between nodes to complete the task (acquire a complete track).

In this context, we choose bandwidth allocation for call routing as the example test bed to demonstrate how our solution can benefit such systems. Our choice is justified since an effective solution to the problem of distributed routing in networks is of fundamental importance in present day computer science (Jones, 1992; Julian et al., 2004). Nevertheless, it is envisaged that the results of this research should also be more broadly applicable to the other instances of this type of RA task.

The particular type of network we consider in this thesis is that of a wireless mesh network (Chandler et al., 1993). Here, a mesh network is defined by a set of wireless nodes able to radio-communicate with their direct neighbours. They have limited bandwidth and operate on batteries and solar power and, hence, are designed to consume as little power as possible. This puts severe restrictions on the duration for communication. Also, the nodes can fail due to battery power outage. A set of fixed or mobile handsets may be connected to these nodes representing the users of the networks (the points of call origin and destination).[2] In this system, therefore, the task is to route calls from the node of origin to the destination using the limited node bandwidths (representing the network resource) and accounting for the unpredictable node failures. Hence, the nodes have to sequentially allocate their bandwidths in order to successfully accomplish the task of routing the call from its source to destination. Now, the above-mentioned

[2]Such networks, typically installed in developing countries, where large areas and lack of resources prevent the installation of traditional technologies, are particularly useful for providing Internet access in educational environments (Moreau et al., 2001).

characteristics of this type of network make it a suitable test bed for a MAS solution. Specifically, in any typical MAS, the agents have limited knowledge of the entire system state (similarly, each node in a mesh network does not know about the entire network state), they can typically communicate with only a subset of the other agents at any given time (similarly, the network nodes can communicate with those that are within their radio range), they have limited computational and communication capacity (just as the nodes have limited processing power and bandwidth), and they require the participation of other agents to execute tasks (multiple nodes are required to route calls). Hence, this application provides us with a practical test bed for the deployment and evaluation of our MAS solution. This is in contrast with the approach of analysing MAS solutions on hypothetical problem settings, the results of which are typically limited by the ad-hoc assumptions made by the designer and the lack of real-world constraints.

Against this background, in the following section 1.1, we first outline our thesis regarding a MAS solution for sequential RA tasks in distributed systems. Subsequently, the research challenges that need to be overcome to substantiate our thesis are discussed. In section 1.2, an overview of the contributions of our research in attempting to address these challenges is presented. Finally, section 1.3 summarises the organisation of the remaining document.

1.1 The Aims and The Research Challenges

We note that while processing tasks, the agents can be acting upon multiple tasks and sharing the task environment. Moreover, the quality of the final outcome of such distributed processing depends upon the fidelity with which each agent performs while sharing the environment with other agents. Thus, the agents must estimate the behaviour of one another so that their actions can be coordinated and scheduled appropriately to achieve consistent solutions. In addition, to resolve potential conflicts between their actions, they should also share information such that they can have estimates of the states of other agents. More importantly, they should have ways to adapt the task execution process dynamically to cope with changes in the system environment. In this context, our thesis of a MAS solution for sequential RA tasks in distributed systems is:

> *Cooperative information-sharing achieves effective state estimation to generate the adaptive behaviour required for efficiently solving sequential resource allocation tasks in dynamic distributed systems.*

Three research challenges are identified as the cornerstones of a MAS solution that can substantiate this thesis. They are described in the following.

1.1.1 State Estimation

In MAS, individual agents can typically monitor the environment in their immediate vicinity. Such local observations provide information about both the physical environment (e.g., bandwidth usage level of network nodes) and the behaviour of other agents (e.g., routing decisions taken by nearby routers). However, an agent also requires the knowledge of the physical environment and agent actions that it cannot directly observe. This is essential to ensure that the partial solutions to the overall task, generated by the individual agents, are consistent with each other and, hence, a better quality overall solution is produced. In network routing, if an agent decides to route packets based solely on the network state in its local neighbourhood, it will result in an inefficient use of the overall network bandwidth and, consequently, a degraded quality of service. This is because routing decisions are taken without considering how the state in different parts of the network would effect the route of a packet. Thus, it would result in increased congestion in certain portions of the network, while certain other parts would remain relatively unused, leading to increased delays and packet losses. On the other hand, by explicitly reasoning about the effects of the environments and actions of other agents, a better global solution will be achieved. For example, if agents had estimates of the network load values at other routers they can determine the possible effects on delay (due to congestion) and packet losses caused by routing along a particular path. Using such assessments, they can take more informed and potentially more effective routing decisions.

To do this, however, an agent requires some knowledge about the environment that it cannot directly monitor; in other words, the estimates of those states that are beyond its local observation space. Thus, the first research challenge is to develop a mechanism that would allow the agents with localised knowledge to maintain good quality estimates of the system's states.

1.1.2 Cooperative Information-Sharing

The preceding discussion highlighted that knowledge of the unobserved states is essential for generating mutually consistent partial solutions to a global task. Information about such states can be obtained by allowing the agents to communicate their local state values with one another. Such communication would ensure that the agents gain access to the information that they cannot directly observe and enable them to maintain reliable estimates of the states. This is essentially a way of assisting each other (a *cooperative* MAS) in generating better solutions to the overall problem by alleviating the partial observability constraint. However, the information-sharing should be designed in a way that while it serves the above-mentioned purpose, it should also attempt to limit the cost of communication (bandwidth usage for transmitting state information). For example,

not all of the information available locally to one agent might be required by another agent. Moreover, certain local information of an agent can be required by multiple other agents. In addition, it is important to consider when a certain agent would require a certain piece of information. Examples of these considerations in the network routing problem are as follows: an agent can transmit its load estimate of a particular network region to another agent that is handling calls that need to be routed through that region; certain critical information (such as, node failure) which is known to an agent may need to be shared with multiple other agents (for example, those that were using the failed node to route calls); by examining the history of communication with another agent, an agent can determine whether a certain change in its local state would be useful for that other agent at that time. These considerations would improve the effectiveness of the information-sharing strategies towards generating better cooperative MAS.

In summary, therefore, the second research challenge is to design an information-sharing protocol that would be both *effective*, allowing the agents to remain updated about the system states, and *efficient*, requiring a communication overhead that is within tolerable limits.

1.1.3 Adaptive Decision Making

Cooperative information-sharing is important to allow the agents to maintain estimates of those states that they cannot directly observe. However, system states change, often in an unpredictable manner (as in dynamic environments), not all of which can be anticipated a priori. Thus, the agents should have a decision-making mechanism that is able to respond effectively (by adapting the actions selected) to changing conditions. For example, the decision to route a call along a particular route should be adapted in response to a critical state change (such as, node failure, or overload) in one or more of the nodes that are currently being used. A good decision strategy may even allow the agents to build predictive estimates of such changes and hence, allow additional flexibility in action selection to counter these changes.

Against this background, the third research challenge is to allow the agents to effectively adapt their actions (by updating their estimates about the state changes) in response to dynamic and unpredictable changes in the environment so that they can maintain quality of performance.

1.2 Research Contributions

This research aims at addressing the above-mentioned challenges to establish the thesis outlined before. In so doing, it achieves a number of important research contributions

to the state-of-the-art in cooperative MAS used for distributed problem solving. In particular, this research starts by proposing a novel communication protocol among cooperative agents. This mechanism allows the agents to inform one another about their local state values in a timely and accurate manner. Thus, this contribution addresses the research challenges of state estimation (section 1.1.1) and cooperative information-sharing (section 1.1.2). Moreover, the agents use Q-learning (Watkins and Dayan, 1992) (a variant of the more generic reinforcement learning (RL) algorithm (Sutton, 1988)) to build statistical models of the dynamics of the system states from the information distributed by the afore-mentioned communication protocol. Since Q-learning allows the learner to generate a model that is both robust (an accurate representation of the actual states) and adaptive (effectively reacts to changes in the states), the agents using such models are capable of taking effective decisions regarding RA tasks in decentralised systems. Hence, this approach addresses the research challenge of adaptive decision making identified in section 1.1.3. The rest of this research goes on to extend the communication protocol with the objectives of improving its performance to handle more dynamic environments and to reduce the communication overhead it incurs. The following discusses the above-mentioned contributions of this research in more detail.

1.2.1 A Novel Information-Sharing Protocol

One of the research challenges identified earlier in this chapter is that of providing information about system states to agents having localised views so that they can coordinate their actions effectively (section 1.1.2). Against this background, this research starts by developing a novel information-sharing mechanism to allow the agents to share the values of their locally observable system states. In particular, such cooperative information-sharing occurs between those agents who work together in solving a given task (a large MAS can have many simultaneous tasks being executed by different subsets of agents). Specifically, the novelty of this mechanism is that it advocates that the cooperative agent group share information only after they complete executing a task. A formal model of this mechanism is developed in chapter 3. This chapter further establishes, via theoretical analyses, that this mechanism generates a more effective cooperative MAS solution in comparison to a relatively standard method of cooperation based on communication between neighbouring agents.[3]

In chapter 4, this information-sharing protocol is used to develop a set of communication heuristics that are empirically evaluated in a simulated circuit-switched call routing application. Moreover, in this simulation-based empirical study, Q-learning is used as

[3]Note that a "neighbourhood" in a MAS is based on the criteria that define inter-agent relationships. For example, relationships can be based on some form of complex functional dependencies between agents (such as, supplier and consumer in a supply chain) or simple physical properties (such as, geographical proximity in a sensor network application).

the mechanism to generate robust and flexible state estimates from the communicated information. Such estimates are then used by the routing agents to perform decentralised bandwidth allocation. In these simulations, our approach is compared against a range of state-of-the-art learning algorithms also used in network routing problems. Experimental results indicate the superior performance of our approach and, therefore, provide empirical evidence to substantiate the theoretical guarantees of chapter 3. Thus, although there have been other cooperative MAS solutions based on information-sharing, our protocol achieves the best performance. Specifically, the above-mentioned results have been published in the Journal of Artificial Intelligence Research (JAIR) (Dutta et al., 2005b) and at the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004) (Dutta et al., 2004).

Against this background, therefore, this contribution addresses all the three research challenges identified before. Specifically, the communication protocol enables the agents to share information cooperatively (section 1.1.2). Moreover, Q-learning, coupled with this protocol, allows for state estimation (section 1.1.1) and adaptive behaviour (section 1.1.3).

1.2.2 Improving Performance of the Information-Sharing Protocol in Highly Dynamic Environments

The communication protocol of chapter 3 prescribes the sharing of information only *after* the completion of tasks by cooperative agent groups. It should be noted, however, that in dynamic MAS, task execution may fail without prior indication and in unpredictable ways. For example, in a transportation management system, delivery vehicles may not be able to reach a warehouse due to unforeseen situations like hazardous weather, road blocks, or breakdowns. Similarly, in a network routing scenario, a node failure can lead to the subsequent failure of call attempts involving the failed node. Thus, such incidents can cause serious disruption in the normal functioning of the system and can degrade its performance significantly. Our communication protocol, relying on sharing information only after tasks have been completed, therefore, would not be able to distribute the critical information related to such disruptive events. Thus, the agents would essentially remain uninformed about the causes of the failure of a given task. This, in turn, would prevent them from taking appropriate counter measures to deal with the effects of such failures. Against this background, this research identifies the following broad objectives that a cooperative MAS should meet to effectively deal with failures:

Adapt behaviour post failures: This objective resonates with the adaptive decision making research challenge identified in section 1.1.3. However, it particularly aims at generating effective responses of the MAS when sudden and severe disruptive events occur. For example, such adaptiveness may imply the selection of alter-

native routes by routing agents when a node fails. Appropriate responsiveness to emergency events would, in turn, ensure that the system performance remains within tolerable limits.

Diagnose failures: In addition to effectively adapting agent behaviour after failures, it is important to identify the failure itself. This is because, in decentralised systems with no central monitoring and management centre available, the individual agents or groups of agents should be able to detect the cause for failures. Such capability would, in turn, enable the system to take appropriate measures to recover from failures. For example, if the individual nodes can detect a network failure and raise an alert, then network support personnel could be summoned and the failure recovered.

To achieve the above-mentioned objectives, this research adopts the approach of extending the communication protocol of chapter 3. Specifically, this extension allows the cooperative agent groups to share information not only after task completion but also after task failures. Thus, this essentially expands the set of events after which the agents are allowed to communicate. By so doing, the agents can inform each other about the changes in the system states caused by failures. Such information, in turn, allows them to update the estimates about their non-local states which further lets them modify their action-selection process. In this way, the extended protocol attempts to enforce appropriate adaptivity in the agents' behaviours. This is a novel approach in that it is, to the best of our knowledge, the first approach to use simple cooperative information-sharing as the mechanism for generating robust adaptive behaviour of a decentralised system when unpredicted and hazardous events occur. This protocol is the topic of discussion of chapter 5. In particular, an empirical evaluation of this protocol is done in the network routing test bed by comparing its performance against the implementation of chapter 4 when network failures occur. Results from these simulation-based experiments indicate that the extended protocol achieves superior performance than the basic protocol when failures occur. Specifically, it is more effective in terms of recovering the system performance post failures. More importantly, it serves as the basis for a mechanism for distributed failure detection. Thus, our extended protocol achieves both the above-mentioned objectives and also addresses the broad research challenges of state estimation and adaptive behaviour via cooperative communication identified before. These results were published at the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005) (Dutta et al., 2005c).

1.2.3 Limiting Communication Cost of the Information-Sharing Protocol

Chapter 5 aims at improving the basic communication protocol in terms of maintaining effective system performance against disruptive events by enhancing the scope of information-sharing among cooperative agents. While this approach does satisfy its objectives, it achieves this at the cost of using a higher communication overhead. This is because communicating more (both after task completion and failure as opposed to only after completion) simply implies that the agents transmit more messages. Now, in any resource-constrained system, such as a limited-bandwidth mesh network, an increase in the communication overhead acts as a performance deterrent. Thus, too many or too large information-bearing messages in the network application would exhaust the available bandwidth from transmitting the calls. Hence, the call throughput would plummet causing a performance degradation.

Against this background, this research investigates ways of improving the efficiency of the extended communication protocol of chapter 5 by controlling the latter's communication overhead. At the same time, an attempt is made to retain the advantage of this protocol in terms of maintaining performance quality when failures occur. The approach adopted to achieve these objectives is as follows.

The objective of sharing local state values is to allow the agents with restricted views of the system states to remain informed about the relevant non-local states. Nevertheless, transmitting too much information leads to a high communication overhead. In this context, it is identified that such overhead can be restricted if information is shared in a selective fashion. In more detail, if an agent can identify that its local state information is redundant for another agent then it can do without transmitting it and, thus, save some communication overhead. Here, redundancy is measured in terms of "similarity" between information; an agent treats a certain state information value as redundant if it has already received a similar value of the same state.

Given this, chapter 6 develops a mechanism to aid such decision making of agents. Specifically, this mechanism allows the agents to retain in memory their past history of communication which, in turn, allows them to verify whether a given piece of information would be redundant to a specific recipient agent. If such a redundancy is detected, then that information is not transmitted. The recipient agent, in this case, extracts the information from its communication history that it did not receive from the transmitter (since the latter had detected a redundancy in transmission) and uses it for its decision making. In this manner, therefore, transmission is restricted but without letting the recipient effectively lose any information. Hence, it is envisaged that this mechanism would reduce communication overhead without sacrificing performance. This way of limiting communication overhead by making information-transmission selective but

without sacrificing performance quality is a novel design of a cooperative MAS solution for sequential RA tasks.

Chapter 6 implements this selective information transmission mechanism for an empirical evaluation in the network routing test bed. Experimental results from the simulation-based studies indicate that a significant saving in the communication overhead is achieved compared to the protocol of chapter 5, but without sacrificing call throughput across a wide range of environmental settings. Hence, this work also addresses the broad challenges identified before: adaptive decision making by way of (selective) information exchange among cooperative agents. This work is submitted for publication at the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006) (Dutta et al., 2005a).

The work in this thesis has contributed in part or in full to the following publications:

- P. S. Dutta, N. R. Jennings and L. Moreau. Adaptive Distributed Resource Allocation and Diagnostics Using Cooperative Information-Sharing Strategies. Submitted to the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006).

- P. S. Dutta, C. V. Goldman, and N. R. Jennings. Efficient communication using selective information exchange in resource-constrained multiagent systems. Submitted to the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006).

- P. S. Dutta, N. R. Jennings, and L. Moreau (2005). Cooperative information sharing to improve distributed learning in multi-agent systems. Journal of Artificial Intelligence Research (JAIR), 24:407–463, October 2005.

- P. S. Dutta, N. R. Jennings, and L. Moreau (2005). Sharing information for Q-learning-based network bandwidth estimation and network failure detection (poster). In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), pages 1107–1108, 2005.

- P. S. Dutta, S. Dasmahapatra, S. R. Gunn, N. R. Jennings, and L. Moreau (2004). Cooperative information sharing to improve distributed learning. In AAMAS-04 workshop on Learning and Evolution in Agent-Based Systems, pages 18–23, 2004.

1.3 Thesis Structure

The rest of this thesis is structured as follows:

- Chapter 2 reviews the most relevant previous research that has developed theories and applications of cooperative MAS solutions in the context of distributed RA problems.
- Chapter 3 develops a novel information-sharing protocol for the design of a cooperative MAS solution to distributed sequential RA problems. Its advantages in generating high quality state estimates are established theoretically using a formal analysis.
- Chapter 4 designs communication heuristics based on the protocol of chapter 3 in the network call routing application to verify its utility in practice using empirical analysis.
- Chapter 5 then improves upon the implementation of chapter 4 to make the protocol more effective in dealing with highly dynamic environments, caused by such phenomena as network failures.
- Chapter 6 further improves the performance of the protocol by reducing its communication overhead with the aid of selective communication.
- Chapter 7 summarises how this research successfully establishes our thesis and the contributions it makes in doing so. Finally, it identifies future research strands that can potentially stem from our work.

Chapter 2

Related Work

An effective MAS solution for distributed problem solving should allow multiple, distributed problem solvers (processors, network nodes, agents, etc.) to cooperate and coordinate in order to generate quality solutions for complex tasks that cannot be produced by any one of them acting alone. Such technology is required to meet the challenges in solving some of the most fundamental problems that form the core of present-day computer science (several examples of these were cited in section 1). To this end, in this chapter, some of the most successful research on cooperative MAS is analysed and its suitability and shortcomings highlighted with respect to the requirements elaborated in chapter 1. Also, it is explained how our research builds upon the existing concepts and attempts to address their limitations. The rest of this section is divided into the following.

In section 2.1, a review of the most relevant theoretical and empirical research on cooperative MAS is presented. This review emphasises that building effective cooperative MAS requires a generic mechanism, capable of generating reliable estimates of unobserved states from limited interactions, and adapting decisions in response to dynamic environments. In addition, it highlights that the agent decision mechanism should not depend on pre-encoded cooperation rules. Instead, the agents should be able to adapt their cooperative behaviour online in response to changing environmental conditions. Otherwise, the mechanism would not be effective when deployed in dynamic environments.

Then, section 2.2 discusses some applications for cooperative MAS based on RL techniques. In this case, RL provides the basic framework for adaptive decision-making. Specifically, it uses an abstract representation of the environment and generates a probabilistic mapping of environment states onto possible actions. Subsequently, the agents, using such techniques, can choose the action with the highest estimated utility in a given environment. Since no domain-specific knowledge or pre-defined rules of coopera-

tion are required to build such state-action mappings, RL can act as a powerful tool for an effective implementation of MAS.

Finally, section 2.3 summarises how our research attempts to build upon and address the limitations of existing literature. Also, it delineates the key contributions that our research makes in so doing.

Before going into the details of the review, it should be noted that, in this thesis we are specifically concerned with *cooperative* MAS. As stated in chapter 1, an orthogonal approach to building MAS solutions to complex distributed problems is to design agents that act as *selfish* individuals that try to maximise their individual utilities (in cooperative systems, on the other hand, it is the overall system utility that the agents try to maximise together). Coupled with the appropriate *mechanisms* to interact with each other or, equivalently, the rules of the play, such selfish agents can act towards solving distributed problems. Typically, researchers in this area draw upon algorithms from a combination of game theory (Fudenberg and Tirole, 1991; Myerson, 1997), auctions (Klemperer, 2004; Milgrom, 2004), and control theory (Brogan, 1990) to develop such non-cooperative solutions. Although non-cooperative algorithms have been proven successful in a variety of applications (Gibney et al., 1999; Haque et al., 2005; Rachlevsky-Reich et al., 1999), there has so far been no evidence that they are generally more preferable to cooperative solutions. Nevertheless, non-cooperative algorithms have been criticised for some restrictive assumptions such as perfectly rational agents (Maynard-Smith, 1982) and the high computational complexity (Kalagnanam et al., 2001; Sandholm and Suri, 2001) that is typically associated with the distributed mechanisms used in such systems. Given this, we will not look at non-cooperative MAS in this chapter.

2.1 Cooperative Multi-Agent Systems for Distributed Problem Solving

In the following, a review of some of the key research works in cooperative MAS is presented.

2.1.1 Functionally Accurate Cooperation

Agents deployed for solving a complex task in a distributed environment (for example, in distributed network monitoring), can have uncertain, incomplete, and erroneous information about the global state of the system. This is because of their restricted capabilities to observe the dynamics of the entire system and to communicate and share information with others. To be able to accomplish the tasks despite such uncertainties,

the functionally accurate, cooperative (FA/C) approach advocates the exchange of partial, tentative solutions of local problems among agents (a distributed speech recogniser application is developed by Erman and Lesser based on this concept (Lesser and Erman, 1988)). The main objective of this method is to ensure, as much as possible, consistent partial solutions develop. In turn, this helps to generate predictive information about future partial solutions that furthers the build up of a consistent global solution. The timeliness of the exchanges of partial solutions is essential to reduce searching for consistent solutions, hence, making distributed problem solving faster. It should be noted that the cost to obtain complete and up-to-date information to build a completely consistent solution can be prohibitively large because of delays in synchronisation and communication. Hence, in such situations, it is more practical (cost effective) to achieve a global solution (that may have a tolerable degree of inconsistency) via the timely exchange of partial, tentative solutions. Thus FA/C is based on the requirement of (see section 1) communication to generate consistent estimates of the global problem solving scenario.

In the research reported here, the agents are situated in different regions of a mesh network without complete and up-to-date information that would have helped them take consistent decisions with respect to allocating appropriate resources for establishing calls. Also, not all resources (bandwidth) may be available for any one agent to place a call. The uncertainty in the failure of network nodes furthers the complexity of the situation. Thus the agents, in this context, can adopt principles of the FA/C approach. They need to share, via communication, their partial local views of the network states so that they can allocate node bandwidth in a manner consistent with each other and maximise the likelihood of successfully routing calls. This implies that all agents place calls through the most efficient channel although their individual actions are asynchronous. Nevertheless, FA/C only discusses "what" is to be done, viz., agents should cooperate with partial solutions to reach an acceptable solution quality. It does not provide a recipe of "how" it could be achieved. Against this background, our thesis proposes a specific protocol for sharing information which is then used (by a Q-learning algorithm) to generate estimates of the distributed states.

2.1.2 Organisational Structuring

Although FA/C allows distributed agents to handle inconsistency by sharing partial solutions, the agents, in general, are ignorant about the broader picture of the problem solving scenario. Such a lack of knowledge can lead to excessive communication, duplication of effort, and uneconomical use of resources while solving a complex problem. To alleviate this bottleneck, some researchers have incorporated organisational "structures" — patterns of information and control relationships that exist between the agents and the distribution of problem-solving abilities among them (Carley and Gasser, 1999) — into the agent models. The purpose is to augment FA/C systems by giving an agent

high-level knowledge about how the system solves problems, the roles that agents play, its own position in the network, and how are they connected. Just as the notion of "bounded rationality" (used in human organisation theory (Galbraith, 1977; March and Simon, 1958; Simon, 1957), implying the limited information processing capability of human beings) has prompted organisations to form and solve complex problems beyond the abilities of an individual, organisational structures have been used to bolster problem solving by a group of cooperative agents. Also, in a way similar to how a human organisational structure can be found to work optimally towards achieving the organisation's objective in a given task environment (viz., with increasing uncertainty of the task domain, hierarchies emerge, whereas increasing complexity demands loosening of rigid structures within organisations (Fox, 1981)), a particular organisational design can be chosen for a cooperative distributed agent system depending on the problem domain. Imposing organisational structures, therefore, is to efficiently resolve the requirement (see section 1) of maintaining high-fidelity estimates of the portions of a system that the agents cannot directly monitor. The effectiveness of this concept was demonstrated in practice by Lesser and Corkill who developed a system of distributed vehicle monitoring agents — the DVMT (Lesser and Corkill, 1983). More recently, it has been demonstrated that a computational organisation is an effective metaphor for formally specifying multi-agent systems and also for analysing their behaviours (Zambonelli et al., 2001). Moreover, the organisational relationships based on roles played by different agents in a MAS are exploited to improve coordination performances in (Soon et al., 2004).

The agents considered in this thesis are also characterised by bounded rationality — they have incomplete information about the problem solving activity of the entire network. Here, some knowledge about the availabilities of various resources of other agents to perform certain tasks would allow them to take informed decisions about which agent to interact with and when. Wasteful communication and uneconomical resource usage would, then, be minimised. However, instead of making agents follow pre-defined structures of roles and relationships, they should be able to identify such properties by interacting with and analysing the behaviour of other agents. This would make the multi-agent cooperation model (that is the aim of this research) more flexible and applicable across domains unlike the ones in existing literature that use fixed organisation structures.

2.1.3 Sophisticated Local Control

Effective cooperation among multiple problem solving agents cannot be achieved by allowing them to simply communicate messages. The sophisticated local control methodology advocates that such agents also need to understand the implications of their actions on those of other agents and on the overall problem solving behaviour of the system.

Such understanding forms the basis of their decisions such as how to exchange information to resolve inconsistencies, whom to interact with to improve cooperation by building complete solutions, what information exchange can achieve that objective, whether to take different organisational roles, and the like.

The partial global planning (PGP) approach (Durfee and Lesser, 1991) emphasises integrating sophisticated control into an agent's decision making. In this framework, agents form contracts, plan their actions and interactions, negotiate over plans, use organisational information to guide their planning and problem-solving decisions, tolerate inconsistent views, and converge on acceptable network performances in dynamic environmental conditions. Each agent maintains its own set of PGP's — a set of local plans that represent the agent's view of the global problem solving situation. The various "slots" in a PGP frame, that represent an agent's goals, actions, and relationships with other agents, are dynamic structures. They are updated by the exchange of local, partial plans among agents and reflect the most recent network scenario in terms of achieving the global solution. Hence, this methodology caters for the requirements (see section 1) of state estimation and strategic information-sharing.

Elaborating on the PGP approach, Decker and Lesser advocated the importance of modelling multi-agent coordination based on the characteristics of the task environment. Their aim was to study how agent architectures arise naturally from their task environments. To this end, they developed the *TAEMS* (Task Analysis, Environment Modelling, and Simulation) framework to model task structures (Decker, 1995b,a), where task models can be expressed at three levels of detail viz., *objective* (the actual, true task), *subjective* (an agent's perspective of a task), and *generative* (a set of statistical properties that can be used to generate the objective and subjective descriptions). Using *TAEMS*, coordination is achieved by a set of mechanisms that address three broad areas of agent behaviour: how and when to communicate and construct non-local views of the current problem solving situation; how and when to exchange the partial results of problem solving; how and when to make/break commitments made to other agents about what results will be available and when. The generalised partial global planning (GPGP) approach consists of a group of coordination mechanisms based on the above broad behavioural types of agents (Decker and Lesser, 1995; Decker and Li, 2000). Specifically, depending on the characteristics of the task environment, agents select the appropriate coordination mechanism. Unlike PGP, GPGP distinguishes local scheduling of an agent from its coordination activities; while the coordination mechanisms provide an agent with non-local views of the problem, its local scheduler creates plans (including both local actions and non-local effects via such actions) to improve global system-wide utility by using information from the coordination mechanisms.

The principles underlying both the PGP and GPGP frameworks address the requirements of an effective cooperative MAS as identified in section 1. However, both PGP

and GPGP employ, albeit flexibly, a set of predetermined coordination mechanisms. Preplanned coordination can prove to be inadequate against all sorts of contingencies that can occur in domains where agents maintain incomplete, incorrect views of the world state (which change non-deterministically) and may even fail without prior indication. In contrast, the approach researched in this thesis, that of learning to map the information about the world states to actions which would guarantee the improvement of the global system performance, requires no such pre-specified coordination rules. An additional goal of this research is to develop an analytical model of the cooperative MAS so that formal analyses of performance can be performed, a feature lacking in PGP and GPGP.

2.1.4 Teamwork Based on Joint Intentions

Probably the most comprehensive cooperative MAS framework existing in current literature (e.g., (Jennings, 1995; Rich and Sidner, 1997)) is STEAM (Tambe, 1997). At its core, STEAM follows the principles of the joint intentions theory of Cohen and Levesque (1991) and the principle of joint commitments by Jennings (1993) as fundamental frameworks for multi-agent coordination. The joint intentions theory is developed on the notion of a "joint persistent goal" (JPG) maintained by a team of agents which dictates that all agents in the team are jointly committed in doing some team activity, while mutually believing that they are doing it. The notion of JPG synchronises the activities of the team. Agents in STEAM arrive at a JPG by exchanging speech acts: "request", that they use to announce their individual partial commitments about attaining the global goal, and "confirm", which establishes that an agent has the same partial commitment to the one who made the "request". While joint intentions ensure synchronous agent activities within a team, they do not contribute towards team *coherence* — following a common solution path by all team members who are jointly committed on a common goal. STEAM borrows principles from the "shared plans" (SP) model (Grosz and Kraus, 1996) to ensure coherence. A full SP is established if the team members mutually believe in the existence of a team goal and also in a *recipe* (a set of action sequences) to achieve it. It also depends on whether a subgroup of the team has the recipe to execute each step of the global recipe. In addition, all other team members believe (and, intend that) the subgroup has some recipe to bring about the step. Thus, STEAM addresses the requirements of maintaining estimates of what the other agents are capable of and their actions. This is ensured via sharing control information.

The strength of the STEAM framework lies in the advantages of using joint commitment as the building block of teamwork models. It provides a principled framework for reasoning about coordination in teamwork, and also provides guidance for monitoring and maintenance of team activities (Tambe, 1997). Nevertheless, achieving joint belief on mutual commitments in large systems of widely distributed agents might prove to be

a performance bottleneck rather than an advantage because of the excessive communication required to achieve it. This is especially true in environments where the agents have to replan the task execution process in response to environmental changes. Although STEAM treats this issue by using a replanning protocol, it requires the re-establishment of joint commitments among all the team members. A significant amount of delay might be incurred to achieve this and the communication latency can degrade the quality of service significantly in some applications. This limitation is addressed by Kalech and Kaminka (2005), where the authors use a set of heuristics to restrict the communication and computational complexity of diagnosing large teams. However, it is important to have a cooperation model that allows the agents to continue solving the overall task without requiring them to establish a system-wide commitment whenever replanning occurs. The Q-learning-based cooperation model built upon information-sharing that is researched in this thesis is envisaged to have this advantage.

2.1.5 Coordination using Markov Decision Processes

In (Boutilier, 1999), the author attempts to design a mechanism that allows multiple cooperative agents to take the optimal joint actions in the presence of coordination conflicts. He recognises that to do this the agents need to consider not only the world state but also the state of their coordination mechanism. To this end, he presents a multi-agent Markov decision process (MMDP) formalisation of the multi-agent coordination. In particular, the value iteration algorithm, used to evaluate optimal joint actions, is extended by augmenting the state space representation with a representation of the coordination mechanism used by agents. In this work, evaluation of the optimal value function (hence, optimal joint policy) is done by reasoning explicitly about short and long-term prospects of coordination, and the long-term consequences of (mis)coordination. Analysis of several coordination problems show that depending on the expected payoff, agents sometimes decide to engage in or avoid coordination problems.

Boutilier's analysis demonstrates that agents need to reason about prospective coordination problems that can arise while selecting optimal joint actions sequentially. This work, therefore, captures the requirement of adaptive decision making outlined in section 1. Since the agents only have partial, and possibly incorrect knowledge of the entire system, they should make decisions based on the possibilities of coordination conflicts (caused by the effects of actions taken by the other agents). However, although faced with a similar challenge, Boutilier's solution cannot be applied, as is, to the research presented in this thesis. This is because, the value iteration algorithm takes a global view of the entire problem-solving situation — it assumes that all actions of all agents are correctly available for evaluation of the optimal joint action. This directly contradicts the fact that the agents have only partial observability. In this situation, applicability of the same value iteration algorithm requires at least one agent to have perfect knowledge

about the actions of every other agent at every time instant in order to compute the optimal joint action to be taken by all agents. This, in turn, necessitates all agents communicating every action that they take to at least one agent who is evaluating the policy. This is precisely the underlying principle of an algorithm developed by Nair et al. (2003) for solving a decentralised partially observable Markov decision problem (DEC-POMDP) (Goldman and Zilberstein, 2004), where the assumption of full observability is removed. However, such indiscriminate communication makes it infeasible to be realised in practice. Also, the computational load for evaluating the policy by a single agent becomes exorbitant. In an extension to their algorithm, they use some ad-hoc communication heuristics to reduce the computational load (Nair et al., 2004). However, they assume a synchronous communication mechanism where all agents need to coordinate their views of the world states at the same time. Again, this is not a practical assumption in real-life MAS applications. Our communication protocol is asynchronous and designed to provide a practical solution to such applications.

2.1.6 Cooperative Social Networks

In developing cooperative MAS, some scientists have drawn inspiration from the social networks literature (Jin et al., 2001; Kautz et al., 1997). These works simulate the growth and evolution of artificial agent societies in an attempt to understand and explain the behaviour of real world societies and how they work together to solve complex problems. Typically, such simulations use simple agents, capable of very limited problem solving abilities, who can interact with a limited subset of other agents using simple interaction protocols. In this way, an empirical model of the essential interaction dynamics in a group of cooperative problem solvers is created. The underlying assumption of these simulation-based studies is that it is unlikely for a tractable mathematical model to precisely and comprehensively capture all details of a complex distributed multi-agent system. Instead, the broad interaction patterns are simulated to simplify the analysis yet retaining the essential mechanics intact.

In this context, the work of Yu et al. (2003) is relevant to our thesis. They model agents as service centres satisfying the needs of their users. However, not all agents retain the relevant services to satisfy all user needs. Thus, they forward the users' service queries to other agents in the system so that the latter can provide the required service. Those agents who receive a request from another agent but do not have the service appropriate to the request, forward the request further. This mechanism allows the agents to explore the system for partners with matching expertise. A network is formed when agents connect with partner agents and can provide a service to each other, hence, representing mutual collaboration. Stability of a collaboration depends on the relevance of the service provided by a partner — the higher the relevance, the stronger the collaboration, the lesser the relevance, the weaker the collaboration.

This work, therefore, addresses the essential requirement of maintaining state estimates for a cooperative MAS (see section 1). By way of forwarding queries and evaluating the quality of information received for a given query, the agents are able to estimate the expertise of other agents in the system. This information, in turn, helps build and maintain cooperative relationships where the agents mutually cater to the information requirement of one another. However, in their work, the agents can, potentially, maintain direct interactions with all other agents. This is impossible in most real-world applications: for example, in the network routing problem, a node/agent can at best maintain an estimate of the load of a distant router and not directly communicate with it. In this thesis, we do not restrict our approach by assuming such direct one-to-one interactions.

2.2 Learning-Based Cooperative Multi-Agent Systems

In the following, reviews of some of the most relevant applications on cooperative MAS that use RL algorithm are presented.

2.2.1 Q-Routing in Dynamic Networks

RL is applied to the problem of cooperative distributed problem solving in a seminal piece of work by Boyan and Littman (1993). Specifically, they applied this technique to solve a network packet routing problem. In their paper, they modelled each communication node on an irregular 6×6 grid as a reinforcement learner who maintains estimates of the delays in routing packets to different destinations. To route a packet to a given destination, an agent requests each of its neighbouring agents for their delay estimates for that destination node. Upon receiving the delay estimates of its neighbours, the requesting agent forwards the packet to that neighbour which has the minimum delay estimate. It then updates its prior delay estimate for that destination with the estimate that it received from this neighbour. Specifically, Q-learning is used to update the estimates.[1] In more detail, if there is a set of $\mathcal{N}$ agents in the network, an agent a (which has a set of $\mathcal{K}_a$ neighbours) has a Q table with each entry $Q_a(d, n)$ (for all $d \in \mathcal{N}$ and $n \in \mathcal{K}_a$) equal to the minimum expected delay to reach destination d via the neighbour n. Upon sending the packet to n, a immediately receives n's minimum estimated delay of the packet for the remainder of the trip, $t = \min_{z \in \mathcal{K}_n} Q_n(d, z)$. Subsequently, a updates its previous estimated delay $Q_a(d, n)$ as, $Q_a(d, n) = (1 - \alpha)Q_a(d, n) + \alpha t$; where α (also

[1] This is a value-search algorithm (Sutton, 1988). Such algorithms work by iteratively computing the maximum expected utility over all possible actions under a given state. In this case, the action is to select the neighbour agent with the minimum estimated delay (the action with the maximum utility) in order to route a packet to a given destination.

termed as the "learning rate") is the weight that this agent gives to the recently received information t.

The authors of this paper demonstrate, using empirical studies, that their approach enables the agents to learn better policies (in terms of choosing a neighbouring agent for routing to a given destination) than a hand-coded shortest path algorithm. The differences are more pronounced when the network load increases indicating that the learning algorithm is able to adapt routing decisions (the paths along which packets are routed) under dynamic network conditions. In addition, the authors test their algorithm with changes in the network topology (by manually breaking the links between certain nodes) and in the pattern of call traffic (changing different regions in the network where calls can originate and terminate). They demonstrate that their Q-routing algorithm successfully adapts to these changes and performs better than the deterministic shortest path algorithm.

In their work, therefore, Boyan and Littman have used a simple communication protocol to allow the agents to cooperatively share their own estimates about the packet routing delays. Also, these agents use RL as the environment-modelling tool to estimate the delays to route packets across the network (not all of which they can directly monitor). In addition, RL allows the agents to adapt their routing decisions in response to environmental (network load and topology) changes. Hence, their work addresses all three requirements identified in section 1. Nevertheless, the communication protocol they have used only allows an agent to inform its immediate neighbour about its own estimates. This method would incur long latency for information to reach agents further away. As a result, considering states change continuously, the information can become outdated by the time an agent receives it. Such outdated information would then be of little use to generate reliable estimates of the non-local states. In this context, therefore, it is envisaged that by allowing state-change information to be shared between the group of cooperating agents only after task completion, the agents can maintain more accurate estimates of their non-local states. This, in turn, can improve the overall performance of the cooperative MAS than in (Boyan and Littman, 1993). Moreover, in (Boyan and Littman, 1993), the agents update their Q-estimates with the *estimates* received from direct neighbours. Note that in so doing, one learner depends on the estimates learned by another. Thus, this approach (essentially, a TD(0)-type learning (Sutton, 1988)) has the potential pitfall that "bad" estimates are propagated due to the poor learning of one agent. We attempt to address these limitations in this thesis. In so doing, we choose the Q-routing algorithm as one of our benchmarks for empirically evaluating the performance advantage of our approach.

2.2.2 Team Partitioned Opaque Transition Reinforcement Learning

The **T**eam **P**artitioned, **O**paque **T**ransition **R**einforcement **L**earning algorithm (TPOT-RL) (Stone, 2000) has the objective to make the learning task easier in a MAS by reducing the state space dimensionality. It does this by mapping the state onto a limited number of *action-dependent* features. Analogous methods of state aggregation have been used in other RL algorithms (e.g., (McCallum, 1996; Singh et al., 1995)) to reduce the size of the learning task. However, TPOT-RL differs from these approaches because it emphasises deriving a set of small yet informative features for effective learning. More specifically, these features are used to represent the short term effect of actions that an agent may take. Thus, the agents learn the utility of selecting actions with respect to their own feature space. This is especially useful when the agents cannot observe or immediately influence the actions taken by other agents (such as in many practical multi-agent settings and, in particular, in network routing). That TPOT-RL is an effective algorithm is demonstrated by its successful application across multiple domains (Stone and Veloso, 1999).

Stone and Veloso have evaluated the TPOT-RL algorithm in a simulated network routing environment. The action-dependent features in this case are the load levels of a node's adjacent links. The agents transmit their delay estimates along with a packet while routing the latter. Furthermore, these estimates, collected at the corresponding destination nodes, are distributed to the nodes who participated in the routing after fixed time intervals. Thus, TPOT-RL in fact uses communication to distribute information among the agents. Their empirical studies demonstrate that TPOT-RL outperforms (performance measure is average packet delivery time) the shortest path and Q-routing protocols when learning is done under switching traffic conditions — the algorithm is trained under conditions where the selection of packet sources and destinations are changed to form two different traffic patterns. Nevertheless, the following limitations are envisaged in this work. First, identifying action-dependent features from local observations only can lead to loss of information. This is because not all non-local state changes may be reflected in an agent's immediate state space, but such information may be required by an agent to select actions. Thus, in such circumstances, the non-local state values should be known. Second, as a consequence of the above-mentioned problem, the fidelity of the derived estimates would deteriorate. This, in turn, would decrease the overall performance of the system. Third, in (Stone, 2000), since the agents update their estimates based on others' estimates and not using the actual states, a similar shortcoming as identified in section 2.2.1 of learning bad estimates can occur. Finally, in TPOT-RL, information is distributed at regular intervals ((Stone, 2000) does not prescribe a formal way of specifying this interval). This is an arbitrary scheme which can result in large latencies in information reaching target nodes. Hence, estimates generated based on such information may not be up-to-date. Again, because

of its claimed effectiveness and broad applicability, we choose TPOT-RL as the second benchmark for empirical evaluation against our approach in which we attempt to remove the above-mentioned shortcomings.

2.2.3 Policy Gradient Search

Another approach of using RL for cooperative distributed problem solving is that of policy gradient search (Sutton et al., 2000). A *policy*, in an RL context, is a mapping from a state on to an action; it is *stochastic* if the mapping specifies a probability distribution over all possible actions given a state. The policy is a function of a set of *parameters* which are variables defining the (local) state and hence, influencing the action selection. A policy gradient search is a mechanism that tries to optimise the parameter values such that the average long-term reward of the learners is maximised. For example, in a network routing problem, these parameters can be the destination of the packet at a router (agent) and the outgoing link it selects for that (note these parameters are locally observable to an agent) and reward is a measure of utility (or, sometimes implemented as a negative cost) that an action achieves given the parameter values. In the network routing domain, the reward can be the negative trip time for a packet to reach its destination node. In the policy gradient approach, it is assumed that all agents (the individual learners) receive the reward of all actions taken by all agents at every time step.[2] It is only this reward information that is globally known by the agents. Thus the policy gradient algorithm is model free — independent of domain models and knowledge about others' states and actions. Individual agents adjust their policy parameters in the direction of the gradient of the average reward that they compute using the global reward information — hence the term *policy gradient*. Therefore, communication (of reward values) is key to allow the learners to optimise the parameter values. However, the dependence on the global reward information to update the policy parameters can be a bottleneck in systems where the communication bandwidth is limited and there is a finite latency in messages to propagate (as in most practical systems). These constraints can lead to very slow responsiveness to environment changes in agents using the policy-gradient approach.

The use of policy gradient search is used to build various RL-based MAS, e.g., (Williams, 1992), (Baxter and Bartlett, 1999), and (Peshkin and Savova, 2002), among others. All of these works demonstrate that the policy-gradient search achieves reasonable performance (in terms of average routing delay compared to other benchmark algorithms such as the shortest path algorithm). However, in both (Williams, 1992) and (Baxter and Bartlett, 1999), an exceedingly large amount of time is required for the learners to converge. This is due to the fact that the learners need the global reward information to

[2]More realistically, each agent may broadcast the reward it receives to all other agents. Hence, the agents may receive the reward signals from the entire system with some delay.

update their policy parameters and, hence, take a long time to optimise the parameter values. This puts a restriction on the applicability of this approach to build practical systems. A similar limitation is likely in (Peshkin and Savova, 2002) although the authors do not provide these results.

In contrast to uninhibited communication required in the policy gradient search approach, in this thesis, communication is used as a controlled strategy to inform the agents about the portions of the global state that are relevant to their action selection. Thus, our work is envisaged to be a better practical solution than the policy gradient approach.

2.2.4 Communication Decisions in Multi-Agent Coordination

In (Xuan et al., 2001), communication decisions are treated as integral to an agent's decision to coordinate in a cooperative, distributed MAS. The authors consider that each agent solves a local Markov Decision Process (MDP) (Feinberg and Schwartz, 2001) that generates both a communication action and a state-changing action at every decision sub-stage. The agents are only given local observability (i.e., they cannot observe the states of other agents). However, they can observe the communication actions of other agents. The important reason for introducing a communication decision in an agent's local MDP, they argue, is because communication incurs cost. Hence, an agent should employ reasoning to decide when communication is required such that the overall utility earned from the agent's decisions is maximised. In this context, this work extends the theoretical analysis of multi-agent MDP (section 2.1.5) where the agents are assumed to have global state information. Specifically, the authors propose two simple heuristics to generating communication decisions that aim to reduce the computational complexity of solving the full MDP to generate the optimal global policy.

As we are interested in studying the impact of communication on the performance of a cooperative MAS, the work reported in (Xuan et al., 2001) is related to our research. However, while they analyse *whether* communication is necessary at any stage of an agent's decision, we consider communication to be inevitable. Moreover, in (Xuan et al., 2001), the following additional shortcomings are identified. Firstly, the agents are assumed to iterate through a sequence of communicate and act stages *synchronously.* A generic approach of completely asynchronous behaviour is suitable for a practical MAS solution. Secondly, communication is assumed to be instantaneous; thus, information sent by an agent is received by another immediately. On the contrary, in real applications, there is always a finite delay associated with communication. Finally, the communication heuristics proposed are based on each agent individually monitoring their own progress towards achieving a commonly agreed upon goal. However, in distributed task processing, an agent can take a local action and the actions of multiple agents together

complete a task (more on this in chapter 3). Hence, in such scenarios, individual agents cannot monitor the progress of a task execution process towards completion; they are only capable of taking their local actions using estimates of the unobserved states.

In this context, Davies and Edwards (1997) propose version space (Mitchell, 1978) boundary sets to generate hypotheses (by agents with localised views) to describe the world states. Moreover, communication is used by these agents to share the hypotheses which can then be further refined (using set-theory operations to alter the hypotheses bounds) in order to find one that is consistent with others. Thus, this work resonates with the broad objective of our research in coordinating distributed agent actions via communication. However, our focus is not primarily on resolving coordination conflicts, but rather on developing a practical communication protocol that would generate effective coordinated behaviour in agents deployed in dynamic distributed applications.

2.2.5 Other Machine Learning Algorithms

Our approach of designing a communication protocol for cooperative information-sharing may appear to be similar to conventional supervised learning (SL) (Widrow and Hoff, 1960), where the actual outcome of a multi-stage prediction problem is fed back to the individual learners (predictors). However, the characteristic properties of SL do not sufficiently meet the requirements of a cooperative MAS solution as highlighted in chapter 1:

- In SL, only the final outcome (such as whether a prediction was "correct") acts as the source of information for the learners to update their prediction algorithm. Thus, a learner does not gain any estimate about how the states of other agents in the cooperative group change along with the outcome.
- In SL, an agent typically learns a mapping from its own actions onto the outcomes of a multi-stage prediction problem. Hence, it does not allow an agent to consider the impact of the states of other agents on the final outcome of the task. Such information is, however, crucial for the distributed agents to effectively coordinate.
- Typically, SL is used for prediction in stationary environments. On the other hand, a robust solution should provide a mechanism for maintaining high-fidelity estimates in fundamentally dynamic and uncertain environments.

Also related to this thesis is the method of using eligibility traces in which a learner is provided with the entire sequence of state transitions after a complete training episode (i.e., after starting from the start state and reaching the goal state) (Mitchell, 1997). In this technique, an agent, upon reaching the goal state after executing a series of actions, updates in reverse order (i.e., starting from the goal state and moving to the

starting state) its Q-estimates for each state transition. In a practical MAS, however, it is not possible for a single agent to observe all transitions occurring during a task processing episode as assumed in the approach using eligibility traces. Further, in a large and complex MAS, the computational load incurred by a single agent attempting to take decisions using the information about of all state-transition sequences of every task would be too prohibitive to be realised in practice. In this situation, therefore, our research contributes towards developing a practical and effective means of distributing state information to improve learning. In so doing, it removes the requirement of an agent having to maintain the entire chain of state transitions in its memory. Rather, it allows the agents to acquire a summary of the state-changes in the cooperative group that perform a task, which, in turn, allows them to take decisions for effective coordination.

2.3 Summary

This chapter presents the key concepts that underlie previous research on cooperative MAS. At the same time, it highlights the limitations in existing literature in the context of the requirements identified in chapter 1. For example, some research addresses only a subset of the three requirements (e.g., FA/C, organisational structuring, and social networks). In our work, we aim to incorporate all the requirements into our solution and, hence, address the above-mentioned limitations of existing work. On the other hand, others address all the three requirements, but under certain restrictive assumptions such as predetermined rules of coordination (leading to inflexibility in dynamic systems) as in PGP and GPGP, joint commitments (causing large communication latencies) as in STEAM, and global knowledge of agent actions (incurring communication overhead) as in the work based on MDPs. Thus, although our research is inspired by these, we still extend these approaches by removing the above-mentioned limitations. In particular, our aim is to build a MAS solution that does not rely on pre-defined coordination rules (e.g., PGP and GPGP), allows the system to function without requiring all agents to jointly commit (e.g., STEAM), and does not require global state knowledge or indiscriminate communication to coordinate the agents' actions (e.g., the MDP-based works).

Furthermore, section 2.2 identifies the limitations of the applications on cooperative MAS based on RL. These include restricted communication (the Q-routing algorithm) and excessive communication to obtain global reward information (the policy-gradient search methodologies). The review of TPOT-RL outlines potential limitations of that approach that bases action selection solely on local observations. Also, several other machine learning algorithms that are used to generate estimates in distributed systems are reviewed and their potential shortcomings identified. Although we use RL to ensure adaptive behaviour, we aim to improve such behaviour by developing a suitable mechanism for sharing information that addresses all the above-mentioned shortcomings.

In this research, we use Q-learning (a specific instance of the more generic RL algorithm) to generate robust and flexible estimates of the system states that the agents cannot directly observe. Moreover, this capability of building estimates allows the agents to adapt their actions to environmental changes. To this end, we develop a novel information-sharing protocol that distributes the agents' local state values so that they can build these estimates effectively. Therefore, we adopt the approach of using RL to build a cooperative MAS as done by other researchers, but, at the same time, advance the state-of-the-art by incorporating information-sharing to improve the system performance. In this way, we address all the essential requirements for addressing our thesis identified in chapter 1. To the best of our knowledge, our information-sharing protocol in both novel in its principle of application and, among the currently available protocols that can be used for sequential RA tasks in distributed systems, the most effective and successfully deployable in real applications.

Chapter 3

Post Task-Completion Information-Sharing Protocol

Chapter 1 identified cooperative information-sharing and adaptive decision making as two key challenges in designing an effective solution for sequential RA tasks in distributed systems. To this end, this chapter develops a novel information-sharing protocol for building an effective cooperative MAS. This protocol is termed the post task-completion (PTC) protocol because it advocates that cooperative groups of agents share their local state values only *after* they have completed a sequence of resource allocation actions to complete a task. While information-sharing has been used previously in cooperative MAS (Tambe, 1997; Xuan et al., 2001), the novelty of our protocol is the "delay" (until task completion) in sharing information between the agents. As will be shown in this chapter, this feature indeed generates more accurate estimates by ensuring a more time-efficient distribution of information compared to a relatively standard mechanism of sharing information between nearest neighbours (section 2.2.1). Furthermore, this protocol forms the basis of state-estimate learning by agents which then allows them to adapt their task processing behaviour in response to environment changes. In particular, this chapter discusses how the PTC protocol is used by the agents to generate such estimates using Q-learning (Watkins and Dayan, 1992). Thus, by designing the PTC protocol and providing theoretical evidence about its effectiveness in generating quality state estimates in sequential RA problems, this chapter sets up a theoretical foundation for establishing the validity of our thesis stated in chapter 1.

In the rest of this chapter, section 3.1 defines the scope of applicability of PTC, that of distributed sequential RA tasks. It also identifies the functional requirements for an information-sharing protocol to be effective in this type of task domain. Subsequently, it explains how these requirements motivate the design of PTC. Section 3.2 describes how the PTC protocol will be designed in the example application of call routing in a wireless mesh telephone network (TN). In addition, this section presents a set of

benchmark algorithms, Q-routing (section 2.2.1) and TPOT-RL (section 2.2.2), used in this application domain against which the PTC-based design will be compared using empirical studies (in chapter 4). Section 3.3 then presents a detailed formal analysis of the advantages of using PTC in generating an effective cooperative MAS. Finally, section 3.4 summarises the contribution of this chapter.

3.1 Application Task Domain and Motivation for Designing PTC

As stated in chapter 1, in our work we focus on designing an effective MAS solution for solving distributed sequential RA tasks. Thus, the design of our PTC information-sharing protocol is motivated by the characteristics of this task domain. In the following, we first define what we consider as sequential RA tasks (section 3.1.1) and then present the PTC information-sharing protocol (section 3.1.2).

3.1.1 Sequential Resource Allocation Tasks

In this research, the way we identify sequential RA tasks is defined in the following.

Definition 1 *A sequential resource allocation task is a sequence of allocation of the appropriate resources such that each allocation contributes towards the partial completion of the task and it is successfully completed once all the necessary resources have been allocated in an order that satisfies its requirements from initiation to completion.*

In more detail, the processing of such tasks is extended in time where the individual allocations by agents are of finite time duration and that requires the availability of the necessary resources to complete part of the task. For the partial completion of a task, an agent allocates resources that it has direct access to. After allocating resource, it forwards the task to another agent for further processing towards achieving completion. In particular, the individual agent decisions (and, thus, the quality of the overall solution) are improved by providing them with estimates of the states of other agents. This is because, such estimates help the agents by making their individual actions contribute towards the successful completion of the task. When an agent delegates the allocation to the subsequent agent, it uses its estimates of the states of the agents in the subsequent sequence of allocations to measure the chance of the successful completion of the task.[1]

[1]If, during a sequence of such allocations and delegation to a subsequent agent, an agent cannot process the task due to non-availability of the necessary resources, the processing fails and the task remains incomplete. Thus, we consider that an uninterrupted sequence of allocations to be required to complete the task. No intermediate restarts are assumed if task processing fails. More on this in chapter 5.

As an example, consider the task to successfully place a call between two nodes. To do so, a set of node bandwidth units should be allocated in order, starting from the source and ending at the destination. Thus, the availability of bandwidth at a set of nodes that can form a continuous circuit from source to destination is necessary. However, at each routing step, a node has to take the decision of forwarding a request for the subsequent allocation to the appropriate node so that the call is placed via an efficient channel. It uses its estimates of the bandwidth availabilities (a measure of node state) of the other nodes in the network to take this decision to ensure an efficient overall allocation.

In this context, PTC contributes by providing good quality estimates of the system states such that efficient allocations are achieved. The following discussion identifies the criteria required for an information-sharing protocol to achieve this objective and defines PTC in this context.

3.1.2 Designing PTC for Sequential Resource Allocation Tasks

To ensure good quality state estimation in dynamic environments by sharing information between agents, the communication strategy should satisfy the following criteria:

- **Time efficient distribution:** There is a latency associated with communication. Hence, the more timely the information that is communicated to an agent, the more likely it is that the information will be up-to-date.
- **Accuracy of information:** In continuously changing environments, it is impossible for all agents to remain synchronised with state changes at all times. Nevertheless, the more accurate the information received, the better.

Given these desiderata, here PTC is proposed as an effective strategy for distributing the local state information of agents. In the following, we define PTC formally.

Definition 2 *Post task-completion information-sharing refers to the distribution of state information between a group of agents, by way of a mechanism that depends on the allowed agent interactions, only after the completion of the sequence of allocation actions by these agents.*

The motivation for using this scheme is to let an agent that participated in completing a task have an indication of the state changes of the other agents in the group that resulted from processing that task. Such information is then used to learn estimates of the states of other agents. These estimates, in turn, are useful for making more informed decisions while processing a subsequent task (as explained in section 3.1.1). In a dynamic system, the world states change while the agents process a given task.

Therefore, by delaying the transmission of information until the task is completed, this protocol ensures that all those agents who participated in the task completion process are informed about these state changes and how that affects the outcome of the task. In so doing, we hypothesise that the agents would be able to distribute information in a time-efficient manner and learn reasonably accurate estimates, thereby satisfying the requirements identified above. In particular, in tasks that require actions of different agents to be ordered sequentially for them to be completed, the PTC model of information distribution allows any agent to maintain highly up-to-date information about the states of the others. The analysis presented in section 3.3 establishes our hypothesis by comparing the timeliness and estimate qualities of PTC against those of a widely used protocol of sharing information between nearest neighbours (hereafter termed as the NN protocol, discussed in section 2.2.1) while processing such tasks.

3.2 Information-Sharing Protocols for a Network Call Routing Application

In this section, we describe the designs of PTC and the benchmark algorithms, Q-routing and TPOT-RL, in the example application of network call routing. First, we describe the sequential RA task performed by agents in this application.

Assuming a TN has a set of agents $\mathcal{A}$, we consider an arbitrary subset of $\mathcal{A}$, $\mathcal{N} = \{a_i \mid i = 1, ..., n\}$ (where $n = \mid \mathcal{N} \mid$), participating in routing a call (the task) at a given time. While doing so, an agent in $\mathcal{N}$ considers one of its neighbours to forward the call (hence, the task processing requires the actions of different agents to be in sequence). This decision is based on the agent's Q-estimates. Specifically, we use a Q-table for each agent a_i where an entry $Q_i(n, k)$ represents the expected utility of choosing neighbour a_k when the call destination is a_n (note the size of the Q-table for each agent is $\mid \mathcal{A} \mid \times \mid \mathcal{K} \mid$, where $\mathcal{K}$ is the set of neighbour agents). In particular, for a TN, we chose the Q-values to represent the *estimated availability of free bandwidth channels* on nodes along the various paths from a_k to a_n. Note, in this representation of the Q-function, n is the *goal state* (the fact that the call has to be routed to a_n) of the agent and the *current state* is i (the fact that the call is currently with a_i). So, effectively, the agent learns to forward a call along the path with the maximum available bandwidth to reach the goal state from the current state. This representation has the advantage that all intermediate state transitions (the sequence of nodes that the call has to be routed through) are collapsed into one effective transition from the current state to the goal state. The Q-value, therefore, signifies the "effective utility" (in terms of the available bandwidth; the higher the value of which the better is the utility in terms of successfully routing a call) of selecting a given neighbour to reach the goal state. This Q-value is learnt from the information distributed by the information-sharing strategies, described shortly. Note,

a similar Q-function has been used previously by researchers studying adaptive routing using Q-learning (see section 2.2.1).

In the above context, an agent (a_i) who wants to route a call to a destination node (a_n) using a set of nodes that have the highest expected bandwidth availability, chooses the neighbour a'_k such that $a'_k = \underset{a_k}{\text{argmax}}\, Q_i(n,k)$, where the maximisation is over all neighbouring agents of a_i. In our empirical study, we use Boltzmann's exploration (Watkins, 1989), a standard scheme for probabilistically choosing a neighbour as opposed to this deterministic strategy (because the deterministic greedy strategy may not allow a sufficient exploration by choosing all possible neighbours and, thus, generate a skewed, sub-optimal performance).

In more detail, an instance of agent a_1's request to its neighbour a_2 to forward a call toward the destination node a_n is shown in figure 3.1(a). Being cooperative, a_2 will accept the request if it has the available capacity. After forwarding the request, a_1 pre-allocates one unit of call channel bandwidth (figure 3.1(b)) for the *partially connected* call until it is either successfully connected or dropped (as described shortly). The forwarding continues (figure 3.1(b)) until the destination node (a_n) is reached. In this manner, therefore, the task gets completed (the call is successfully connected between its source and destination nodes) by the actions of bandwidth allocation and request forwarding by the agents in sequence. At this point, a message is transmitted back along the route through which the call was routed to inform each agent that the call has actually *connected*. Each agent then allocates one unit of call channel bandwidth to complete a circuit from the source to destination (before this, the nodes had only pre-allocated bandwidth) (figure 3.1(c)). Also, using this message, agents transmit their local state values to other agents on the route. Hence, those agents that cooperated on a task (routing the call) share among themselves their local state information after the task is completed (after the sequence of requests reach the destination node). In this way, therefore, the PTC principle (see definition 2) has been instantiated specifically in the TN domain. More details on this follow shortly.

However, the forwarding process stops if an agent is contacted that has no unallocated bandwidth. Then, the agent transmits a message to inform those on the route to drop the partially connected call and deallocate the pre-allocated bandwidth (figure 3.1(d)). In addition, the worst case setup time of a call is bounded by an upper limit for the time that the agents can continue with the forwarding process. After this time, the call is dropped if it has not connected. This time is equivalent to the maximum delay a caller would experience between dialling a number and hearing the ring tone.[2] Finally, if an

[2]In case of the above two conditions, we do not use any backtracking to search for alternative paths. This keeps the routing protocol simple and makes the analysis of the system behaviour easy. Moreover, this simplification should not impact the overall conclusions of this research since inclusion of backtracking would impact all strategies equally.

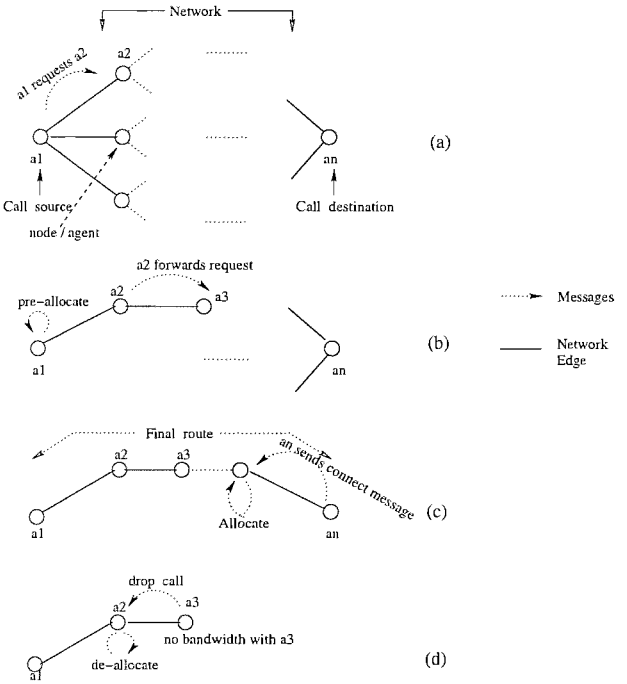


Figure 3.1: The call forwarding process

agent, while forwarding the call, detects a cycle in the route taken by the sequence of requests it generates a message with a penalty and transmits it to the agents on the cycle. This penalty, an exponentially decreasing function with the distance of a node from the loop end, is subsequently used by those agents to update their estimates such that the occurrence of further cycles is reduced. Moreover, while penalising, the agents on the loop de-allocate the previously pre-allocated bandwidth since loops are redundant portions of a call path (for more details, see section 4.2).

In the preceding, we presented a broad description of the sequential bandwidth allocation task performed by the routing agents. In the following, we first explain how the PTC protocol is implemented in this setting. Then, the implementations of the benchmarks in this domain are presented. A more detailed discussion of our simulation of the call routing application is presented in chapter 4.[3]

[3]In all of these information-sharing strategies, communication is used as a *controlled* mechanism for distributing information between agents. In this context, it may be argued that *broadcasting* information between all agents can yield better solutions than any of these strategies. This is because, by broadcasting, all agents could be updated about the global state at all times. However, this is not the case. This is because broadcast is completely impractical in the mesh network application due to the very high bandwidth requirement caused by excessive messaging. Moreover, in large distributed systems, it is the right information about the right, often partial, portions of the global state (rather than indiscriminate communication used in broadcast) that generates the desired performance. Thus, broadcast is not treated as a benchmark in our work. The fact that it is not used as a benchmark by other researchers using communication-based MAS solutions in real applications is also indicative of the rationality of our choice. We report a brief evaluation of broadcast in appendix A. The observations of this study confirm the above arguments.

3.2.1 Post Task-Completion Information-Sharing

Consider (as described above) a call is routed from a_1 (source) to a_n (destination) along the path $a_1...a_n$. In the instantiation of the PTC principle, agent a_n starts communicating its own local state (in a TN a node's state is represented by the available bandwidth units on that node) when the bandwidth allocation process completes at time t at a_n, thereby, establishing a complete circuit from source to destination. Agent a_n communicates this information to a_{n-1}. Thus, a_{n-1} updates its prior estimate of a_n's available bandwidth units $Q_{n-1}(n,n)$ with the new state information using the standard Q-update rule (Watkins and Dayan, 1992): $Q_{n-1}(n,n) \leftarrow (1-\alpha)Q_{n-1}(n,n) + \alpha\, s(n,t)$, where $s(n,t)$ represents the local state of a_n at time t and is the "reward" for the Q-learner to update its prior Q-estimate. Subsequently, a_{n-1} communicates to a_{n-2} its own local state at time t' $s(n-1,t')$ ($t' \neq t$, because of the latency in communication between neighbour agents) and the information it had received from a_n. Alternatively, it can use its own state information and that received from a_n to communicate a summary information that captures the overall state of the path being used to route the call. Agent a_{n-2} similarly updates its prior estimate of the bandwidth availability $Q_{n-2}(n,n-1)$ using the information received from a_{n-1}. This procedure of distributing their own and the previously received state information continues until the source agent (here, a_1) is reached. Typically, the information about the states of multiple nodes is used to generate a summary estimate of the state of the downstream route as described in section 4.3.[4] The distinction between local state (an agent's own state) and non-local state (another agent's state) is lost in aggregating information of multiple nodes to maintain a summary of call routes. However, this is not a problem in a TN since information about the path bandwidth availability is sufficient for an agent to take effective routing decisions (selecting a subsequent node to forward a call request). On the other hand, maintaining the information of individual nodes would necessitate each agent solving a computationally expensive least-cost-path problem before every routing decision. However, in a different domain, it is entirely possible for an agent to maintain separate state estimates of other agents in the cooperative group while using the same PTC principle.

In this instantiation of PTC, only those nodes who participated in routing a call share information. So, the state information of the other nodes in the system is not distributed. But, the decision of which agents should be informed about which system states is a separate problem. In this thesis, we advocate PTC as a specification for distributing information among those agents who cooperate on a task after its completion. Therefore, distributing information among the routing nodes follows this specification. In future, we plan to investigate the problem of how we can determine which agents within a cooperative group should be notified of a certain piece of state information after task

[4]Such summarisation of state information is separate from the basic PTC principle which simply states that information is shared between a cooperating group after task completion.

completion and the consequence of such selective distribution. Chapter 6 presents one approach of using such selective information distribution in the context of the sequential RA tasks. Note that in the above description, the communicated state information acts as the "reward" for Q-learning. Therefore, the accuracy and the timeliness of this information is critical in determining the quality of the Q-estimates. This, in turn, directly impacts the effectiveness of an agent's decision to route calls.

3.2.2 Q-routing

The Q-routing algorithm is based on the NN protocol (as discussed in section 2.2.1). In Q-routing (hereafter, referred to as QR), an agent a_i, after forwarding a call to neighbour a_{i+1}, receives the latter's current best estimate of the bandwidth availability to reach destination a_n. Thus, neighbour a_{i+1} informs a_i with $\hat{Q}_{i+1} = \min(s_{i+1}, \max_{a_{i+2}} Q_{i+1}(n, i+2))$, where the maximisation is done over all neighbours a_{i+2} of a_{i+1}. Since, on a given path in a TN, the node with the minimum bandwidth availability determines the maximum number of calls that can be placed via that path, a_{i+1} determines the minimum of its own bandwidth availability (its "state", denoted by s_{i+1}) and its estimate of the subsequent path. In this way, in QR, information is shared between neighbour agents only *while* the task is being processed as opposed to PTC in which the sharing occurs after task completion. Now, agent a_i, upon receiving this estimate, updates its prior estimate $Q_i(n, i+1)$ as $Q_i(n, i+1) \leftarrow (1-\alpha)Q_i(n, i+1) + \alpha\,\hat{Q}_{i+1}$. This process of asking neighbours and receiving the latter's estimates continues until the destination is reached (see figure 3.1(c)) or the forwarding process is terminated (see figure 3.1(d)). The way the Q-estimates are updated in QR is similar to the update rule used in PTC as shown in section 3.2.1. The difference, however, is in the reward: whereas in QR, the reward is the estimate of the immediate neighbour, in PTC, it is a summary of the actual state information of all subsequent path agents.

3.2.3 TPOT-RL

TPOT-RL is implemented in our TN domain following the description of (Stone, 2000). The characteristic features of TPOT-RL (see (Stone, 2000) for more details) are implemented as follows: (i) a partitioning function identical to (Stone, 2000); (ii) an action-dependent feature function also identical to (Stone, 2000), where the `activity-window` parameter is chosen as 100, and the usage-threshold for a link connecting to a neighbour is set to 5.0 (half the maximum bandwidth capacity of a node, defined in section 4.5); and (iii) the reward `update-interval` is set to 100. Section 4.2 explains in more detail how these parameters are used by the agents to learn estimates and route calls. While forwarding a call, a node using TPOT-RL transmits to the subsequent node its current estimate about the bandwidth availability to reach the given call destination. Thus,

TPOT-RL uses an information-distribution mechanism similar to that of QR: between neighbour agents while processing tasks. Now, each agent also records the amount of bandwidth usage on each of the links connecting to its neighbours. Its Q-values estimate the bandwidth availability to reach a given destination via a given neighbour for the given link usage level, monitored over the past `activity-window` time steps. A call-forwarding decision is taken by using a Boltzmann exploration over the Q-estimates. Reward distribution occurs in TPOT-RL every `update-interval` time steps. More specifically, for all calls that are successfully connected, the corresponding destination nodes accumulate the information that was transmitted by the forwarding nodes along the call paths. Then, after every `update-interval` time steps, these destination nodes start sending the accumulated information back along the corresponding call paths. An agent located along such a call route, updates its Q-values after receiving this information. So, a node a_k along the path $a_1, \cdots, a_k, \cdots, a_n$, gets the *estimates* (as opposed to the actual node states) of its subsequent nodes $a_{k+1}, \cdots, a_n$, using which it updates its Q-value. Note that we have used aggregation of the information received from subsequent agents similar to that in PTC (section 3.2.1).

3.3 Advantages of the PTC Protocol

In this section, we present a formal analysis to explain the advantage of our information-sharing model in generating better learning than the nearest-neighbour-sharing protocol. While this establishes the benefits of PTC on a theoretical ground, it also provides an explanation for the performance improvements observed in our empirical studies. In the following, we first state our assumptions and notations that will be used in later discussions. The rest of this section develops a formal representation of the timeliness of distributing information by both strategies. This representation is then used to compare the accuracies of the non-local state information in each case.

3.3.1 Basic Assumptions and Notations

In section 3.1, it was stated that we consider task episodes that require the sequential participation of the appropriate agents for successful completion. In this context, our analysis focuses on a particular set of agents $\mathcal{N} = \{a_1, ..., a_n\}$ where agent a_1 initiates the task execution process by receiving a new task. Also, we assume, without loss of generality, that the order in which the agents process tasks is: $a_1 \rightarrow \cdots \rightarrow a_n$. This assumption implies that, in this task processing instance, a_1 uses its knowledge of the states of other agents and selects a_2 to forward the task, a_2 similarly selects a_3, and so on until a_{n-1} selects a_n which is the agent at which the task processing is completed (this was described in section 3.2 in the context of a TN). Note that $\mathcal{N}$ represents

one possible set of agents that can complete the task (equivalently, in a network, there can be multiple routes through which a call can be routed to the destination). The state estimates that the agents use to select the subsequent agent are generated using communicated information of the unobserved states via the particular communication protocol used by the agents (PTC or NN). Thus, focusing on one particular set (in this case, $\mathcal{N}$) simplifies the analysis of how a communication strategy affects the accuracy of a given agent's knowledge of the agents in that set. Since $\mathcal{N}$ is arbitrarily selected, it is equivalent to selecting any other set of agents. Therefore, the results of our analysis do not depend on which $\mathcal{N}$ is chosen.

Further, we consider that the agents process tasks that are generated continuously. To this end, let the symbol t_c denote the fixed time after which successive tasks are generated. Such an assumption of periodicity in the task environment is made to simplify our analysis. Subsequently, in section 3.3.6, we show that even under more general, non-periodic environments, the same general conclusions hold.

In the sequence of agents that jointly participate in executing the task, there is the notion of a "subsequent" agent (and, for that matter, a "preceding" agent) for any agent except the last (the first). We represent the agent subsequent to an agent a_i by the identifier a_{i+1} and the one preceding a_i by a_{i-1}.

The agent state, as per section 3.2, is represented by a *real-valued* function s. For example, in a network, this can represent the load level, or, equivalently, the fraction of the total bandwidth used on a node. Also as discussed in section 3.2, an agent learns these agent states, using the communicated information from other agents, to decide which subsequent agent to choose. The *actual* state of agent a_i (as observed by a_i itself) at time t is represented by $s(i,t)$. Agent a_i's knowledge of a_j's state at time t is $s'(i,j,t)$ $(i \neq j)$. The knowledge that a_i has of the agents in $\mathcal{N}$ at time t is represented by:

$$\mathcal{S}_i^t = \{s'(i,j,t) \mid j = 1,...,n\}. \tag{3.1}$$

The corresponding set of actual state values of the agents in $\mathcal{N}$ is represented by:

$$\mathcal{S}^t = \{s(j,t) \mid j = 1,...,n\}. \tag{3.2}$$

Note that in dynamic systems these states change with time. For example, the load level of a communication node varies with time. Thus, without timely updates, the known values can be different from the actual state values.

As noted earlier, agent a_i uses $\mathcal{S}_i^t$ to select the subsequent agents to whom it forwards a task. This decision, in turn, affects the overall utility earned from processing tasks in the system. The exact function used by an agent to determine the subsequent agent depends on the task and the domain characteristics. In a TN, for example, an agent

can select a subsequent agent for which it estimates that the average load on all nodes from that agent to the destination node is minimised. Thus, this decision has the effect of using the least congested path every time a call has to be set up which, in turn, maximises the number of calls routed in the system. Our analysis does not depend on the exact form of the decision function. Rather, it studies the delay between consecutive reinforcements of state information by a given information-sharing protocol that generate the knowledge $s'(i,j,t)$. The $s'(i,j,t)$ values act as the parameters in an agent's decision function. Hence, it can be concluded that the closer these values are to the true states (which, as stated earlier, change with time), the higher is the accuracy of the agent's decision. Intuitively, the shorter the delay in sharing information, the more up-to-date is the information maintained. Thus, the more effective an agent's decision. In the following subsections, we analyse the timeliness of information-sharing by the different communication strategies.

3.3.2 Timeliness of Information Distribution in NN

In this section, we compute the delay incurred by an agent to get the state information from the others in $\mathcal{N}$ using the NN protocol. For this, we focus on a_1's knowledge about the states of $\{a_1, ..., a_n\}$ at time t. At this time, a_1 has the knowledge of its actual state $s(1,t)$. However, its knowledge of the other agents $s'(1,j,t)$ $(j = 2, ..., n)$ are different from the corresponding true states by an amount equal to $\mid s(j,t) - s'(1,j,t) \mid$. Note that the information that an agent maintains at a given time is the result of the previous communication that occurred between the agents (refer to section 3.2.2 for a discussion on how a particular implementation (QR) of the NN protocol works).

Since tasks originate every t_c time steps, an agent a_j $(j = 1, ..., n-1)$ requests its subsequent agent a_{j+1} for the latter's knowledge every t_c time steps. Following a request at any time t, a_j receives the response from a_{j+1} after a delay of $2\Delta t$, assuming the request arrives at a_{j+1} after a delay of Δt, and the response of a_{j+1} comes back to a_j after a further delay of Δt, at $t + 2\Delta t$. Note here Δt refers to the communication delay of a message between directly communicating agents. Referring to the description in section 3.2.2, we note that a_{j+1} provides the information that it has of the set of agents $\{a_{j+1}, ..., a_n\}$.[5] However, a_{j+1}'s knowledge is based on the information it received from a_{j+2} on its previous request to a_{j+2}. The previous request of a_{j+1} to a_{j+2} was during the processing of the previous task at $t + \Delta t - t_c$ for which it had received a response at $t+\Delta t-t_c+2\Delta t$ (i.e., after a delay of $((t_c-\Delta t)-2\Delta t)$). In a similar way, that response of a_{j+2} to a_{j+1} contained information that a_{j+2} received from its previous request to a_{j+3}.

[5]Note that in section 3.2.2, we discussed QR, where a *summary* of the states of the subsequent agents is communicated. In this formalisation, we consider an agent maintains separate records of the states of other agents. Such a consideration helps explain the impact of a given communication protocol on the accuracy of an agent's knowledge.

Table 3.1: Time diagram for nearest-neighbour sharing

Agent	← Time			
a_1	$(t+2\Delta t)$	$(t+2\Delta t-t_c)$	$\cdots$	$(t+2\Delta t-(n-2)t_c)$
a_2		$(t+\Delta t+2\Delta t-t_c)$	$\cdots$	$(t+\Delta t+2\Delta t-(n-2)t_c)$
$\vdots$				$\vdots$
a_{n-1}				$(t+(n-2)\Delta t+2\Delta t-(n-2)t_c)$

That request of a_{j+2} to a_{j+3} was at $t+2\Delta t-2t_c$ for which it had received a response at $t+2\Delta t-2t_c+2\Delta t$ (i.e., after a delay of $(2(t_c-\Delta t)-2\Delta t)$). Extending this procedure to all subsequent agents, therefore, at time $t+2\Delta t$, the information that a_j has of any other subsequent agent a_k is the state of a_k delayed by an amount $(d-1)(t_c-\Delta t)-2\Delta t+\Delta t$, where $d=k-j$. Note here, an extra Δt is added to the delay because although the response was received at $t+(d-1)\Delta t-(d-1)t_c+2\Delta t$, but this contained information about a_k at time $t+(d-1)\Delta t-(d-1)t_c+2\Delta t-\Delta t$.

The above description is summarised in table 3.1. In this table, the rows represent agents (with the agent numbers increasing from top to bottom) and the columns represent time (with time farther in the past as we move from left to right). More specifically, in table 3.1, each element represents a time when an agent (represented by the row number) received the information from its subsequent agent. In particular, it focuses on agent a_1 and assumes that it has requested a_2 for the latter's knowledge at time t. Therefore, a_1 receives information from a_2 at $t+2\Delta t$ (row 1, column 2 in table 3.1). However, the knowledge about the subsequent agents that a_2 provides a_1 is based on the requests that these agents made at times further delayed in the past. These are the first element of each row. Hence, at time $t+2\Delta t$, the set of state information of the agents $\{a_1,...,a_n\}$ that a_1 has is the following:

$$\begin{aligned}\mathcal{S}_1^{t+2\Delta t} &= \{s(1,t+2\Delta t)\}\cup \\ &\quad \{s(i,t+(i-2)\Delta t+2\Delta t-(i-2)t_c-\Delta t)\mid i=2,...,n\}. \qquad (3.3)\end{aligned}$$

Note that, an additional Δt is subtracted in the u values of all subsequent agents in equation (3.3). This is because, while table 3.1 shows the last time an agent *received* information from its subsequent agent, this information is, in fact, delayed by an amount of Δt; hence, the Δt is subtracted. For clarity, $t+2\Delta t$ in equation (3.3) is replaced by t'. Thus,

$$\mathcal{S}_1^{t'} = \{s(1,t')\} \cup \{s(i,t' + (i-2)\Delta t - (i-2)t_c - \Delta t) \mid i = 2,...,n\}. \tag{3.4}$$

The true states of these agents, at time t', are

$$\mathcal{S}^{t'} = \{s(i,t') \mid i = 1,...,n\}. \tag{3.5}$$

Thus, equation (3.4) shows that the knowledge that agent a_1 has of any other agent in $\mathcal{N}$ is delayed by an amount that depends on the distance (number of hops) between them. More specifically, the knowledge that an agent, say a_i, has of another agent, say a_j, that is k hops away is delayed by an amount:

$$t_{\texttt{delay}}^{\texttt{NN}} = (k-1)(t_c - \Delta t) + \Delta t. \tag{3.6}$$

This measure of delay incurred in NN will be compared to the same in PTC.

3.3.3 Timeliness of Information Distribution in PTC

In this section, we compute the delay incurred by an agent to get the state information from the others in $\mathcal{N}$ using the PTC protocol. Similar to the analysis in section 3.3.2, we assume that a task originates every t_c time steps when a_1 initiates the processing of the task and forwards the request to the remaining agents $\{a_2,...,a_n\}$. In the PTC sharing protocol, the state information of agents is communicated only after a task is completed. Since new tasks are processed every t_c time steps, it can be inferred that the distribution of state information by the agents occur every t_c time steps (i.e., after every task completion phase and assuming that the communication delay between any two directly communicating nodes remains the same).

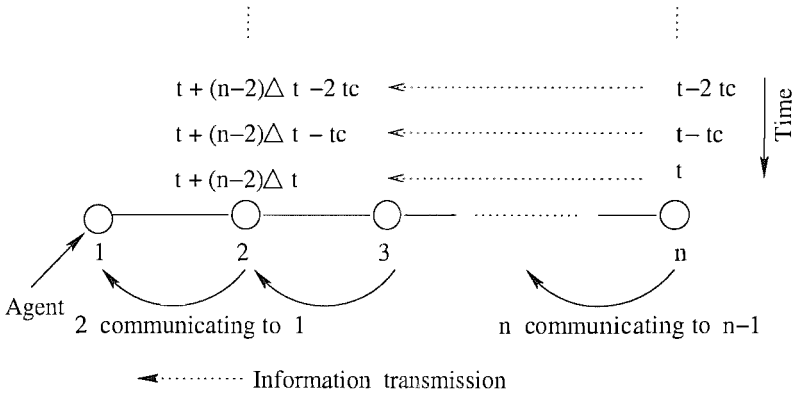


Figure 3.2: Time diagram for PTC sharing

Therefore, a_n transmits its state information to a_{n-1} at time t (i.e., when a task completes at time t, for any value of t), and then at $t+t_c$, $t+2t_c$, and so on. Given this, a_{n-1} transmits its own state and the information received from a_n to a_{n-2} at $t+\Delta t$, and then at $t+\Delta t+t_c$, $t+\Delta t+2t_c$, and so on (considering the delay of Δt for the information to reach a_{n-1} from a_n). Extending this process, it can be inferred that a_2 transmits its own state and its knowledge of the set of agents $\{a_3, ..., a_n\}$ to a_1 at $t+(n-2)\Delta t$, and then at $t+(n-2)\Delta t+t_c$, $t+(n-2)\Delta t+2t_c$, and so on. Figure 3.2 shows this process. In this figure, each agent is labelled with the time at which it transmits its state information to its previous agent. Thus the state information that agent a_1 receives from its subsequent agent (in this case, a_2) contains the information of the rest of $\mathcal{N}$ delayed by multiples of Δt. Thus, a_1 has the following information about the subsequent agent states (assuming it received information from a_2 at time t'):

$$\mathcal{S}_1^{t'} = \{s(i, t' - (i-1)\Delta t) \mid i = 1, ..., n\}. \tag{3.7}$$

The true states of these agents at this time t', shown in (3.5), however, are different from these values.

Thus, (3.7) shows that the information that agent a_1 has of any other agent in $\mathcal{N}$ is delayed by an amount that depends on the distance (number of hops) between them. More specifically, the information that an agent, say a_i, has of another agent, say a_j, that is k hops away is delayed by an amount:

$$t_{\texttt{delay}}^{\texttt{PTC}} = k\Delta t. \tag{3.8}$$

In the following section, we use the delay measures computed in (3.8) and (3.6) to establish the advantage of PTC compared to NN.

3.3.4 Comparing Timeliness of Information Distribution in PTC and NN

The analysis of section 3.3.3 shows that using the PTC protocol, an agent, say i, after the completion of a task episode, receives the local state information of another agent j after a delay of $k\Delta t$ (see formula (3.8)), where k is the hop count between i and j. In the NN protocol, on the other hand, agent i receives j's state information after a delay of $(k-1)(t_c - \Delta t) + \Delta t$ (see formula (3.6)). Comparing the delays, the following can be concluded.

Proposition 1 *The delay for non-local information to reach an agent is less using PTC than using NN if the task environment periodicity is greater than the round trip communication delay of a message between directly communicating agents.*

This is because, for $k > 1$,

$$t_c > 2\Delta t \Rightarrow k\Delta t < (k-1)(t_c - \Delta t) + \Delta t. \tag{3.9}$$

In a typical MAS, the interval between successive episodes of task execution (t_c) is much longer than the communication latency between two directly communicating agents (Δt).[6] Hence, the delay due to PTC is, for all practical purposes, much less than that of NN.

Having established that PTC distributes information in a more time-efficient manner than NN, we now focus on analysing how this characteristic of PTC creates more up-to-date information.

3.3.5 Improved Estimation Accuracy using PTC

In this section, the improved time-efficient information distribution of PTC is mapped to the improved quality of information learnt by PTC over NN. The key idea is that, the shorter the delay between successive information messages, the more accurate is the knowledge of the actual states.

As stated before, node states vary dynamically over time. However, at a given time, the state of a node can have a certain value from a certain finite set of values, say $\mathcal{V}$. Also, a node n_i retains its state s_m[7] (where $s_m \in \mathcal{V}$, $m \in \{1, \cdots, M\}$, and $M = \mid \mathcal{V} \mid$), for a certain length of time, say l_m. We consider that a certain node n_i is dynamically estimating the states of another node n_j that is at a distance of k hops from n_i. Given the above information, we want to compute the expected value for a given number of state changes that can occur in n_j in a given time duration, say t_D. We hypothesise that if t_D increases, so does the expected value for any number of state changes within t_D. Therefore, if n_i receives information from n_j with higher delays then it loses more state-change information of the latter. Since NN has a greater delay than PTC (section 3.3.4), it incurs a higher loss of state-change information than PTC. In this context, we define the following:

Definition 3 *Given a finite set of states* $\mathcal{V} = \{s_m \mid m = 1, ..., M\}$ $(M = \mid \mathcal{V} \mid)$, *where each state value* s_m *lasts for a time-length of* l_m, *and a time duration* t_D, *where* $t_D < \sum l_m$, *a* **coverage of size** h **from** $\mathcal{V}$ **on** t_D, *represented by* $c(h, \mathcal{V}, t_D)$, *is a set of* h

[6]For example, in the type of communication networks we are studying in this research, the typical delay between successive calls is of the order of minutes, whereas the communication latency between adjacent nodes is of the order of milliseconds.

[7]The representation $s(i,t)$ used earlier in this section to identify the state value of agent a_i at time t is replaced with s_m. In the current discussion, since we are considering one agent and the different state values that it can take, the identifier i and time t are dropped for an easier notation. Nevertheless, $\forall i, \forall t, \exists m, s(i,t) = s_m$.

different states of $\mathcal{V}$ $(h \leq M)$ *such that* $\sum_h l_i \leq t_D$.

Using the above, the expected value of h different state changes of n_j within a time-interval t_D is given by:

$$\frac{\text{number of possible } c(h+1, \mathcal{V}, t_D)}{\binom{M}{h+1}}, \tag{3.10}$$

where the numerator counts all possible coverages of size $h+1$ (hence, having h different state *changes*) from $\mathcal{V}$ on t_D. The denominator enumerates all possible ways of choosing $h+1$ different states from $\mathcal{V}$.

Now, if we consider a duration $t'_D > t_D$ ($t'_D < \sum l_m$),[8] then it is trivial to identify that every $c(h, \mathcal{V}, t_D)$ will also be a $c(h, \mathcal{V}, t'_D)$, for all h. This is because, all combinations of l_i's that "fit" within t_D would necessarily fit within t'_D. Therefore, it can be said that $\lfloor c(h, \mathcal{V}, t'_D) \rfloor = c(h, \mathcal{V}, t_D)$. Hence, the numerator of formula (3.10) with t_D replaced by t'_D would at least be equal to that for t_D. The above reasoning brings us to the conclusion that the expected value for observing K different state changes (for any K) increases with increasing delay between successive observations. Thus, more state-change information is lost as the delay between observations increases. Since, according to section 3.3.4, PTC achieves a lower delay than NN between successive observations, the following can be concluded.

Proposition 2 *PTC incurs a lower loss of state-change information than NN.*

The analysis presented so far assumes a periodic environment where the task episodes repeat after intervals of constant length. In the following, we present a similar analysis with the periodic assumption removed and demonstrate that the same conclusions hold.

3.3.6 Non-periodic Task Environment

The analyses presented in sections 3.3.2 and 3.3.3 are based on the assumption that task completion episodes repeat every t_c time steps, with a constant t_c. Therefore, the formulae (3.6) and (3.8) were derived using only one of these episodes. In a more general setting, however, the task processing episodes would be non-periodic, with the time between successive task completion episodes varying. In that case, these formulae have to be computed considering the successive episodes as opposed to only one. In this context, note that the information dissemination delay of NN alone (formula (3.6)) depends on the value of t_c. Therefore, the assumption of non-periodic episodes impacts

[8] If $t_D = \sum l_m$, then the expected value of observing h different state-changes is equal to 1, for all $h \leq M-1$. If $t_D > \sum l_m$, then we can apply the same reasoning as above for the modified duration t''_D, where $t_D \equiv t''_D (\text{mod} \sum l_m)$ to reach the same conclusions.

the delay terms of only NN. The following discussion indicates how to account for the non-periodicity.

Considering the case described before in section 3.3.2 where agent a_i maintains the state of a_j which is k hops away. In a non-periodic situation, formula (3.6) changes to,

$$t^{\mathtt{NN}}_{\mathtt{delay}} = \sum_{m=1}^{k-1} t_c^{j-m} - (k-2)\Delta t, \tag{3.11}$$

where, t_c^{j-1} (for any j) represents the most recent episode, t_c^{j-2} the second most recent episode, and so on.

Using a method similar to that discussed in section 3.3.4, $t^{\mathtt{PTC}}_{\mathtt{delay}}$ given by formula (3.8) can be compared to $t^{\mathtt{NN}}_{\mathtt{delay}}$ given by formula (3.11). Therefore, we can conclude $t^{\mathtt{PTC}}_{\mathtt{delay}} < t^{\mathtt{NN}}_{\mathtt{delay}}$ if:

$$2(k-2)\Delta t < \sum_{m=1}^{k-2} t_c^{j-m}. \tag{3.12}$$

The following summarises this observation.

Proposition 3 *In a non-periodic task environment, the information dissemination delay is less for PTC than for NN if the time between any two successive task originations is greater than the round trip communication delay of a message between directly communicating agents.*

This is because (from condition (3.12)), for $m \geq 1$,

$$t_c^{j-m} > 2\Delta t \Rightarrow t^{\mathtt{PTC}}_{\mathtt{delay}} < t^{\mathtt{NN}}_{\mathtt{delay}}. \tag{3.13}$$

Proposition 3 is similar to proposition 1. It is true for all practical purposes because the time interval between successive task processing episodes is typically much greater than the round-trip communication delay of a message between directly communicating nodes.

Since the delay between successive information received is smaller in PTC than in NN, it can be shown similar to section 3.3.5, that under the non-periodic task assumption, the knowledge about the non-local states generated by PTC captures the changes in these states better than that by NN.

Proposition 4 *In a non-periodic task environment, PTC incurs a lower loss of state-change information than NN.*

The preceding analysis demonstrates that, under all practical purposes, the timeliness

of information distribution and the quality of non-local state information learnt by our PTC information-sharing protocol are better than the nearest-neighbour protocol under general non-periodic environments. With these theoretical results, it is reasonable to infer that PTC allows the agents to take better informed decisions than NN, which, in turn, generates better system performance. To demonstrate this further, the practical advantage of PTC is evaluated using empirical analysis in a simulated wireless telephone network. The following chapter describes this analysis.

3.4 Summary

This chapter develops a novel information-sharing protocol that can be used to generate good quality state estimates for agents so that an effective MAS solution can be built for sequential RA tasks in distributed systems. The novelty of this protocol is that cooperative groups of agents share their local state values by delaying it until a task is completed. The advantages that this protocol generates (compared to the widely used benchmark algorithm of sharing information between nearest neighbours) are a time-efficient information dissemination and, hence, highly up-to-date state estimates. Thus, the above-mentioned contributions of this chapter address the research challenges identified in chapter 1 and, hence, theoretically establish the validity of our thesis.

Now we set our goal to design communication heuristics based on the PTC protocol that can be deployed in a real application: the network call routing problem. In particular, we aim to use empirical analysis to compare the performances achieved by these heuristics in this domain against those of the benchmark algorithms QR and TPOT-RL. In this manner, we would be able to empirically confirm the validity of our thesis. This is the focus of the following chapter.

Chapter 4

Empirical Evaluation of PTC in Telephone Network Routing

In chapter 3, we have developed a novel protocol for cooperatively sharing information between agents. It has been shown, using a theoretical analysis, that our protocol is capable of generating good quality state estimates in a distributed system which, in turn, can improve the performance of RA in sequential tasks. In this chapter, we attempt to substantiate these theoretical propositions using empirical analysis. To do so, we deploy PTC in the exemplar telephone-network call routing problem. Note that this domain represents a distributed system where task completion (to place a call by allocating node bandwidth in a circuit-switched network) requires sequential RA; hence, these empirical studies are targeted towards justifying our thesis. More specifically, a set of communication heuristics are developed on the basis of the PTC protocol using which the network nodes would be able to share information and accomplish the routing tasks. The effectiveness with which the agents achieve routing by using the PTC-based heuristics is evaluated empirically and compared against the performances of the benchmark algorithms. Such empirical validation of the theoretical claims of chapter 3 would help to establish our thesis further.

In the rest of this chapter, we first enumerate a number of important physical properties and functional characteristics of the TN application (section 4.1). These properties are simulated in our system to appropriately capture their effects on its performance. Subsequently, we describe our implementation of a cooperative RA system for routing calls in a circuit-switched network (section 4.2). In particular, it elaborates the implementations of PTC and the QR and TPOT-RL strategies in the simulation. Then, we present two heuristics based on the PTC protocol for aggregating state information in a way specified in section 3.2.1 (section 4.3). Section 4.4 then outlines the experimental setup used for our empirical studies of the various communication strategies. In particular, it identifies a set of performance measures used to evaluate the commu-

nication strategies and elaborates on the physical properties of the simulated network. Subsequently, section 4.5 presents and analyses in detail the experimental results obtained from using the communication strategies in a call routing problem in the simulated networks. These results indicate a superior performance of PTC over those of the benchmarks across all environmental settings. Finally, section 4.6 summarises the contributions of this chapter.

4.1 Domain Properties

We assume the following characteristic properties of our simulation of a TN. These properties are typical of the broad class of wireless mesh networks (Krag and Buettrich, 2004) where a set of wireless nodes with limited communication bandwidth radio-communicate with those within their transmission range. Note that these properties correspond to the more general description of the TN domain presented in section 3.1. These features are representative of both the physical properties of the network and of its behavioural constraints that we aim to model. Thus, the following enumeration is segregated into *physical properties* and *model properties.*

- *Physical Properties:*
  - The communication nodes have limited bandwidth. Therefore, the number of calls that can be handled by a node is limited.
  - A node can only communicate with the nodes that are within its transmission range (its immediate neighbours).
- *Model Properties:*
  - Calls can originate/terminate at any node. These calls originate throughout the simulation and last for a finite duration (thus, indicating a continuous usage of the network).
  - A node's total available bandwidth is divided into two segments: the *call channel* and the *control channel.* The former is used to route calls and the latter to communicate information and control messages.
  - The resource available at a node is its call-channel bandwidth. One unit of this is allocated for each call routed via the node. The bandwidth units of the nodes on a call path are occupied throughout the duration of a call to establish a circuit. Thus, we consider a circuit-switched network where bandwidth is allocated end to end to establish calls.
  - Each node is modelled as an agent. Every agent has the aim of forwarding a call to the neighbour it believes is the first node on a path to the destination with the maximum call-channel bandwidth availability.

– We define the *state* of a node at a given time as the ratio of the call channel bandwidth units it has unallocated to the maximum number of units that it can handle. Each agent has *perfect knowledge* of the state of the node it represents and *estimates* of the states of other agents.

4.2 Cooperative Bandwidth Allocation for Call Routing

In section 3.2, we described broadly how the routing agents allocate bandwidth in sequence to connect a call between the source and destination nodes. Here, a more detailed description of our implementation of this system is presented. To this end, we identify from the discussion of section 3.2 that the different actions of agents are in response to four types of information message:

1. Request to forward a call (m_r),
2. Request to connect a call (m_c),
3. Request to drop a call (m_d), and
4. Request to penalise loop agents (m_p).

In the following, these activities are elaborated. To facilitate this discussion, we first define the following set of variables used in our description: $\mathcal{N}$, set of total agents in the network $\{a_1, ..., a_N\}$, where $N = |\mathcal{N}|$; $\mathcal{K}_i$, set of K_i neighbours of agent a_i ($K_i = |\mathcal{K}_i|$); a Q-function Q_i for each agent a_i, where $Q_i : \mathcal{N} \times \mathcal{K}_i \rightarrow [0, 1]$. The Q-function $Q_i(n, d)$ is a_i's estimate of the bandwidth availability in the nodes on all paths to a_d ($\in \mathcal{N}$) via its neighbour a_n ($\in \mathcal{K}_i$); $s(i)$: actual node *state* of a_i; $\mathcal{B}_{j,k,...,z}$: a set of node *states* $\{s(j), s(k), ..., s(z)\}$; $E_i(d)$: feedback information provided by a_i about its estimate of call channel bandwidth units available over all routes to a_d from a_i; $E_{\mathcal{K}_i}(d)$, a set of $E_j(d)$ feedback estimates from all a_j in $\mathcal{K}_i$; R_i, a *reward* computed by a_i from node state values that it receives via communication from other nodes. In TPOT-RL, however, there is a difference in the representation of the Q-estimates from that described above. This is because, in TPOT-RL, an agent uses an action-dependent feature function, which summarises the local effects of its actions, to estimate bandwidth availability. This function is based on the bandwidth-usage level on the links connecting a_i to its neighbours. Thus, if the variable $l_{i,n}$ denotes the portion of a_i's total call-channel bandwidth being used for calls that are routed via its neighbour a_n, then a_i's Q-estimate for bandwidth availability to reach a destination a_d via a_n is: $Q_i(e_{i,n}, n, d)$. In this, $e_{i,n}$ is "high" if the value of $l_{i,n}$, measured over a certain `activity-window` time interval in the past, is more than a certain threshold, or "low" otherwise (section 3.2.3 specifies the values used for these parameters in our experiments).

A message of type m_r, m_c, or m_d contains the following information: ids of the call source (a_s) and destination (a_d), the set of ids of the nodes through which it has been routed (*path*), the time of origination (t_o, on the global clock) and setup (t_{live}, an absolute interval) of the call, and (depending on the information-sharing protocol used) a set of node state values $\mathcal{B}_{k,\ldots,z}$. The current global time is represented by t. An m_p-type message contains, instead of the call source id, the id of the node where the loop starts. Also, on a given route, $a_{i+1}(m_T)$ (T can be r, c, d, or p) returns the agent id at a position one hop closer to the call destination $a_d(m_T)$ than the current agent a_i, while $a_{i-1}(m_T)$ returns the id one hop closer to the call source $a_s(m_T)$. However, $a_s(m_p)$ denotes the source of the loop and not the call source node.

With this in place, figure 4.1 shows the agent activities. Upon receiving an m_r (line 1), an agent (say, a_i) checks whether it has no unallocated bandwidth or the call forwarding process has lasted beyond the maximum setup time limit (line 2). In either case, the forwarding process is stopped and a_i transmits an m_d (line 5), generated from m_r (line 4) to refer the appropriate call represented by m_r, to the previous agent. Upon receiving m_d (line 33), an agent frees up the pre-allocated call channel bandwidth (line 34) and sends the same m_d to the previous agent (line 36) until $a_s(m_d)$ is reached.

If neither of the conditions in line 3 are satisfied, a_i first checks if a loop has occurred (by checking if $path(m)$ includes its own id). If it has (line 7), a_i generates an m_p-type message (line 8) and computes a penalty $p = (-1)0.9^{x+1}$ (line 10), where x is the hop count from a_i to the end of the loop (i.e., where the loop was first detected). This penalty amount is added to m_p (line 11). Also, the source node id in m_p is set to be the id of the node where the loop was detected (line 12). Then this m_p message is transmitted to the previous agent on the loop (line 13). Upon receiving an m_p-type message (line 50), a_i uses the penalty in m_p to update its prior Q-estimate of the destination node; depending on whether the algorithm used is TPOT-RL or not, one of the updates (line 52 or 54, respectively) gets executed. Subsequently, if a_i is not the agent where the loop was detected (line 55), it computes a new penalty which is an exponentially decreasing function of the distance of that node from the loop end, adds this to m_p, and de-allocates the pre-allocated bandwidth for this call (line 57). Finally, it transmits the m_p to the previous agent (line 58). The intuition here is that the further a node is from the loop end, the less it is responsible for causing the looping to occur. Hence, the lower the penalty it gets.[1]

After checking for cycles, a_i then checks if it is the destination of the current call (line 14). If it is, it will allocate one bandwidth unit (line 16) and send an m_c (generated from m_r for reasons cited before) to the previous agent on the route of this call (line 21)

[1]In our experiments, we have observed that this heuristic substantially reduces the number and size of loops. Hence, it effectively reduces wasteful use of resources, since loops represent redundant portions of a call path.

```
//for a message m in control channel C_i of agent a_i
1. if(m_r) // CALL FORWARD MESSAGE
2.       if (s(i) == 0||t > (t_o(m_r) + t_live(m_r))) // no available bandwidth
3.                                                   // or, time exceeded setup time
4.           m_d ← deriveFrom(m_r);//generate drop call message
5.           inform(a_{i-1}(m_r), m_d); // inform drop call to previous agent
6.       else
7.           if(loop(m_r)) // loop detected in current call route
8.               m_p ← deriveFrom(m_r); //generate penalty message
9.               x ← m_p.loopEndHopCount(a_i);//distance of a_i from the loop end
10.              penalty = (-1)0.9^{x+1};//penalty amount, x = 0 for the first loop agent
11.              m_p.addPenalty(penalty);
12.              m_p.setSourceNode(a_i); //set the source node id to the node where the loop was detected
13.              inform(a_{i-1}(m_r), m_p);//penalise previous loop agent
14.          if(a_d(m_r) == a_i) //this node is the destination
15.              m_c ← deriveFrom(m_r);//generate connect call message
16.              allocateUnitBandwidth(m_c);
17.              if PTC //see sections 4.3.1, and 4.3.2
18.                  append s(i) to m_c;
19.              if TPOT-RL//see section 3.2.3
20.                  storeMessage(m_r);// accumulate estimates
21.              inform(a_{i-1}(m_r), m_c);// inform connect to previous agent
22.          else
23.              //select a neighbour based on Q-estimates for the given destination
24.              a_j ← selectNeighbour(Q_i, a_d(m_r));
25.              preAllocateCall(m_r); //pre-allocate bandwidth
26.              if TPOT-RL
27.                  append Q_i(e_{i,i+1}, a_d(m_r), a_{i+1}) to m_r;// send own estimate
28.              inform(a_j, m_r);// forward call to selected agent
29.              if QR //see section 3.2.2
30.                  //selected neighbour a_j returns its local estimate of bandwidth availability
31.                  E_j(a_d(m_r)) ← min(s(j), max_{a_k ∈ K_j} Q_j(a_d(m_r), a_k))
32.                  Q_i(a_d(m_r), a_j) ← (1 − α)Q_i(a_d(m_r), a_j) + αE_j(a_d(m_r));
33. else if(m_d) // DROP CALL MESSAGE
34.      deAllocateCall(m_d);//de-allocate bandwidth
35.      if(a_i ≠ a_s(m_d)) //if not source
36.          inform(a_{i-1}(m_d), m_d);
37. else if(m_c) // CONNECT MESSAGE
38.      if TPOT-RL and m_c^r//reward message in TPOT-RL
39.          R_i ← computeReward(B_{a_{i+1}(m_c^r),...,a_d(m_c^r)});
40.          Q_i(e_{i,i+1}, a_d(m_c^r), a_{i+1}(m_c^r)) ← (1 − α)Q_i(e_{i,i+1}, a_d(m_c^r), a_{i+1}(m_c^r)) + αR_i;
41.      else
42.          allocateUnitBandwidth(m_c);
43.          if PTC //see sections 4.3.1, and 4.3.2
44.              // compute reward using own estimate and others'
45.              R_i ← computeReward(B_{a_{i+1}(m_c),...,a_d(m_c)});
46.              Q_i(a_d(m_c), a_{i+1}(m_c)) ← (1 − α)Q_i(a_d(m_c), a_{i+1}(m_c)) + αR_i;
47.              append s(i) to m_c;
48.      if(a_i ≠ a_s(m_c)) //if not source
49.          inform(a_{i-1}(m_c), m_c);
50. else if(m_p) // PENALTY MESSAGE
51.      if TPOT-RL
52.          Q_i(e_{i,i+1}, a_d(m_p), a_{i+1}(m_p)) ← (1 − α)Q_i(e_{i,i+1}, a_d(m_p), a_{i+1}(m_p)) + αm_p.penalty;
53.      else
54.          Q_i(a_d(m_p), a_{i+1}(m_p)) ← (1 − α)Q_i(a_d(m_p), a_{i+1}(m_p)) + αm_p.penalty;
55.      if(a_i ≠ a_s(m_p))
56.          d ← m_p.loopEndHopCount(a_i);//distance of a_i from the loop end
57.          penalty = (-1)0.9^{d+1}; m_p.addPenalty(penalty); deAllocate(m_p);//de-allocate bandwidth
58.          inform(a_{i-1}(m_p), m_p);
59. if TPOT-RL //tasks specific to TPOT-RL
60.      monitorLinkUsage(K_i); //measure usage of all l_{i,j}, a_j ∈ K_i
61.      if(t % update-interval == 0)
62.          for all accumulated m_r
63.              m_c^r ← deriveFrom(m_r); //create reward message
64.              inform(a_{i-1}(m_r), m_c^r); //transmit upstream along this m_r's route
65. t ← t + 1;
```

Figure 4.1: Agent actions in response to various message types

to continue this process of allocation (line 42) and sending the message (line 49) until $a_s(m_c)$ is reached; at which point a complete circuit is established. However, if an agent uses the TPOT-RL algorithm, before sending the m_c (line 21), it stores the estimates obtained along with the m_r (line 20); how such estimates are propagated in TPOT-RL is described shortly.

In the two information-sharing heuristics that we have designed based on PTC (described in sections 4.3.1 and 4.3.2), an agent a_i attaches its own node state ($s(i)$) to the message m_c (lines 18 and 47).[2] Each path agent a_i, using PTC, upon receiving an m_c that contains the node state information transmitted from other agents, computes a reward R_i as a function of the set of communicated state values $\mathcal{B}_{a_{i+1}(m_c),\ldots,a_d(m_c)}$ contained in m_c (line 45). Sections 4.3.1 and 4.3.2 define two heuristics for calculating this reward value. In either case, however, the reward is used to update the prior Q-estimates (line 46). The update rule follows standard Q-learning, where α is the learning rate. Thus, the agents cooperatively share their local information to one another to improve their estimates of the unobserved node states.

On the other hand, if a_i is not the destination for this call, it selects one of its neighbours (excluding the one from which it received the m_r) to forward the call request (line 24). This is done by defining a probability distribution over a_i's set of Q-estimates of its neighbours. In particular, the probability of selecting a neighbour a_j is given by:

$$Pr(a_j) = \frac{\exp(\frac{Q_i(a_d(m),a_j)}{\tau})}{\sum_{a_k \in \mathcal{K}_i, a_k \neq a_{i-1}(m_r)} \exp(\frac{Q_i(a_d(m),a_k)}{\tau})}. \quad (4.1)$$

Note that equation 4.1 refers to the selection mechanism when PTC or QR is used. If TPOT-RL is used, then, $Q_i(e_{i,j}, a_d(m), a_j)$ replaces the Q-values in equation 4.1. Here τ is the "temperature" parameter and controls how much the relative differences between various Q-estimates would affect the relative probabilities of selection (the smaller the τ, the larger the skewness). This is a standard heuristic (Boltzmann exploration (Watkins, 1989)) to probabilistically choose between alternative options. Subsequently, a_i pre-allocates a bandwidth unit of its call channel (line 25) and forwards m_r to a_j (line 28). In the QR algorithm, the selected neighbour a_j responds to a_i with its own estimate $E_j(a_d(m_r))$ of bandwidth availability on routes to the destination $a_d(m_r)$ (line 31) which is equal to $\min(s(j), \max_{a_k \in \mathcal{K}_j} Q_j(a_d(m_r), a_k))$ (refer to section 3.2.2). The requesting agent a_i uses this estimate $E_j(a_d(m))$ to update its prior estimate $Q_i(a_d(m), a_j)$ (line 32). In

[2]The PTC principle advocates information distribution only after task completion. However, a task execution process can fail (e.g., call routing failing in our example application). Note that in this situation, PTC can use the task failure as the event to trigger information distribution. We have used this concept and introduced information distribution after call failures (see chapter 5). In so doing, we have observed that the bandwidth allocation quality is better than distributing information only after call successes.

TPOT-RL, a_i appends its Q-estimate to the m_r (line 27) before forwarding the latter to a_j. This is how the sequence of estimates gets propagated along with the call forwarding request in TPOT-RL.

Now we describe a number of activities that an agent performs only if TPOT-RL is used. First, a_i monitors the link usage levels for all its neighbours (line 60). This information is used to compute the value of $e_{i,j}$ ($j \in \mathcal{K}_i$), as described before. Second, every `update-interval` time steps (line 61), a_i starts sending reward messages along the paths of those calls that terminated at a_i during that period (for those m_r's that it had stored during that period). Specifically, these reward messages are analogous to the m_c type message, except that no bandwidth is allocated when an agent receives one (as opposed to bandwidth being allocated when an m_c is received (line 38)). To distinguish from m_c, we denote these messages as m_c^r for our description in figure 4.1. Every `update-interval`, for each m_r stored over the past interval, a_i creates an m_c^r (line 63) and sends this to the neighbour (line 64) which is the immediate upstream node along that call path. Upon receiving an m_c^r, an agent computes a reward using an aggregation of the information of the subsequent path nodes (line 39) (similar to PTC) and updates its Q-values with this reward (line 40) before sending the m_c^r upstream. It should be noted here, the information used by the agents to compute the reward in TPOT-RL are the Q-estimates that the agents had appended while forwarding the m_r. Therefore, the set $\mathcal{B}_{a_{i+1}(m_c^r),\ldots,a_d(m_c^r)}$ in line 57 represents *estimates* and not actual node states. We have used the same aggregation method for TPOT-RL as described in section 4.3.2.

The above discussion corresponds to a more detailed description of the general domain description of section 3.1. As identified in that description and observed in the above system description, the decision to select a specific neighbour to forward a call is critical in determining how effective the system is in successfully routing calls. Since this decision is taken based on the Q-estimates, the more accurately they reflect the true node bandwidth availabilities, the better informed are the decisions taken by an agent. It is emphasised that information-sharing plays a key role in determining the estimation accuracy. In the following, we formulate two simple heuristics of PTC that are used to define the *computeReward* function in our simulations.

4.3 PTC Information-Sharing Heuristics

The discussion on agent interactions in section 4.2 explains how the agents using PTC delay transmitting the information until a call is connected. In this case, the agents along the call path can aggregate the information received from those "downstream" and pass on that information to the previous path agent. We have formulated two simple heuristics for information aggregation based on PTC, viz., to *average* the state

estimates (termed PTC-A) and to take the *minimum* state estimate (termed PTC-M). These are described in the following.[3]

4.3.1 PTC Inform Average Capacity (PTC-A)

In figure 4.1, upon receiving the m_c, an agent using the PTC-A heuristic computes a reward value R_i by *averaging* the states of all nodes on the route from a_i to a_d as:

$$R_i = \frac{\sum_{k \in \{i+1,\ldots,d\}} s(k)}{L(i,d)}, \tag{4.2}$$

where $L(i,d)$ is the hop count on this route from a_i to a_d. This describes how the function *computeReward* of figure 4.1 is implemented. Subsequently, this reward is used to update its prior Q-estimate. Thus the estimates are updated with the information about the resource usage on the "downstream" nodes on the path of this call.

4.3.2 PTC Inform Minimum Capacity (PTC-M)

This is similar to PTC-A: but instead of average available capacity, the *minimum* available capacity is used as reward. For example, agent a_i using the PTC-M heuristic computes the reward as:

$$R_i = \min(s(i+1), \ldots, s(d)). \tag{4.3}$$

This is a more conservative estimate of bandwidth availability than the average capacity model. Thus it has the advantage that the probability of a dropped call due to agents overestimating the bandwidth availability is reduced. This heuristic is also used to aggregate estimates in TPOT-RL in our experiments.

4.4 Experimental Setup

Based on the scenario discussed in section 4.2, we have conducted a series of experiments to empirically evaluate the effectiveness of PTC compared to the benchmarks (QR and TPOT-RL). In this section, we start by enumerating the measures chosen as indicators of system performance. Subsequently, the results obtained from the experiments on these measures are analysed.

[3]Here, it should be noted that the formal analysis in section 3.3 is based on the agents having separate estimates of individual agents. Maintaining an aggregate estimate on a set of agents, however, reduces the computational complexity at decision time in our simulations. Alternatively, for example, with estimates of individual nodes, the run-time complexity of determining the least cost path between any two nodes is quadratic in the number of nodes in the graph (the worst case run time of Dijkstra's single-source shortest path algorithm (Cormen et al., 2001)).

4.4.1 Performance Measures

The following measures are chosen to evaluate the performance of a given communication protocol in a TN.

4.4.1.1 Number of Successful Calls

The overall objective of the cooperative agent network is to maximise the number of successfully routed calls, given the set of resources (call-channel bandwidth) available, the rate at which new calls originate, and the duration calls remain connected (hold up resource). Thus, the average number of successful calls determine how successful a given protocol has been given the above parameter values. In our system, we record the total number of calls that have originated (NO) and the total number of those calls that have been successfully routed (NC). Thus, the value $x = NC/NO$ determines the fraction of successful calls routed over the time period in which the measurements are taken.

In addition to measuring the number of calls successfully routed, we keep track of the number of calls that could be connected if the agents had global knowledge about the network bandwidth availability and if call routing could be done instantaneously. This is computed by globally searching for the availability of a path for each call. This search is done instantaneously at the beginning of each simulation time step. We term this essentially idealistic procedure the "**I**nstantaneous **Z**ero **D**elay **S**earch" (IZDS). Since, in practice, it takes a finite number of time steps to connect a call, the IZDS is repeated at every time step until either it finds a path or the call is dropped/connects.[4] In case of the former outcome, the NC_{izds} count is incremented by 1. The value $x_{izds} = NC_{izds}/NO$ gives a hard upper limit to the call success rate under the given conditions.[5] The various strategies are compared both against the absolute success rate (x) values and the percentage deviation of success rate from IZDS, $(x_{izds} - x)/x_{izds}$. It is infeasible for any practical system to attain the IZDS success rate because, in practice, there is a finite amount of delay to connect a call as opposed to the instantaneous connection in IZDS. Also, the agents in a practical system attempt to connect a call by forwarding it one hop at a time based on their individual estimates of the world states. IZDS, on the other hand, takes a global and accurate view of the entire network to find the least cost path.

[4]Thus, IZDS is guaranteed to find a path if one is found by the actual routing algorithm, but not necessarily vice versa.

[5]Note, IZDS is not an optimal measure. Since it tries to connect a call as soon as one originates, it is essentially a greedy strategy. Some other scheduling strategy, say that uses some form of lookahead before attempting to place a call, may outperform IZDS.

4.4.1.2 Successful Routes of Different Lengths

The call success rate metric, described in section 4.4.1.1, measures the overall rate of successful calls in the system. This is an important performance measure because it indicates how the system performs at a broad scale. So it is an indicator of how successful a communication protocol is in improving the system performance. However, it does not indicate how effective the protocol is at connecting calls at a given distance (since the success rate metric counts all calls in the system).

In our case, calls can be required to be routed to destination nodes that are at various distances from the nodes of origin. Calls destined for nodes that are at short distances are relatively easier to route than those further away. This is because more accurate estimates of the load at nodes that are nearer can be maintained. As shown in the analysis of section 3.3, the farther an agent, the longer is the delay for the information to arrive and, thus, the less up-to-date are the estimates. Therefore, it is less probable for a call to be routed successfully to such a distant node. A communication protocol that allows better success rates at longer distances can, therefore, be considered more competent than another that achieves a poorer success rate at long distances (all other conditions being equal).

Thus we measure the number of successful call connections for various distances between the call source and destination nodes. More specifically, the minimum hop count (say, d) between the source and destination of a call is computed (global knowledge of the network topology is used to measure this distance) and the success counter (NC_d) of calls at distance d is incremented if such a call is successfully connected. In this manner, a success rate for calls at a distance d can be computed as $x_d = NC_d/NO_d$, where NO_d stands for the number of calls originated with a source-destination distance of d. For different values of d, therefore, the different x_d values could be used to compare the ability of different communication strategies to place calls at different lengths.

4.4.1.3 Reward Information Messages

The agents in our system use different message types for communication (section 4.2 enumerates these). However, the most important among these are the ones that carry the reward information by which the state value of one agent is transmitted to another. This is because using this information, the agents update their prior estimates of the network load level. Therefore, these messages contribute directly to the quality of learning and to the overall performance of the system in allocating resources to place calls. In QR, for example, this is the message that an agent receives from its neighbour after handing over to the latter a call forwarding request (see section 3.2.2 for details). Therefore, at each call forwarding step, a new message is generated by the contacted agent and

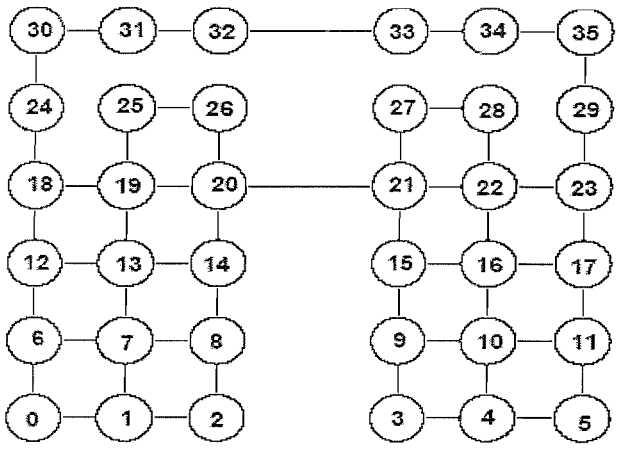
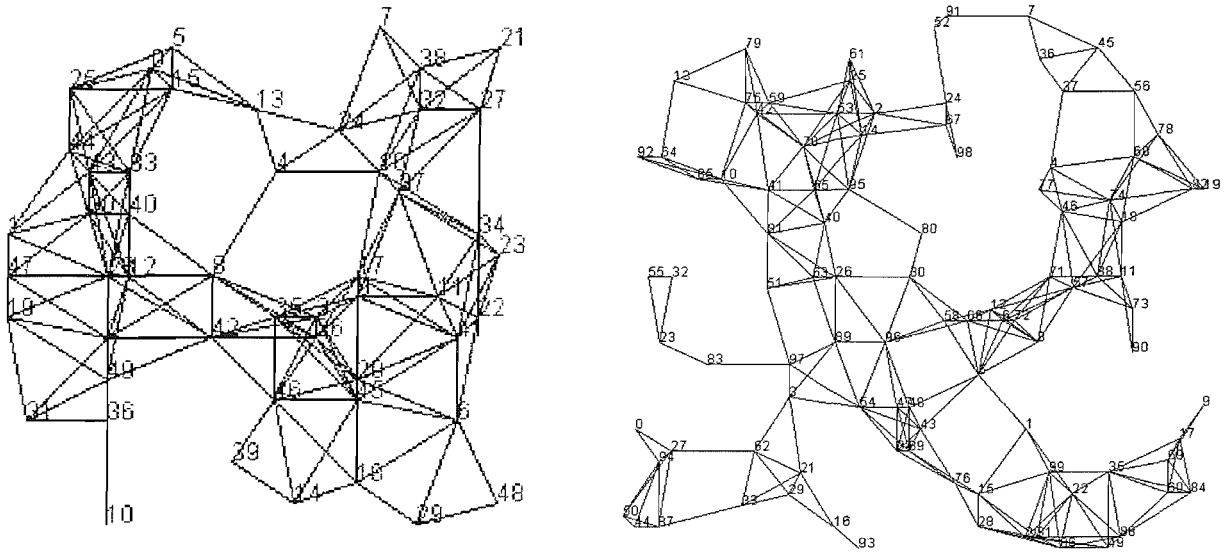


Figure 4.2: A 36-node irregular grid topology

transmitted to the contacting agent. In the PTC-based models, on the other hand, these are the m_c type messages, transmitted after a call connects, that contain the summary reward information (sections 4.3.1 and 4.3.2 have the details). Note, this is only one message generated by the destination node after every successful routing and transmitted upstream along the call path.[6] Although several other types of messages (e.g., m_r, m_p, and m_d) are exchanged between the agents, it is the messages that contain the reward that affect the learning quality the most. On the other hand, in our implementation of TPOT-RL, an agent, while forwarding a call, transmits along with the call, its own estimate of the bandwidth availability along paths to the call destination. Subsequently, this information is used to update the Q-values of the agents (see section 4.5 for details). Therefore, these messages affect the learning of agents in the way that m_c does in PTC. Hence, the message rate for TPOT-RL is measured by counting these messages.

The number of such messages can, therefore, be used as a measure of the efficiency of a given communication protocol — the lower the number of messages, the higher is the efficiency (assuming a given value of some other performance measure such as call success rate). More specifically, the total number of information exchanges (represented by m, say) for transmitting the agents' state values to one another is computed at every T time steps during a simulation. The value $r = m/T$, therefore, gives the rate of messages transmitted in the entire system during the interval T. The total simulation is divided into several intervals and the values of r over each such interval generate an overall time-variation of the message rate. The different communication strategies are then compared against various message rates.

[6]In chapter 5, we investigate a modified version of the PTC-based heuristics where information is shared after a call fails in addition to a call successfully connecting. A whole new set of capabilities, those of adaptiveness to failures and diagnosing failures, that are not present in the current strategies are achieved by this modification.



(a) 50 node random graph

(b) 100 node random graph

Figure 4.3: Random network topologies

4.4.2 Network Characteristics and Experimental Parameters

Experiments are conducted on a number of different network topologies. In the following, we report our results and observations based on some of those. Figures 4.2 and 4.3 show, respectively, a 36-node irregular grid and two randomly generated topologies — a 50-node random graph (figure 4.3(a)), and a 100-node random graph (figure 4.3(b)). The 36-node irregular grid topology has been used in previous papers on the application of RL in network routing (see section 2.2 for a discussion on these papers) and thus we chose it to make valid comparisons. To verify our conclusions across a wider range of topologies, we tested the same against random graphs (figures 4.3(a) and 4.3(b) show two such examples) of different sizes.[7] In all figures, the nodes are numbered for ease of reference. The edges between nodes indicate that those nodes are within each other's radio range. Note that the random graphs are designed such that any node is linked to only those within a certain maximum radial distance which simulates the transmission range of wireless nodes.

The following parameter values were used for all experiments reported henceforth (unless otherwise stated): learning rate $\alpha = 0.03$, Boltzmann exploration temperature $\tau = 0.1$, call setup time (t_{live}) of 36, 50, and 100 time steps for the topologies in figures 4.2, 4.3(a) and 4.3(b), respectively. We allowed for a larger call setup time in the bigger topologies to give allowance for the larger size of the networks. An average

[7]Various other topologies were used with varying number of nodes and connectivity patterns and the same general trends in the results were observed. Hence, here, we report on three sample topologies.

call duration of 20 times the setup time was used. The "load" in the network is set by assigning a probability with which calls originate at every time step in the network. This call origination probability was varied to study the effect of different network loads on the performance. Calls were allowed to originate and terminate on any randomly selected node. We have tested the strategies using both (i) a constant-load simulation where the call origination probability is maintained the same throughout a simulation run, and (ii) a dynamically changing load simulation where the call origination probability is changed during the course of a simulation run. We have used different numbers and values of the call origination probability changes in a single simulation run to test the effect of various degrees of load fluctuations on the performance of PTC, QR, and TPOT-RL. In these dynamic settings, the call origination probability is changed at equal intervals in a simulation run. A single simulation run lasted for 500,000 time steps for the topology in figure 4.2, for 1,000,000 time steps for the topology in figure 4.3(a), and for 2,000,000 time steps for the topology in figure 4.3(b). Results are averaged over 10 simulation runs (these figures are statistically significant at the 95% confidence level). Also, every node had a maximum call channel capacity of 10 units.

4.5 Results and Analysis

In this section, the performance of QR, PTC-A, PTC-M, and TPOT-RL are compared against the measures described in section 4.4.1. Specifically, results on the overall success of the various communication heuristics in connecting calls is presented in section 4.5.1, the effectiveness of these heuristics in successfully connecting long-distance calls is shown in section 4.5.2, and the overhead due to communicating messages is analysed in section 4.5.3.

4.5.1 Performance — Call Success Rate

We anticipate that the higher the accuracy of learned estimates of unobserved states, the more capable the agents will be of routing calls to the destination via the most appropriate paths. Hence, in turn, the higher will be the call success rate and the lower the deviation from the IZDS success rate. In the following, the results from constant load are presented first followed by those obtained from dynamically varying load.

4.5.1.1 Constant Load.

We experimented with all strategies to calculate: (i) the call success rates, and (ii) the percentage deviation of the measured success rate from the IZDS, under steady state

conditions (i.e, when the call throughput in the system reached a steady value).[8] The average success rate and the average IZDS success rate, both computed during the steady state phase of the simulation, are further averaged over 10 simulation runs for a given value for the call origination probability. Further, call success rate is measured against various call origination probabilities to test the impact of network load on the success rate. These measurements are repeated for each of the three topologies.

Table 4.1: Call success rates for all strategies -- topology of figure 4.2

Load	QR			PTC-A			PTC-M			TPOT-RL		
	Avg	Stdev	IZDS	Avg	Stdev	IZDS	Avg	Stdev	IZDS	Avg	Stdev	IZDS
0.1	50.94	0.0048	72.66	50.98	0.0076	72.11	**51.85**	0.0052	70.25	25.74	0.0022	99.57
0.2	32.41	0.0034	60.69	32.64	0.0037	60.56	**33.02**	0.0039	58.86	19.94	0.0016	97.65
0.4	19.99	0.0018	55.37	20.27	0.0021	55.23	**20.38**	0.002	52.79	14.82	0.0009	89.44
0.6	14.87	0.0012	54.87	15.07	0.0011	53.44	**15.08**	0.0014	50.6	11.52	0.0012	85.2

Table 4.2: Call success rates for all strategies — topology of figure 4.3(a)

Load	QR			PTC-A			PTC-M			TPOT-RL		
	Avg	Stdev	IZDS	Avg	Stdev	IZDS	Avg	Stdev	IZDS	Avg	Stdev	IZDS
0.1	57.31	0.0042	81.75	58.15	0.0052	81.67	**58.49**	0.0052	80.9	12.18	0.0027	99.99
0.2	37.11	0.002	69.75	37.32	0.0028	68.73	**37.65**	0.0026	68.33	10.21	0.0011	99.8
0.4	22.98	0.0012	61.44	23.35	0.0012	62.6	**23.51**	0.0013	60.43	7.46	0.0007	99.8
0.6	17.17	0.001	58.8	17.47	0.0009	61.29	**17.49**	0.0008	57.53	6.11	0.0004	99.8

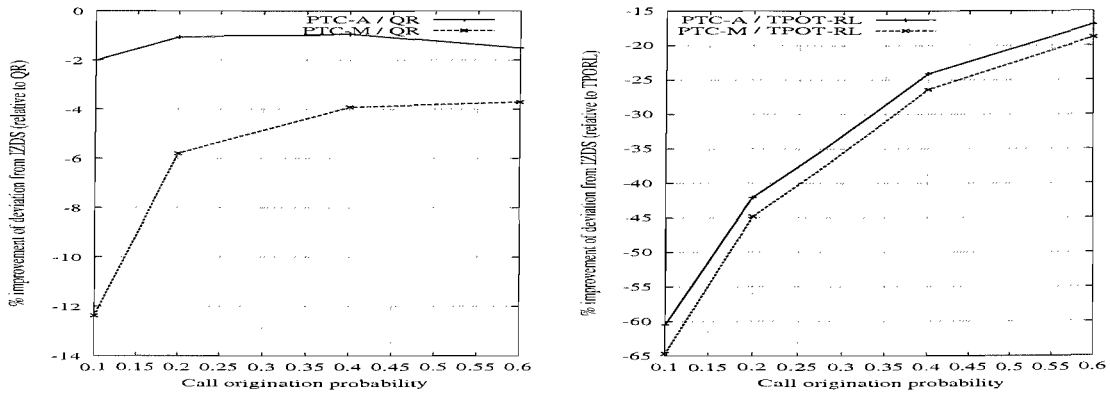
Table 4.3: Call success rates for all strategies — topology of figure 4.3(b)

Load	QR			PTC-A			PTC-M			TPOT-RL		
	Avg	Stdev	IZDS	Avg	Stdev	IZDS	Avg	Stdev	IZDS	Avg	Stdev	IZDS
0.1	35.79	0.0032	66.52	35.86	0.0034	65.58	**37.34**	0.003	62.65	8.53	0.0031	99.88
0.2	24.17	0.0017	57.41	24.56	0.002	59.18	**24.8**	0.0014	54.27	6.12	0.0013	99.83
0.4	15.88	0.0011	54.01	16.16	0.0007	55.53	**16.18**	0.0008	50.72	4.79	0.0006	99.9

In more detail, table 4.1 shows the average steady state success rates achieved by the different strategies and by IZDS used alongside these strategies for different network loads (call origination probabilities) in the topology of figure 4.2. Tables 4.2 and 4.3 show the same measurements for the topologies in figure 4.3(a) and 4.3(b), respectively.

[8]Note, our TN application represents a dynamic system where node bandwidth availability changes with time (new calls are placed and existing calls terminate). The Q-values estimate the bandwidth availability. So, as bandwidth availability changes, so do the Q-values. However, the call success rate is an overall system measure which reaches a steady state when a constant call origination probability is used.

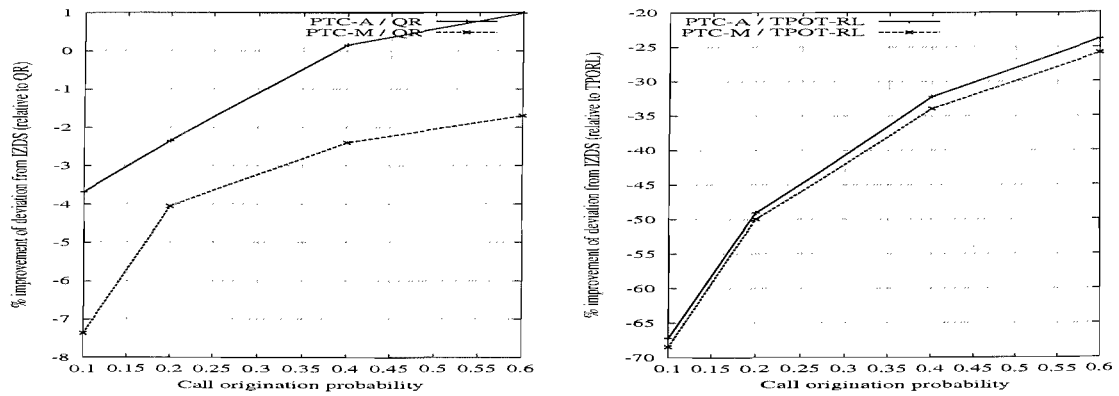
The results indicate that the average steady state call success rate (shown under column "Avg" in these tables) achieved with PTC-M dominates that of the other strategies under different network loads. For example, in table 4.1, when the load is 0.1, PTC-M achieves a steady state average call success rate of 51.85%, PTC-A achieves 50.98%, QR 50.94%, and TPOT-RL 25.74%. As indicated by the analysis of section 3.3, PTC maintains more up-to-date information of the network states. Thus, using PTC, the agents are capable of taking better informed decisions of forwarding a call which, in turn, ensures a higher likelihood of successful connections. This is reflected in the (statistically significant) higher call success rate achieved by PTC-M over all other strategies. In particular, since the minimum-capacity heuristic restricts overestimation of the node bandwidth availability, it generates a slightly better success rate than the average capacity heuristic.



(a) PTC against QR

(b) PTC against TPOT-RL

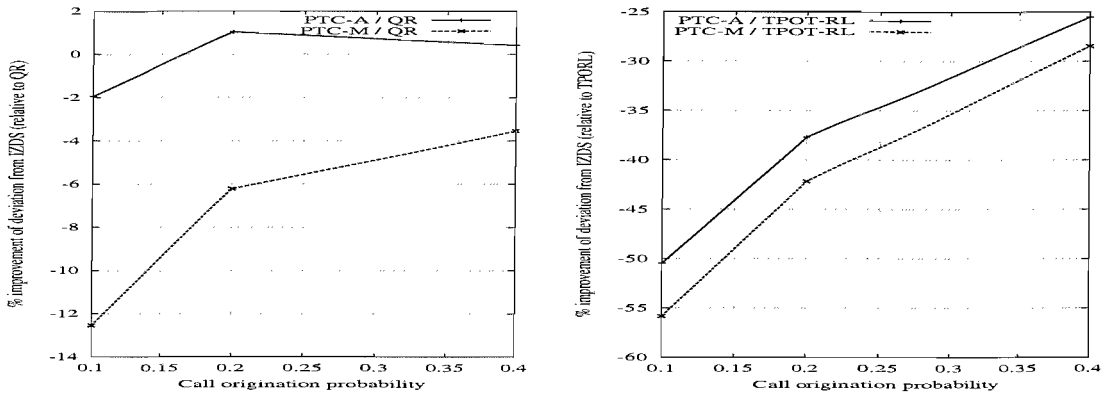
Figure 4.4: Improvement of call success rate deviation from IZDS relative to QR and TPOT-RL — topology of figure 4.2



(a) PTC against QR

(b) PTC against TPOT-RL

Figure 4.5: Improvement of call success rate deviation from IZDS relative to QR and TPOT-RL — topology of figure 4.3(a)



(a) PTC against QR

(b) PTC against TPOT-RL

Figure 4.6: Improvement of call success rate deviation from IZDS relative to QR and TPOT-RL — topology of figure 4.3(b)

We have also measured the improvements in the success rate values achieved from the PTC-based information strategies relative to those of QR and TPOT-RL. To do so, first the success rate deviation from IZDS is computed as $p = (x_{izds} - x)/x_{izds}$ (see section 4.4.1.1). Subsequently, p_{PTC-M} and p_{PTC-A} are compared to p_{QR} and $p_{TPOT-RL}$ as: $(p_{QR} - p_{PTC})/p_{QR}$ and $(p_{TPOT-RL} - p_{PTC})/p_{TPOT-RL}$, respectively. The graphs in figure 4.4 show the relative improvements of the average success rate deviation from IZDS achieved by using PTC over QR (figure 4.4(a)) and over TPOT-RL (figure 4.4(b)) when the topology in figure 4.2 is used. Note that in this figure *a greater negative value indicates less of a deviation from the IZDS relative to QR or TPOT-RL, and, hence, a better performance of PTC.* From these graphs, a significant improvement is observed in PTC-M relative to both QR and TPOT-RL under different values of network load (statistical significance is tested at the 95% confidence level). For example, (see the row corresponding to "Load" = 0.1 in table 4.1) with a call origination probability of 0.1, the deviation of the call success rate from the IZDS is 26.2% (= (70.25 − 51.85)/70.25) for PTC-M and 29.9% (=(72.66 − 50.94)/72.66) for QR. Therefore, the success rate due to PTC-M is 12.37% (=| (26.2 − 29.9)/29.9 |) closer to IZDS relative to that of QR. The deviation of the success rate of PTC-A from IZDS also remains lower relative to that of QR in figure 4.4(a). Similarly, comparing PTC-M with TPOT-RL at the same load level, PTC-M achieves a 26.2% deviation from IZDS while the corresponding figure for TPOT-RL is 74.15% (=(99.57 − 25.74)/99.57). Therefore, the success rate due to PTC-M is 64.66% (=| (26.2 − 74.15)/74.15 |) closer to IZDS relative to that of TPOT-RL. The performance of PTC-M dominates that of PTC-A because of the conservative nature of the minimum capacity heuristic compared to the average capacity (as stated above). These results further strengthen our analysis in section 3.3 that by providing better quality estimates, PTC performs better than QR.

An additional observation is that with increasing load, the relative improvement of PTC over QR and TPOT-RL reduces. Thus, in figure 4.4(a), the improvement of PTC-M over QR is 12.37% with load of 0.1, while it is 3.7% with load of 0.6. Similarly, in figure 4.4(b), PTC-M is 64.66% better than TPOT-RL at load 0.1, but 18.82% better at load 0.6. Note that an increase in load implies reduction in the time between successive task processing episodes since calls originate more frequently with increased load. As explained in section 3.3, the smaller the value of the interval between successive task processing episodes, the less is the difference between the timeliness of information distribution of PTC and NN. In this context, we notice that with increasing load, the success rate deviation of PTC-A from IZDS is slightly higher than that of QR (see figures 4.5(a) and 4.6(a) where the "PTC-A / QR" plots show positive values at higher loads). Now, since IZDS is executed on a network *simultaneously* with the given communication heuristic (PTC, QR, or TPOT-RL), its performance is affected in a way that depends on how the heuristics perform. Thus, although the deviation of call success rate of PTC-A from that of IZDS is slightly higher than that of QR in some of the cases, it does not necessarily indicate a poorer performance of PTC-A compared to QR. Moreover, we have shown in tables 4.2 and 4.3 that PTC-A always generates a higher call success rate than QR, thereby establishing a superior performance of PTC-A. Hence, although the performance differences between PTC and the other strategies decrease with an increase in the load, PTC still maintains a clear advantage.

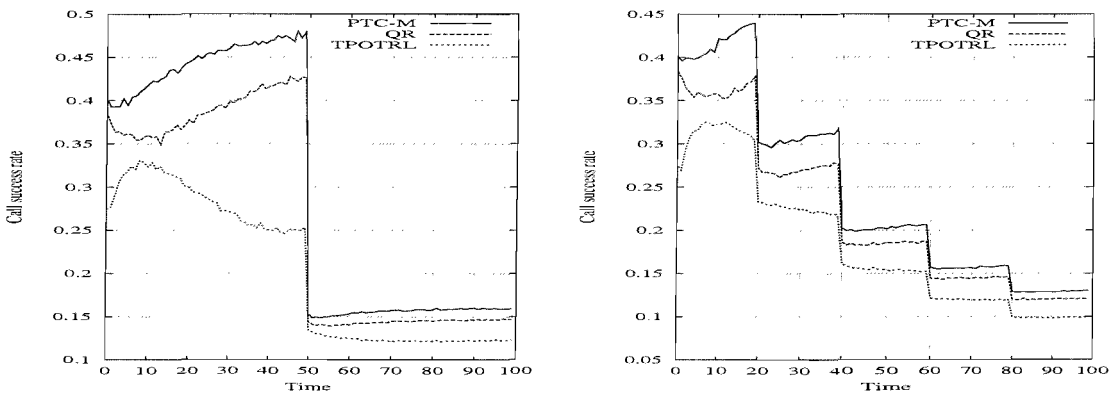
We observe identical trends in the performances of PTC-M relative to the benchmarks with the other topologies. Figure 4.5 shows the results for the topology in figure 4.3(a), and figure 4.6 for the topology in figure 4.3(b).

4.5.1.2 Dynamically Changing Load.

In contrast to the steady-state call success rate measured with constant load, here we present the time-variation of the call success rate as the network load fluctuates. This captures the responsiveness of the system call success rate to dynamically changing load levels given a particular information-sharing protocol. To this end, figure 4.7(a) shows the time-variation of the call success rates of PTC-M, QR, and TPOT-RL as the call origination probability is increased from 0.1 to 0.6 in a simulation run using the topology of figure 4.2.[9] It demonstrates that the PTC (in this case we show PTC-M; PTC-A is excluded from the results since it performs slightly worse than PTC-M and better than QR and TPOT-RL) call success rate remains the highest both when the load level is 0.1 (before time = 50) and when it increases to 0.6 (after time = 50). The call success rates of all strategies suffer a drop with the increase in load level since the nodes have

[9]We have experimented with all the other topologies under conditions of dynamically changing load. The broad patterns observed in the results are identical across all of them. Hence, we choose one sample topology to report our results in this thesis.

only limited bandwidth to allocate calls, a result that we have observed previously with constant load (see table 4.1). We further experimented with multiple load fluctuations over a simulation run. Figure 4.7(b) shows the results when the load level changes between five different values: 0.1, 0.2, 0.4, 0.6, and 0.8 in that order. In all cases, PTC is observed to have the highest call success rate out of PTC-M, QR, and TPOT-RL. Note that in these results, we have used two different types of load fluctuations: one large increase (0.1 to 0.6) and monotonically increasing. There can of course be several other patterns of load variations, such as random fluctuations or following specific probability distributions (e.g., Poisson) but these are not considered in this work. In this context, we identify that the impact of a load change is that the agents have to re-learn the new environmental condition so that calls can be placed effectively. The most severe case in this situation is that of increase in load because a higher load demands more efficient allocation of (limited) resources. Thus, having shown that PTC outperforms the benchmarks under this condition, we envisage that similar broad trends would be observed for other patterns.



(a) With 2 degrees of dynamism

(b) With 5 degrees of dynamism

Figure 4.7: Time variation of call success rates of QR, PTC-M, and TPOT-RL with network load fluctuations — topology of figure 4.2

Now we aim to summarise the effects of dynamically changing load on the network call success rate given an information-sharing protocol. In so doing, we first designate the number of load levels in a simulation run as the "degree of dynamism". For example, figure 4.7(a) has a degree of dynamism of 2 and figure 4.7(b) has 5 degrees of dynamism. Then, for a given degree of dynamism, we compute the percentage difference of the call success rates of QR or TPOT-RL using PTC-M as the baseline as: $(r^t_{QR} - r^t_{PTCM})/r^t_{PTCM}$, and $(r^t_{TPOTRL} - r^t_{PTCM})/r^t_{PTCM}$, where r^t is the time-varying call success rate. Note that this difference measure is also time-dependent. To summarise the improvement of call success rate using PTC-M, we find the minimum, the mean, and the maximum of this difference over every time interval during which the

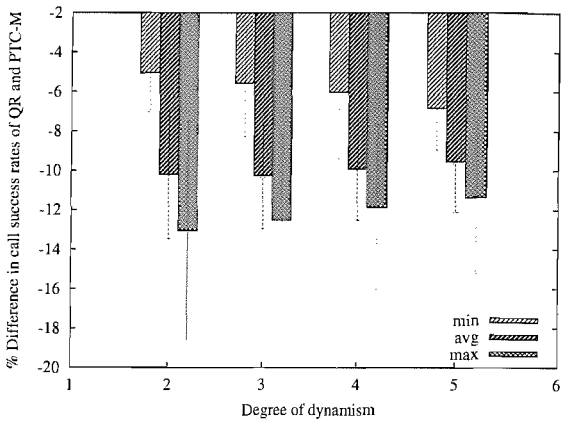


Figure 4.8: Summary statistics of call success rate differences between QR and PTC-M with network load fluctuations — topology of figure 4.2

load-level remains constant. These statistics present the call success rate improvement range within the time interval when the load remains at a certain level. Subsequently, the means and the standard deviations of each of the minimum, the mean, and the maximum differences over all such intervals are computed. This step generates the summary of the different improvement ranges across all such intervals. For example, in figure 4.7(a), the minimum, mean, and maximum percentage differences are computed in the interval where load = 0.1 and in the interval where load = 0.6. Subsequently, the mean and standard deviations of these two sets of difference statistics are computed. Figure 4.8 shows the above-mentioned measures for all degrees of dynamism used in our experiments when QR is compared to PTC-M. Since TPOT-RL is always observed to achieve a lower call success rate rate than both PTC-M and QR, we exclude the summary comparison of TPOT-RL with PTC-M. Along the horizontal axis of figure 4.8 is the degree of dynamism while the summary statistics of the percentage success rate difference are along the vertical axis. In this figure, a degree of dynamism 2 indicates the load level is changed from 0.1 to 0.2; 3 indicates a change of 0.1, 0.2, 0.4; 4 indicates 0.1, 0.2, 0.4, 0.6; and 5 indicates 0.1, 0.2, 0.4, 0.6, 0.8. The negative values of the call success rate differences indicate r^t_{QR} is lower than that of r^t_{PTCM}. The figure shows that the gap between the minimum and the maximum differences reduces with the number of degrees of dynamism. For instance, this gap is approximately 8% with degree of dynamism 2, and it is about 4.5% when the degree of dynamism is 5. This is because, with a higher number of load changes within a given simulation run, there is less time for any strategy to re-learn the changes in the environment. Hence, the call success rates do not attain their steady state values (as in table 4.1). Nevertheless, the average statistics of the call success rate differences show that PTC-M achieves a significantly higher call success rate across all degrees of dynamism. For example, this is about 10.2% with 2 degrees of dynamism and 9.5% with 5 degrees of dynamism.

Table 4.4: Call success rates at various distances — topology of figure 4.2

Load	Strategy	Min hop count									
		1	2	3	4	5	6	7	8	9	10
0.1	QR	**0.87**	**0.76**	0.63	0.51	0.42	0.37	0.355	0.343	0.344	0.34
	PTC-A	0.863	0.748	0.632	0.516	0.426	0.374	0.355	0.345	0.348	0.354
	PTC-M	0.857	0.753	**0.643**	**0.534**	**0.438**	**0.381**	**0.362**	**0.356**	**0.353**	**0.365**
	TPOT-RL	0.633	0.532	0.356	0.218	0.164	0.148	0.126	0.094	0.071	0.059
0.2	QR	**0.785**	**0.592**	0.44	0.31	0.222	0.178	0.158	0.147	0.147	0.139
	PTC-A	0.773	0.583	0.44	0.315	0.231	0.181	0.164	0.155	0.151	0.151
	PTC-M	0.768	0.583	**0.444**	**0.326**	**0.237**	**0.187**	**0.168**	**0.155**	**0.151**	**0.152**
	TPOT-RL	0.584	0.462	0.266	0.139	0.101	0.098	0.086	0.066	0.056	0.048
0.4	QR	**0.693**	**0.421**	0.266	0.161	0.103	0.074	0.064	0.0563	0.053	0.05
	PTC-A	0.675	0.417	0.272	0.17	0.111	0.081	0.068	0.06	0.055	0.058
	PTC-M	0.667	0.416	**0.274**	**0.177**	**0.113**	**0.081**	**0.068**	**0.061**	**0.057**	**0.057**
	TPOT-RL	0.496	0.375	0.198	0.086	0.059	0.063	0.054	0.04	0.037	0.029
0.6	QR	**0.632**	**0.326**	0.185	0.103	0.062	0.045	0.037	0.03	0.029	0.027
	PTC-A	0.61	0.323	0.19	0.111	0.068	0.047	0.04	0.035	0.032	0.03
	PTC-M	0.596	0.322	**0.194**	**0.113**	**0.069**	**0.049**	**0.039**	**0.035**	**0.032**	**0.03**
	TPOT-RL	0.418	0.301	0.148	0.065	0.043	0.046	0.037	0.023	0.015	0.013

These observations reinforce our hypotheses that, under the given simulation environment, *the post task-completion information-sharing model with minimum-capacity reward achieves the best call success rate among all strategies under different network loads under conditions of both static and dynamically fluctuating loads.* For example, with constant load, the deviation of the success rate from the IZDS is up to 12.37% lower in PTC-M than QR, and up to 65% lower in PTC-M than TPOT-RL. Further, with dynamically changing load, PTC-M achieves an average 10% improvement in call success rate over QR with five different load-level changes in a simulation run.

4.5.2 Performance — Success Rate for Calls of Different Lengths

The call success rate values reported in section 4.5.1 indicate better performance of PTC-M over QR and TPOT-RL. In addition, to measure the effectiveness of an information-sharing heuristic in connecting a call to a destination that is at a given distance from the source, the measure x_d (see section 4.4.1.2 for its definition) is measured for increasing values of d. Results from the constant load experiments are reported in this section. With dynamically changing load, the summary statistics (as described in section 4.5.1) of call success rates indicated better performance of PTC than the benchmarks for all values of d. Since we have already shown the better success rate of PTC under dynamic load conditions in section 4.5.1, we have excluded the call success rates at different distances under the same conditions in this section.

In more detail, tables 4.4, 4.5, and 4.6 show the values of x_d for different values

Table 4.5: Call success rates at various distances — topology of figure 4.3(a)

Load	Strategy	Min hop count						
		1	2	3	4	5	6	7
0.1	QR	**0.886**	**0.761**	0.6	0.504	0.428	0.379	0.353
	PTC-A	0.876	0.746	0.586	0.495	0.426	0.381	0.365
	PTC-M	0.873	0.753	**0.601**	**0.511**	**0.443**	**0.398**	**0.369**
	TPOT-RL	0.187	0.129	0.11	0.102	0.104	0.139	0.203
0.2	QR	**0.774**	**0.57**	0.371	0.268	0.197	0.161	0.141
	PTC-A	0.77	0.561	0.369	0.265	0.2	0.163	0.145
	PTC-M	0.766	0.564	**0.375**	**0.272**	**0.207**	**0.173**	**0.159**
	TPOT-RL	0.214	0.127	0.085	0.065	0.055	0.06	0.061
0.4	QR	**0.653**	0.383	0.203	0.12	0.782	0.58	0.492
	PTC-A	0.648	**0.385**	0.208	0.126	0.828	0.631	0.53
	PTC-M	0.64	0.384	**0.212**	**0.13**	**0.87**	**0.662**	**0.545**
	TPOT-RL	0.202	0.109	0.054	0.025	0.021	0.024	0.048
0.6	QR	**0.58**	0.286	0.133	0.712	0.428	0.305	0.249
	PTC-A	0.566	**0.289**	0.14	0.781	0.481	0.347	0.287
	PTC-M	0.557	0.287	**0.143**	**0.802**	**0.504**	**0.371**	**0.305**
	TPOT-RL	0.177	0.095	0.046	0.017	0.011	0.0070	0.0050

Table 4.6: Call success rates at various distances — topology of figure 4.3(b)

Load	Strategy	Min hop count											
		1	2	3	4	5	6	7	8	9	10	11	12
0.1	QR	**0.894**	**0.755**	**0.616**	**0.452**	0.324	0.247	0.208	0.175	0.149	0.12	0.087	0.082
	PTC-A	0.89	0.749	0.589	0.432	0.317	0.247	0.22	0.197	0.169	0.141	0.111	0.1
	PTC-M	0.88	0.728	0.598	0.443	**0.336**	**0.267**	**0.238**	**0.223**	**0.191**	**0.164**	**0.124**	**0.122**
	TPOT-RL	0.22	0.172	0.14	0.104	0.079	0.057	0.051	0.045	0.037	0.029	0.028	0.029
0.2	QR	**0.847**	**0.647**	**0.47**	**0.296**	0.189	0.123	0.098	0.079	0.061	0.047	0.036	0.029
	PTC-A	0.837	0.628	0.45	0.287	0.178	0.124	0.1	0.087	0.071	0.056	0.046	0.039
	PTC-M	0.823	0.622	0.448	0.292	**0.192**	**0.136**	**0.109**	**0.096**	**0.086**	**0.066**	**0.053**	**0.046**
	TPOT-RL	0.202	0.158	0.11	0.072	0.047	0.032	0.029	0.025	0.016	0.012	0.008	0.006
0.4	QR	**0.77**	**0.502**	**0.315**	0.17	0.094	0.057	0.041	0.031	0.022	0.017	0.011	0.012
	PTC-A	0.766	0.484	0.298	0.161	0.09	0.057	0.042	0.034	0.026	0.021	0.0145	0.015
	PTC-M	0.746	0.476	0.303	**0.172**	**0.098**	**0.061**	**0.046**	**0.039**	**0.031**	**0.024**	**0.017**	**0.018**
	TPOT-RL	0.189	0.141	0.09	0.056	0.034	0.02	0.0167	0.011	0.006	0.003	0.002	1.0E-4
0.6	QR	**0.71**	**0.4**	**0.225**	0.112	0.06	0.03	0.024	0.018	0.012	0.009	0.006	0.005
	PTC-A	0.706	0.393	0.216	0.109	0.057	0.031	0.025	0.0197	0.014	0.0117	0.008	0.0075
	PTC-M	0.688	0.386	0.221	**0.113**	**0.061**	**0.036**	**0.028**	**0.023**	**0.017**	**0.013**	**0.009**	**0.008**
	TPOT-RL	0.171	0.127	0.079	0.047	0.027	0.015	0.01	0.006	0.003	1.0E-4	1.0E-4	1.0E-4

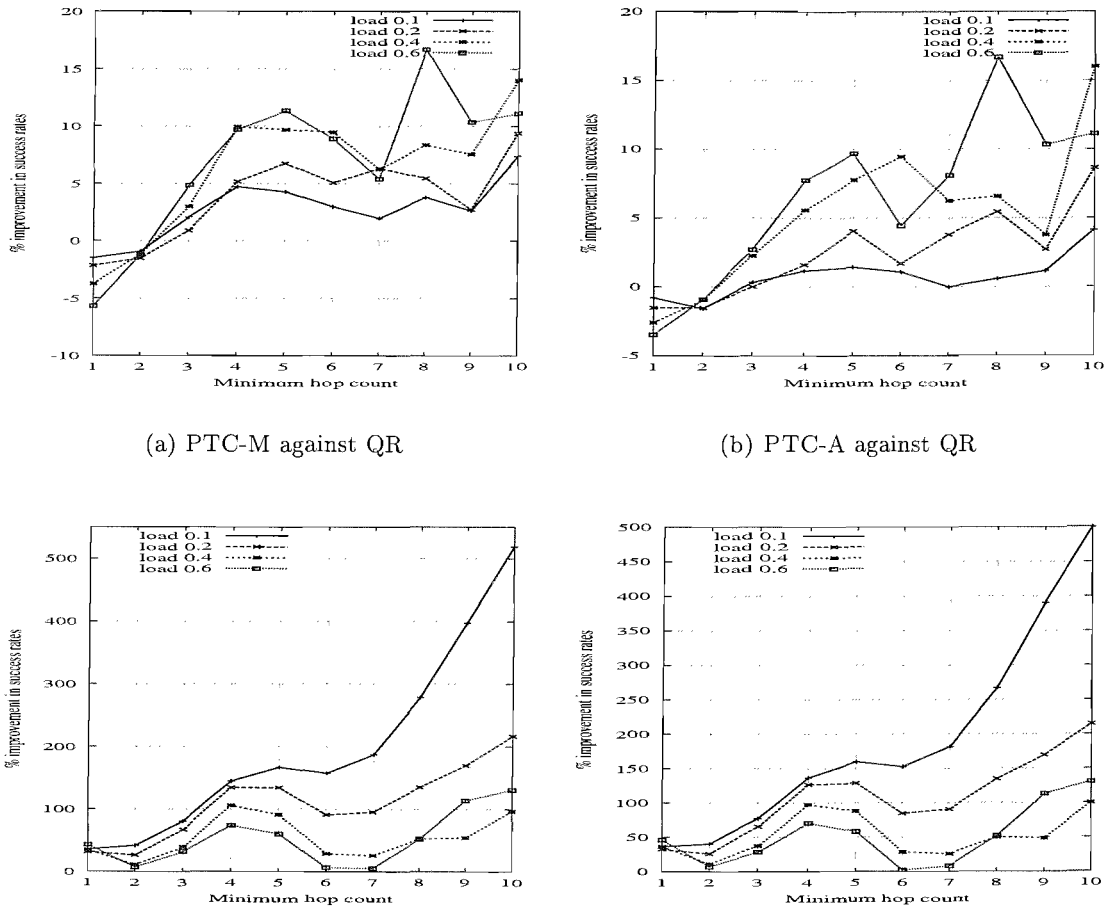
of d (the "min hop count"[10]) under different network loads for the topologies of figures 4.2, 4.3(a), and 4.3(b), respectively. These tables indicate the following trends. For short distances, QR achieves a slightly higher (although, not statistically significant at the 95% confidence level) success rate than PTC-M.[11] However, the success rate achieved at longer distances with both PTC-M and PTC-A strategies far outperform that of QR. Additionally, PTC performs better than TPOT-RL for all distances. To obtain a clearer picture of the relative advantage of the PTC-based strategies over QR and TPOT-RL in this context, we plot the relative improvement of x_d that is achieved by using the PTC strategies over QR and TPOT-RL, $(x_d^{PTC-Y} - x_d^{QR})/x_d^{QR}$ and $(x_d^{PTC-Y} - x_d^{TPOTRL})/x_d^{TPOT-RL}$, respectively (where, Y can be A for the average capacity heuristic, or M for the minimum capacity heuristic). Figure 4.9 shows the percentage change of x_d achieved by using PTC-M over QR (figure 4.9(a)), PTC-A over QR (figure 4.9(b)), PTC-M over TPOT-RL (figure 4.9(c)), and PTC-A over TPOT-RL (figure 4.9(d)) at increasing values of the minimum hop count between call source and destination nodes in the irregular grid topology of figure 4.2. Each plot in each of these figures is for a different value of the call origination probability. The graphs of figure 4.10 and 4.11 show identical measures using the topologies of figure 4.3(a) and 4.3(b), respectively.

Focusing first on figures 4.9(a) and 4.9(b), it is observed that, for a given call origination probability, when the call destinations are very close to the call sources (i.e., when the "minimum hop count" axis has values of 1 and 2), QR performs slightly better than either PTC-M or PTC-A, indicated by the small negative deviation. For example, at load 0.6, QR achieves about a 5% improvement over PTC-M in connecting calls at nodes 1 hop away (see figure 4.9(a)). However, with increasing distances between the call source and destination nodes (i.e., where the "minimum hop count" is 3 and above), the rate of successful connections is much higher for the PTC-based strategies than QR (the deviation values are positive). For example, in figure 4.9(a), for a load value of 0.6, PTC-M achieves more than 15% improvement over QR in the connection rate of calls that have their destinations at least 8 hops away from the sources. Also, this relative improvement is observed to generally increase with increasing distance. Thus, the PTC information-sharing strategies are more effective in connecting calls for which the source and destination nodes are farther apart. In figures 4.9(c) and 4.9(d), PTC-M and PTC-A perform better than TPOT-RL at all distances and under all load values.

To explain this advantage of PTC, note that although the call connection process is executed in steps by multiple agents who use their individual estimates to forward a call,

[10]The maximum value of the minimum hop count is the property of the corresponding network.

[11]In chapter 3, we showed that the delay associated with distributing information is lower in PTC than in NN. However, the difference reduces as the distance of the information source reduces (the value of k in equation 3.9). This is the reason why the call success rates of PTC and QR (which is based on the NN protocol) are almost similar at very short distances.



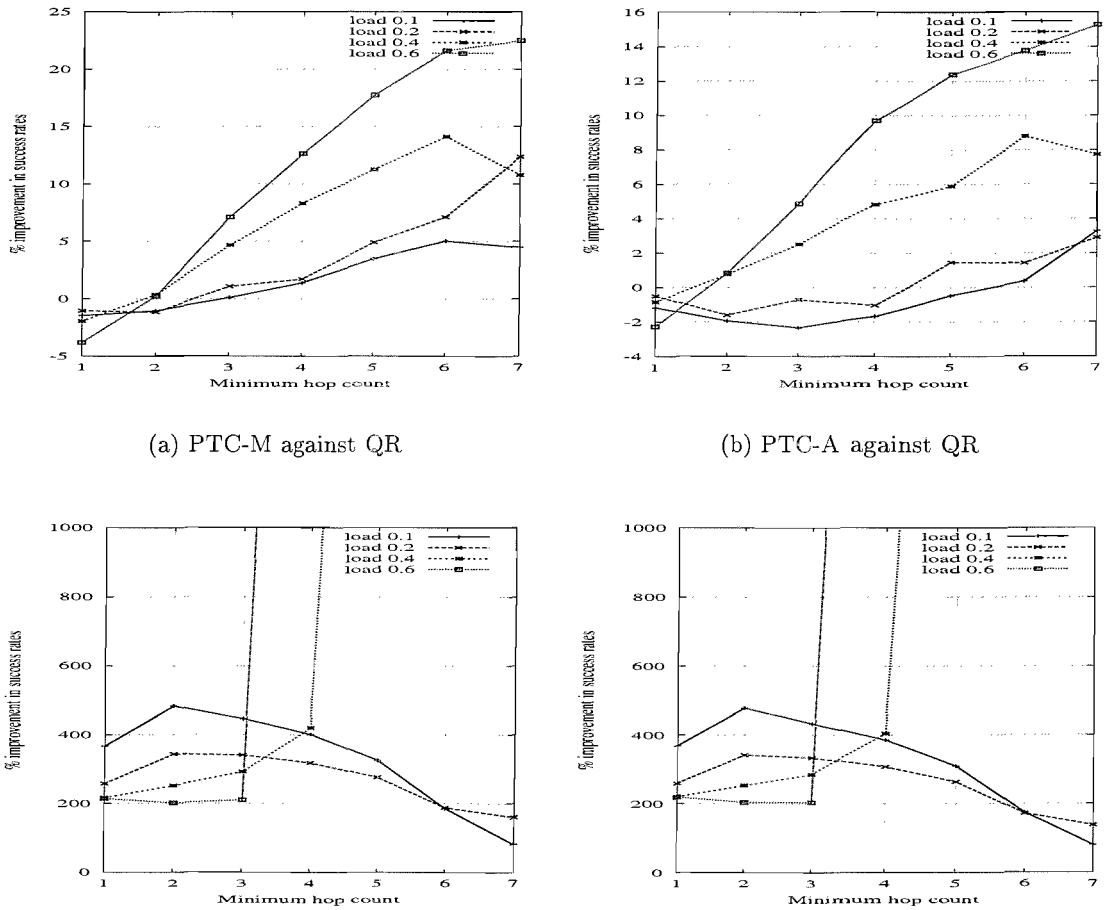
(a) PTC-M against QR

(b) PTC-A against QR

(c) PTC-M against TPOT-RL

(d) PTC-A against TPOT-RL

Figure 4.9: Call success rates at various distances — topology of figure 4.2



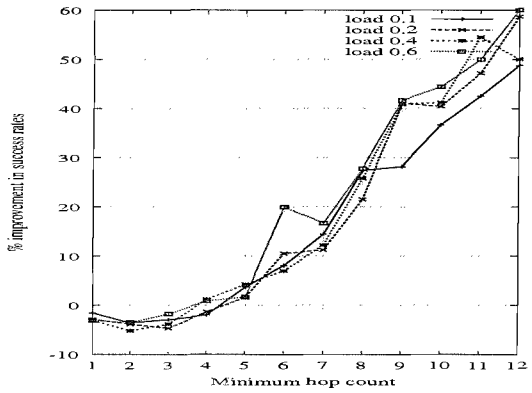
(a) PTC-M against QR

(b) PTC-A against QR

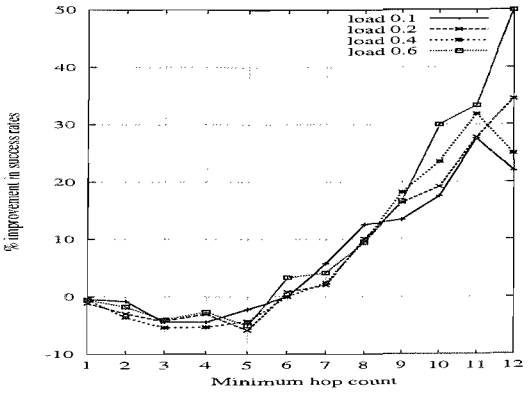
(c) PTC-M against TPOT-RL

(d) PTC-A against TPOT-RL

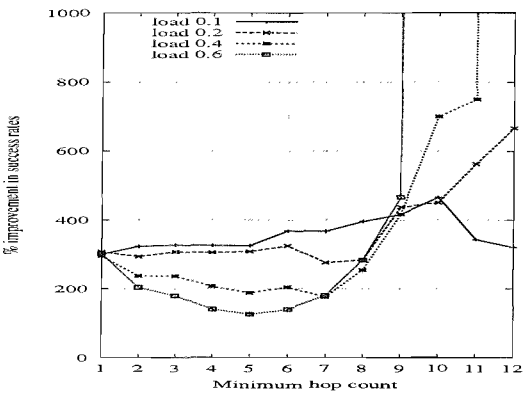
Figure 4.10: Call success rates at various distances — topology of figure 4.3(a)



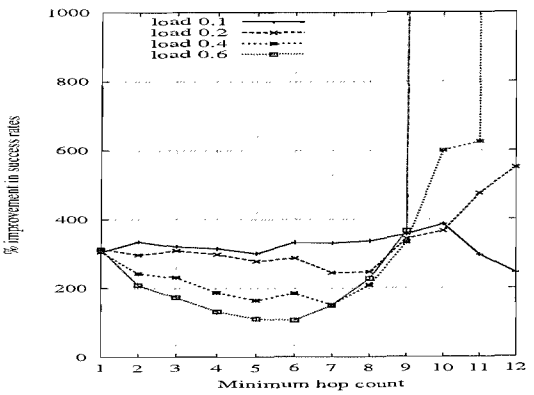
(a) PTC-M against QR



(b) PTC-A against QR



(c) PTC-M against TPOT-RL



(d) PTC-A against TPOT-RL

Figure 4.11: Call success rates at various distances — topology of figure 4.3(b)

the forwarding decisions of the agents who are closer to the call origin are more critical in determining whether it will be successfully routed to the destination. This is because if these agents start forwarding a call in a direction where there is a high bandwidth occupancy on the nodes, then that will be a sub-optimal decision at the beginning of a task execution process. In this case, it is more likely that the call will be forwarded where there is no bandwidth left and, therefore, the forwarding process would terminate (the call would be dropped). With more up-to-date estimates achieved by PTC (as observed in the analysis of section 3.3), an agent is capable of taking better routing decisions in terms of forwarding in the appropriate direction than using QR. This is why we observe higher call success rates in the PTC-based strategies than QR when calls have to be routed at longer distances. Moreover, with increasing distance, successful routing becomes more difficult since the farther the destination the less up-to-date are the estimates. Given this, the increasing (more positive) deviation of the x_d values with increasing distance, as observed in figure 4.9, indicates that PTC is more capable (less affected) than either QR or TPOT-RL to route calls at long distances. PTC-M is more effective in placing long-distance calls than PTC-A although both are better than QR and TPOT-RL. This is evident by the slightly higher positive deviation of PTC-M over QR (figure 4.9(a)) than that of PTC-A over QR (figure 4.9(b)) or by the higher positive deviation of PTC-M over TPOT-RL (figure 4.9(c)) than that of PTC-A over TPOT-RL (figure 4.9(d)) for the same call origination probability.

The above observations from figure 4.9 also hold in figures 4.10 and 4.11. Additionally, in all of these figures, it is observed that, for a given distance d, the deviation of the x_d values are generally higher for a higher call origination probability. To justify this observation, note that the success rate of any given protocol decreases with increasing load (see section 4.5.1 for this result) because with more calls originating, the number of dropped calls increases since the nodes have only a limited amount of call channel bandwidth. Nevertheless, the communication protocol that generates better estimates would enable the agents to cope with the increasing load better by maintaining a higher call success rate. This is indicated by the observation that increased load impacts QR and TPOT-RL more than the PTC-based strategies, i.e., the decrease in the success rate is more in QR or TPOT-RL than in PTC-M or PTC-A with increasing load. Hence, the deviations of the x_d values between PTC and the benchmarks increase with load. For example, in figure 4.9(a), at a hop length of 8, the deviation of x_d is about 2.5% with load 0.1, and more than 15% with load 0.6.

The observations in this section can be summarised as, *the post task-completion heuristics are more capable of connecting calls at longer distances than the benchmark strategies and exhibit increased effectiveness in achieving this against high network loads.* A relative improvement of more than 50% is observed in the rate of successful call connections of PTC-M over QR at distances of 12 hops in a 100-node random graph with

a high network load (see the graph for a load of 0.6 in figure 4.11(a)). For the same parameter values, PTC-M achieves a large improvement over TPOT-RL of more than 1000% (see figure 4.11(c)).

4.5.3 Performance — Information Message Rate

The total number of messages (see section 4.4.1.3 for its definition) transmitted in the entire network is measured every T time steps of a simulation. The average number of such messages per unit time is computed over the steady state. These measurements are repeated for each of the three topologies. Results from constant-load experiments are reported first, followed by those from a dynamically changing load.

4.5.3.1 Constant Load.

Table 4.7 shows the message rate obtained in each of the strategies under steady state conditions in the topology of figure 4.2. Tables 4.8 and 4.9 show the same for the topologies of figure 4.3(a) and figure 4.3(b), respectively. In all of these results, it is observed that for any given call probability, the message rate (under column "Avg") is significantly lower in the PTC strategies than both QR and TPOT-RL. For example, in table 4.7, it is 0.25 for PTC-M, 0.26 in PTC-A, 0.39 for QR, and 0.523 for TPOT-RL with a load level of 0.1.

An additional observation is that the relative reduction of the message rate (column named "% Saving" shows these values) achieved by using PTC becomes more pronounced with increasing load. These values are computed as $(\mid m^{PTCY} - m^{QR} \mid)/m^{QR}$, $(\mid m^{PTCY} - m^{TPOTRL} \mid)/m^{TPOTRL}$, where m represents the average steady state message rate of a given communication protocol for a given load (and Y can be either A or M). For example, in table 4.7, at load 0.1, PTC-M has about 35.6% (=| (0.251 − 0.39)/0.39 |) less message rate than QR and about 52% (=| (0.251 − 0.523)/0.523 |) less than TPOT-RL. However, this saving in the message rate increases to about 80.3% (=| (0.285 − 1.45)/1.45 |) relative to QR and to about 72.8% (=| (0.285 − 1.05)/1.05 |) relative to TPOT-RL when the call probability is 0.6. This is because with increasing network load, the increase in the number of messages in both QR and TPOT-RL is much higher than the increase in the PTC-based strategies (e.g., in table 4.7, the message rate increases from 0.39 to 1.45 in QR — a 272% increase, from 0.523 to 1.05 in TPOT-RL — a 101% increase, and from 0.251 to 0.285 in PTC-M — only a 13.5% increase — as the load increases from 0.1 to 0.6).

Note that in QR, a node transmits a new information message to the forwarding (upstream) agent at each step of the call forwarding process. Thus, with increased load, this algorithm incurs a large increase in the number of messages because there are many

Table 4.7: Information message rates for all strategies — topology of figure 4.2

Load	Strategies								% Saving			
	QR		PTC-A		PTC-M		TPOT-RL		PTC-A / QR	PTC-M / QR	PTC-A / TPOT-RL	PTC-M / TPOT-RL
	Avg	Stdev	Avg	Stdev	Avg	Stdev	Avg	Stdev				
0.1	0.39	0.0048	0.26	0.0019	**0.251**	0.0019	0.523	0.0023	33.33	35.64	50.28	52.00
0.2	0.66	0.0079	0.281	0.002	**0.275**	0.0022	0.683	0.0029	57.42	58.33	58.85	59.73
0.4	1.1	0.01	0.289	0.0018	**0.284**	0.0016	0.891	0.0036	73.73	74.18	67.56	68.12
0.6	1.45	0.0125	0.2894	0.0017	**0.285**	0.0015	1.05	0.0077	80.04	80.34	72.43	72.85

Table 4.8: Information message rates for all strategies — topology of figure 4.3(a)

Load	Strategies								% Saving			
	QR		PTC-A		PTC-M		TPOT-RL		PTC-A / QR	PTC-M / QR	PTC-A / TPOT-RL	PTC-M / TPOT-RL
	Avg	Stdev	Avg	Stdev	Avg	Stdev	Avg	Stdev				
0.1	0.33	0.0023	0.23	0.0012	**0.23**	0.0014	0.759	0.0054	30.30	30.30	69.69	69.69
0.2	0.55	0.0048	0.26	0.0013	**0.25**	0.0012	1.42	0.0074	52.73	54.55	81.69	82.39
0.4	0.9	0.0049	0.27	0.001	**0.26**	0.0009	2.163	0.0091	70.0	71.11	87.51	87.97
0.6	1.2	0.0054	0.272	0.0007	**0.267**	0.0012	2.70	0.0106	77.33	77.75	89.92	90.11

more calls to be routed. In TPOT-RL, a node forwards its state information while routing a call. So, although reward distribution occurs in TPOT-RL every `update-interval` time steps, the state information messages propagated during call routing contribute toward the message rate in the system. So, when larger numbers of calls need to be routed at high load values, the number of such messages increases. The PTC-based strategies, on the other hand, attain a significant saving in the message rate by delaying the information transmission until a call connects and then transmitting a single message from the destination to the source node (with only updating the reward information at each step of the message propagation). This prevents any information exchange from occurring for calls that fail to connect. Both QR and TPOT-RL, on the other hand, would still incur the messaging cost even if a call finally fails to connect. In addition, the PTC-based strategies save some messages from being exchanged that QR and TPOT-RL incur on the loops of a call route. Because the agents using PTC transmit messages only after a call is connected, and the loops on call routes are dropped (as a call is forwarded), messages are prevented from being transmitted on loop portions.

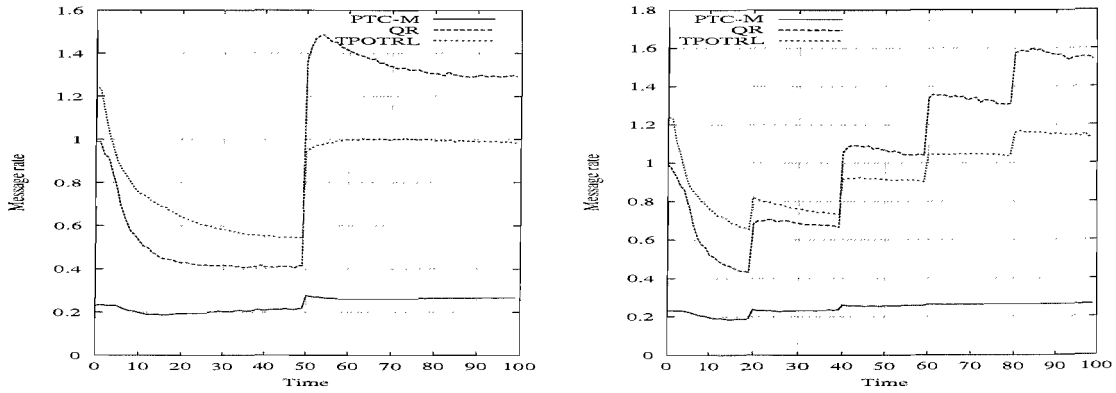
4.5.3.2 Dynamically Changing Load.

Similar to the call success rate measurements of section 4.5.1, the time-variation of the information message rates are recorded for all strategies when the network load fluctuates

Table 4.9: Information message rates for all strategies — topology of figure 4.3(b)

Load	Strategies								% Saving			
	QR		PTC-A		PTC-M		TPOT-RL		PTC-A / QR	PTC-M / QR	PTC-A / TPOT-RL	PTC-M / TPOT-RL
	Avg	Stdev	Avg	Stdev	Avg	Stdev	Avg	Stdev				
0.1	0.58	0.0117	0.214	0.0009	**0.21**	0.0014	1.42	0.018	63.10	63.79	84.92	85.21
0.2	0.83	0.0112	0.235	0.001	**0.229**	0.001	2.19	0.0091	71.69	72.41	89.27	89.54
0.4	1.26	0.0098	0.249	0.001	**0.241**	0.0008	2.98	0.0061	80.24	80.87	91.64	91.91
0.6	1.63	0.0095	0.253	0.0007	**0.245**	0.0008	3.37	0.0079	84.48	84.97	92.49	92.72

dynamically in a simulation run.

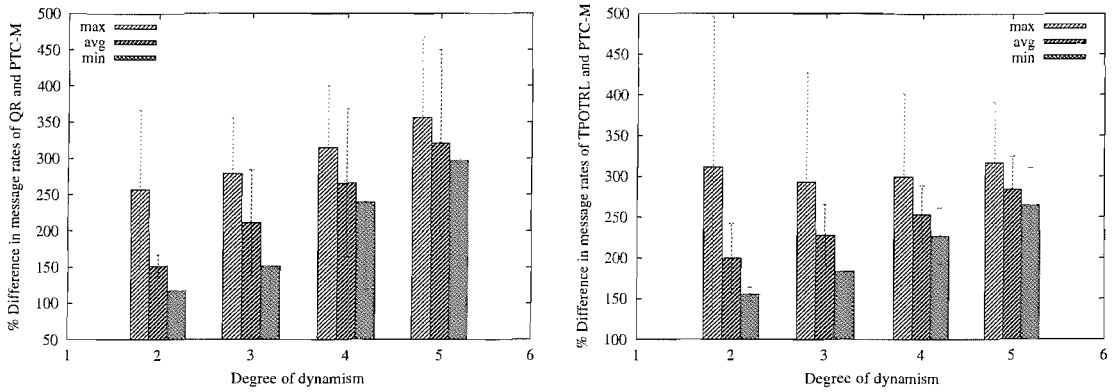


(a) With 2 degrees of dynamism

(b) With 5 degrees of dynamism

Figure 4.12: Time variation of message rates of QR, PTC-M, and TPOT-RL with network load fluctuations — topology of figure 4.2

Figure 4.12(a) shows the message rate variations of PTC-M, QR, and TPOT-RL when the load level changes from 0.1 to 0.6 in a simulation run using the topology of figure 4.2. Figure 4.12(b) shows the same when the load is varied as 0.1, 0.2, 0.4, 0.6, and 0.8 after equal intervals of time in the course of a simulation run. Both of these figures show that PTC-M has a significantly lower message rate than both QR and TPOT-RL under fluctuating load conditions. Further, as the network load increases dynamically, both QR and TPOT-RL incur a large increase in the message rates while the increase in PTC is insignificant. In both QR and TPOT-RL, information is transmitted during call setup (although reward updates occur in TPOT-RL after every `update-interval` time steps). Hence, as more calls originate with increasing network load, the number of such information propagations increase in these strategies. It is also observed that with increasing load, TPOT-RL has a lower message rate than QR. We identify that this relative advantage of TPOT-RL against QR is due to its poorer call success rate than QR (reported in section 4.5.1). Since TPOT-RL is less efficient than QR in connecting



(a) QR compared to PTC-M

(b) TPOT-RL compared to PTC-M

Figure 4.13: Summary statistics of message rate differences between QR, TPOT-RL, and PTC-M with network load fluctuations — topology of figure 4.2

calls (this implies that in TPOT-RL a large number of call attempts are unsuccessful), and since both propagate information while call forwarding, TPOT-RL does not incur as much of an increase in the number of messages as incurred by QR with increasing load.

In a way similar to the success rate results with fluctuating load, the summary of the differences in message rates between QR and PTC-M and between TPOT-RL and PTC-M are presented in figures 4.13(a) and 4.13(b), respectively. In both of these figures, the degree of dynamism is plotted along the horizontal axis while the percentage increase of message rate using QR or TPOT-RL against PTC-M is plotted along the vertical axis. As observed in the success rate results of figure 4.8, in both figure 4.13(a) and 4.13(b), the gap between the minimum and the maximum differences of message rates reduce with increasing degrees of freedom. For instance, in figure 4.13(a), this gap is about 140% with 2 degrees of freedom, while it is 60% with 5 degrees of freedom. Nevertheless, the mean difference between the message rates of PTC-M and QR or that between PTC-M and TPOT-RL increases with increasing degrees of dynamism. For example, the mean difference increases from 151% to 321% in figure 4.13(a), while it increases from 200% to 284% in figure 4.13(b). This indicates the advantage of PTC in terms of maintaining a limited number of message overhead compared to both QR and TPOT-RL.

From these observations, it can be concluded that the *post task-completion heuristics not only achieve a higher call success rate (section 4.5.1) and higher effectiveness in connecting calls at longer distances under high loads (section 4.5.2) than the benchmark strategies, but also achieve these at the expense of a significantly lower rate of messages under both static and dynamically changing load conditions.* For example, PTC-M

achieves an 80% saving in message rate compared to QR, and a 72% saving compared to TPOT-RL in the grid topology under high network load. Further, in a dynamic load setting, with five changes in the network load, PTC-M saves about 320% in message rate than QR and about 284% than TPOT-RL. Thus, PTC is shown to be a *more efficient protocol toward developing a cooperative MAS for the sequential RA problem.*

4.6 Summary

This chapter develops a set of communication heuristics based on the PTC protocol (introduced in chapter 3) that can be used by network nodes to share information and ensure effective call routing (a sequential RA task in a distributed environment). This chapter also describes the implementations of the benchmark algorithms, Q-routing and TPOT-RL, against which the PTC-based heuristics are compared. The simulation environment is described in detail along with the performance measures used for empirical evaluation. Finally, a detailed analysis of the experimental evaluation of the communication strategies under a wide variety of environmental settings is presented. These results provide substantial evidence that PTC is a better information-sharing protocol than the benchmark algorithms. Hence, this empirical study further strengthens the analysis of chapter 3 that had theoretically established the superior performance of PTC. This, in turn, substantiates the theoretical justification of our thesis done in chapter 3. Therefore, it implies that the PTC information-sharing mechanism does allow the agents to generate good quality state estimates and effectively perform sequential RA tasks in distributed domains.

Now we aim to subject PTC to further empirical tests using environments that exhibit more dynamism than that studied in this chapter. In doing so, our goal is to further validate the usefulness of PTC as a mechanism that can indeed be used to generate highly adaptive behaviour in the context of sequential RA tasks — a claim of our thesis, stated in chapter 1. In particular, such dynamism is simulated in the network call routing problem by means of network failures.

Chapter 5

Adaptiveness in Highly Dynamic Environments

In an attempt to establish our thesis, identified in chapter 1, chapter 3 developed the novel PTC protocol for sharing information and demonstrated theoretically that it can be used as a mechanism for generating good quality state estimates; such estimates being useful for effectively solving sequential RA tasks in distributed domains. These theoretical claims were then further substantiated using empirical analyses in chapter 4, where communication heuristics designed using PTC were shown to perform better in a sequential RA task domain (call routing in telephone networks) than a host of benchmark algorithms. Now, in this chapter, we focus on using PTC-based communication heuristics to generate effective adaptive behaviour in highly dynamic environments. Such adaptiveness is of significant importance in dynamic systems where all possible environment changes cannot be predicted a priori. With this capability, the agents would be able to effectively respond to unforeseen changes in order to successfully generate quality solutions to tasks — a claim made in our thesis in chapter 1. Therefore, by subjecting PTC to empirical evaluation in dynamic environments, we attempt to further establish the thesis.

Against this background, we note that adaptive behaviour was already demonstrated by our communication heuristics in chapter 4 in response to some degree of system dynamism — that of changing load. In this chapter, however, we aim to achieve effective adaptiveness against dynamism caused by drastic events such as a failure of a task processing episode. In our domain of interest, such an event can occur due to network failures. In particular, failures are non-trivial phenomena in wireless mesh networks used for telephony (Chandler et al., 1993). Hence, in this chapter, failures are considered to be inevitable in the network domain and a careful investigation is carried out about how the PTC framework can be extended such that the network can effectively adapt to failures.

In more detail, failures of network nodes, caused due to various reasons like hardware failures, environmental hazards, power outage, and so on, are a significant cause for concern in the type of networks we are investigating (Akyildiz et al., 2002). Such incidents are non-deterministic in nature and typically very hard to predict and, therefore, problematic by causing sudden and serious disruptions in network services. Moreover, in decentralised networks with no single monitoring and management centre (which is an undesirable choice because it is a non-scalable solution and a single point of failure), it is of critical importance to design strategies to counteract the effects of such failures efficiently and effectively. Without this, failures, which effectively result in an alteration of the network topology by making a set of network paths unavailable for routing (thus, inducing dynamism in the environment), would cause the call success rate to plummet. This happens since routing decisions taken by the nodes in such decentralised systems are based on their individually estimated routing tables (refer to section 4.2 for a discussion on the learning and routing mechanisms). With failures occurring, therefore, unless the nodes are updated with the resulting network state changes, the information in the routing tables would effectively be out-of-date with respect to the actual network condition. Thus, routing decisions taken would be highly sub-optimal leading to performance degradation.

Against this background, network failures are a significant problem in the given application area and one that requires to be counteracted. Given this, the following objectives need to be met:

Adapting routing. Nodes should be able to effectively update their routing tables (essentially, the Q-tables) to capture the changes caused by failures. This would ensure that they can take the correct routing decisions in the context of failures occurring.

Diagnose failure. It would be of significant benefit for automating the management of decentralised networks if nodes can actually detect failures.[1] Moreover, this capability can then be used for appropriate recovery actions to be summoned (for example, assign support engineers to recover a failed node, or attempt to re-route any ongoing calls that are affected by the failure).

As it stands, however, the PTC heuristics described in chapter 4 are not capable of achieving the above-mentioned objectives (the reasons are discussed shortly). Therefore, this chapter focuses on extending the previous PTC heuristics by *introducing additional information-sharing*. Specifically, it is demonstrated by empirical studies and a theoretical analysis that this modification is successful in addressing the problems identified

[1] By *detection*, it is implied that a node can identify that another node has failed and, therefore, can raise an alarm. More details follow in section 5.6.

with network failures. It is observed, however, that these advantages are attained at the cost of generating larger messages than those in chapter 4.

In the remainder of this chapter, section 5.1 describes the failure model we adopt and how, after a failure occurs, another node can identify this and initiate transmitting information. Section 5.2 explains why and how the previous PTC heuristics need to be extended to achieve the above-mentioned objectives when failures occur. Section 5.3 describes the extended PTC heuristic (henceforth, labelled as e-PTC) in the network call routing scenario. Section 5.4 presents the performance measures used for empirical evaluation of the e-PTC heuristic in the example application. Then, section 5.5 presents experimental results to demonstrate the advantage of this extended heuristic in improving routing performance with failures. Subsequently, section 5.6 presents how distributed failure diagnosis is achieved with the e-PTC heuristic by designing a detection algorithm (section 5.6.2) and then applying it to a simulation-based study (section 5.7) to demonstrate its capabilities. Finally, section 5.8 presents concluding remarks on the contributions of this chapter.

5.1 The Network Failure Model

In this section, we outline our network failure model and the related assumptions. In so doing, we identify two distinct functional parts in such a model. These are: (i) the characterisation of what defines a failure and (ii) how such a failure is detected. The second part is important since it details our assumptions regarding how a failure is actually detected by another node, which is when the e-PTC protocol is initiated. The following subsections describe each of these.

5.1.1 Failure Characterisation

By network failures, we imply failures of network nodes. In particular, only **stop** failures are considered. This is because such failures are the most common type studied in distributed systems (Lynch, 1996) and, hence, form a good experimental testbed for measuring the performance of e-PTC. We assume that a node failure is associated with the following changes in its activities:

- It cannot run any processes after failure.
- It cannot communicate with other nodes after failure.

Thus, after a failure, all ongoing calls on this node will be lost since there would not be any process available on the node to handle this call. Moreover, it will not be able

to detect or acknowledge a request to forward calls from its neighbours. Thus, any new attempts to place calls through this node will fail too. In addition, we assume that once a node fails, it continues to remain so throughout our simulations.

5.1.2 Detection Characterisation

After a node fails, one of its neighbours actually detects that it has failed when the latter makes a request to forward a call. The requesting node detects the failed status of the requested node since it does not receive any response from the latter (the latter cannot communicate after failure).[2] It is at this time when the requesting node de-allocates its reserved bandwidth for the ongoing calls that were being routed via the failed node. It also does the same for the call it was trying to set up. Furthermore, it generates `drop` messages to inform its neighbouring nodes to de-allocate bandwidth for the affected calls.

The above description is clarified in figure 5.1 where node f has failed. One of its neighbours, i, detects that f has failed. Thus, it deallocates its bandwidth for all calls that were being routed through both i and f, and that for the new call it was trying to set up. Also, it informs each j (a neighbour of i that is also on the path of calls via i and f) by sending a `drop` message to de-allocate bandwidth. Each such j continues to propagate the `drop` message until the terminal node on that path is reached. Note that, a similar process is undertaken by each neighbour of f whenever they detect that f has failed. We assume that call requests, and, thus detections, occur fairly frequently to ensure that, post failure, the reserved bandwidth units for the lost calls are de-allocated efficiently.

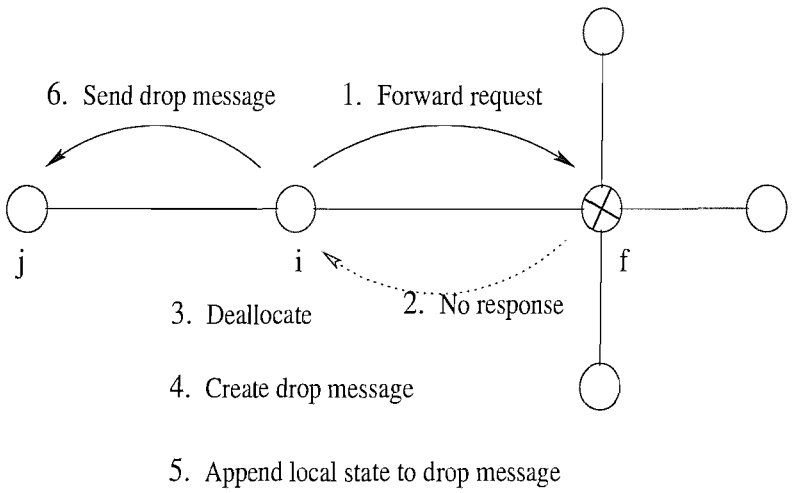


Figure 5.1: Failure Detection Model Schematic

[2]We do not assume slow communication to be causing a delay in receiving responses which may trigger an incorrect detection.

5.2 Extending PTC to Deal With Failures

In this section, we describe the extension made to the PTC heuristics of chapter 4 and explain why it is capable of generating adaptiveness in dynamic conditions. In chapter 4, the PTC heuristics allow the nodes to transmit their local state knowledge to others *only after a call successfully connects at its destination node.* Thus, after each successful call connection, a node along the path of the call gets updated about the states of the downstream nodes along the path. However, if a node fails, call setup attempts via this node would also fail. This is because a call forwarding request to a failed node would not get a response and, hence, the forwarding process along that path would stop. In case of a call failure, the previous PTC heuristic transmitted a `drop` message (refer to section 4.2 for details) to free up the pre-allocated bandwidth units on the nodes along the partially completed call path.[3] Given this mechanism, where information is shared only for successful calls, the information about failures remains hidden from the nodes. This is because the `drop` message simply causes a bandwidth unit to be freed, but does not affect a node's Q-estimates in any way. Therefore, the Q-table retains the previous information. Hence, on a subsequent attempt to forward a call, a node would still use the same outdated Q-estimates (possibly causing the call attempt to fail again). Moreover, after a successful call connection, when the `ack` message is carrying state information upstream (as in chapter 4), if an intermediate node fails, the propagation of the `ack` message is stopped and the remaining path nodes do not get updated about the state information.

Against this background, to alleviate the limitation of the previous PTC heuristic in not being able to inform nodes about failures, `drop` *messages are used to transmit the local state values of nodes in addition to the* `ack` *messages.* This is the simple yet fundamental extension to the PTC heuristics of chapter 4 that is made to achieve the required adaptiveness in highly dynamic environments and that defines e-PTC.

In more detail, in e-PTC, each node along the path of a `drop` message appends its local state value to the message just like they do for an `ack` message. Moreover, similar to the Q-update mechanism of section 4.2 using `ack` messages, in e-PTC information-sharing, the nodes update their Q-estimates using the state information contained in `drop` messages. In this manner, therefore, the nodes would be able to remain updated about the bandwidth availability changes caused due to a node failure. Such capability would, in turn, allow the nodes to adapt to failures more effectively than that achieved in chapter 4. The following section describes how we have used this e-PTC information-sharing protocol in our network call routing simulations.

[3]Note, however, only new call attempts failed (and not ongoing calls) in chapter 4 due to node saturation and not due to node failures.

5.3 Extended PTC Information-Sharing in Network Call Routing

As explained in section 5.1, after a failure, the node becomes inaccessible to others which effectively changes the network topology by removing the set of paths containing it. The way e-PTC is implemented when existing calls and those being attempted are affected by a node failure is explained in the following.

Referring to figure 5.1, consider an ongoing call that is routed through i and f when f fails (note that this description is generic and applies equally to all such ongoing calls). After this happens, i detects the failure as explained in section 5.1. Then, it generates a `drop` message, appends its state value s_i, and transmits to j. Node j, in turn, appends its state value s_j to this message and transmits to its neighbour on this path. This process continues until the terminal node on this path is reached.[4] While the `drop` message is transmitted, each node receives the state values of the nodes between itself and the failed node f. Thus, in a way similar to the Q-update mechanism (using only `ack` messages) described in section 4.2, each node receiving the `drop` message, updates their Q-estimates. Thus, this additional communication over that of the PTC heuristics of chapter 4 characterises e-PTC.

Now, consider the case of a new call being set up. Thus, when node i requests node f, it does not receive an acknowledgement from f. As explained in the failure model earlier, it is at this time that i detects that f has failed. So, it generates a `drop` message and transmits to the immediate "upstream" node on the path along which the new call was being set up. The processes of generating `drop` messages, appending state values to these messages, transmitting them upstream, and updating Q-estimates are exactly similar to what has been explained above.

Thus, by the above-mentioned procedures, state information distribution after node failures provides the basis for a mechanism by which the nodes that receive such information can effectively update their Q-estimates. Moreover, this mechanism also provides a means for the nodes receiving such information to detect failures. For example, in figure 5.1, all nodes along all call paths that have f get the state information of f which they can use to diagnose its state. In this context, when a node neighbouring the failed node transmits state information along with the `drop` message after unsuccessfully attempting to contact the latter, it assigns a state of zero for the latter. When another node receives this information, a state value of zero can be used to interpret a node failure. However, there is an important issue that needs to be resolved before correct

[4]Note that if the node transmitting a `drop` message fails, the `drop` message will be lost. However, in this case, a new failure detection and `drop` message propagation would be initiated. This will be able to de-allocate the bandwidth reserved for the calls via the previously failed node but which had not been de-allocated since the corresponding `drop` message terminated.

interpretation is possible. This is, when a node's bandwidth is completely occupied, it rejects a forwarding request and, therefore, the call attempt fails. Similar to a failure, the state value (of the node where the call failed) transmitted in this case is also zero. Therefore, correct failure diagnosis requires distinguishing between the information (zero) received from a failed and a saturated (also zero) node. The key to do this is to observe a *sequence* of state values.[5] Note, to indicate failure, a separate message (e.g., `fail`) could be used instead of a zero state. However, the state values are used to update the Q-estimates that indicate summary bandwidth availability along network paths. Thus, a separate message to explicitly indicate failure would not be useful for bandwidth estimation and it would also increase the bandwidth usage (thereby increasing the communication overhead of the network). In fact, as will be shown shortly, the zero state values can indeed be used for failure detection, in addition to bandwidth estimation. This is done with the help of an algorithm that computes the *number of consecutive state values* an agent needs to monitor from another node to achieve the correct diagnosis. Specifically, section 5.6.2 describes this algorithm which has two important features; it is (i) without false positives — detection is made only if a failure has occurred, and (ii) without false negatives — detection is guaranteed once a failure occurs.

5.4 Bandwidth Allocation Using e-PTC

In this section, e-PTC, as described in section 5.3, is applied to the network routing problem. In particular, its performance is compared against that of the PTC-based communication heuristics of chapter 4 when node failures occur. As in the experiments of chapter 4, the call success rate and message overhead are used as the measures of performance. Moreover, to compare the performances of the information-sharing strategies against failures, an additional measure is chosen: recovery of call success rate. The following describes these performance measures in detail:

Call Success Rate(CSR): As is done in the experimental evaluation of chapter 4, the average CSR is chosen as the metric to measure the bandwidth allocation quality achieved by a given information-sharing protocol. Thus, the more effective the bandwidth allocation, the higher the CSR. So, if e-PTC achieves a higher CSR than PTC under a given setting, then it can be concluded that e-PTC is a more effective protocol than PTC in allocating node bandwidth under the given environmental conditions. To measure CSR, if K calls originate in the network in a given time interval, out of which k are successfully connected, then CSR within that period is given by k/K.

[5]The justification in doing this is that when a saturated node becomes unsaturated, a non-zero state value would be received from this node. A failed node, on the other hand, would remain failed and thus, would not transmit a non-zero state value again. Sections 5.6 elaborates on this further.

CSR Recovery: When a network failure occurs, the CSR would necessarily drop since a set of network paths would be unavailable and, therefore, call attempts via these paths would fail. However, with the agents attempting to estimate the changes and find alternative routes, the CSR should recover from the initial drop. This is measured by comparing the CSR (denoted as r_{fail}) when a node fails in the simulation (the "test" case) to the CSR (r_{miss}) when the corresponding node is removed from the topology from the start of the simulation (the "baseline" case) keeping all other parameters the same. In the baseline, the agents learn routing policies based on the topology that results after failure in the test simulation. If all agents had global information, then, after failure, r_{fail} could recover and become equal to r_{miss}. But, the agents do not have global information. Also, the network state before the failure in the test would be different from that of the baseline at the same time because the baseline has a different topology with one less node. After a failure, therefore, r_{fail} would not be able to recover to exactly r_{miss}. However, the closer and faster it can get to r_{miss}, the more efficient the protocol. The CSR recovery of e-PTC is therefore compared against that of PTC by comparing the CSRs of both e-PTC and PTC-based test simulations to that of a baseline with PTC.

Message Size (MS): Any advantage that e-PTC has over PTC in terms of superior bandwidth allocation quality is due to sharing more information. Note that one message is transmitted for a call success (`ack`) and one for a call failure (`drop`) in both e-PTC and PTC.[6] However, these messages may not be of the same size in e-PTC and PTC since MS is determined by the number of state values appended to it. For example, in a `drop` message, PTC does not append any state information but e-PTC does. So, the size of these messages in e-PTC would be larger than in PTC. Specifically, the average MS of k messages (including both `ack` and `drop` messages) is $\sum_{i=1}^{k}(l_i)/k$, where l_i is the size of the i^{th} message. The average MS values of the two strategies are compared across various experimental settings to assess the message overheads of e-PTC and PTC. In this regard, it is important to clarify the following. In our simulations, although there are messages of types other than `ack` and `drop`, we consider only these two since they are the only ones that contain state information (thus, are of a non-trivial size) and directly impact estimate learning which, in turn, affects the call routing performance. However, we use a separate control channel (refer to section 4.1 for a listing of network properties) for all such information-bearing messages. *Message size* on its own, therefore, would not impact call success rate in our simulations. Nevertheless, message size is still an important measure for performance since larger messages imply larger bandwidth requirement which conflicts with the property of bandwidth scarcity (an expensive resource) in mesh networks. The objective, therefore, is to minimise the size of the `ack` and `drop` messages

[6]This is why *message rate* is not used as a performance measure as done in chapter 4. The message rates of e-PTC and PTC are the same in a given experiment. Instead, it is the *message size* that is different in these heuristics and, hence, is chosen as the performance metric.

as much as possible.

5.5 Results and Analysis

The topologies of figures 5.2 and 5.3 (reproduced from chapter 4) are used for the experiments reported in this section.[7] Also, as is done in the experimental study of chapter 4, evaluation is done by varying the call origination probability to test the impact of load on the system performance. All other parameters are kept at the same values as in chapter 4.

In what follows, two broad classes of experiments are presented. First, the CSR and MS of e-PTC and PTC are compared in the absence of failures (section 5.5.1). These experiments test if e-PTC has any inherent advantage (higher CSR) or overhead (higher MS). Second, experiments are designed with failures (section 5.5.2). These results specifically highlight the advantages that e-PTC has in coping with failures over PTC. Note, in chapter 4, it has already been shown that PTC outperforms a range of common benchmark algorithms in this area. Hence, any improvement in e-PTC would also indicate an improvement over the current state of the art.

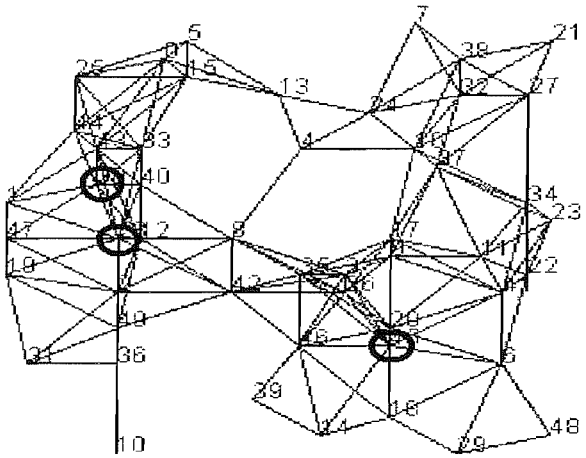


Figure 5.2: A 50-node network topology

5.5.1 Without Failures

In this section, the call success rates and message sizes of e-PTC and PTC are evaluated and compared when there are no network failures.

[7] As done in the previous chapter, we have performed the experiments on various other topologies and observed the same broad trends in performance. So, we report our results on only a sample subset of the topologies used.

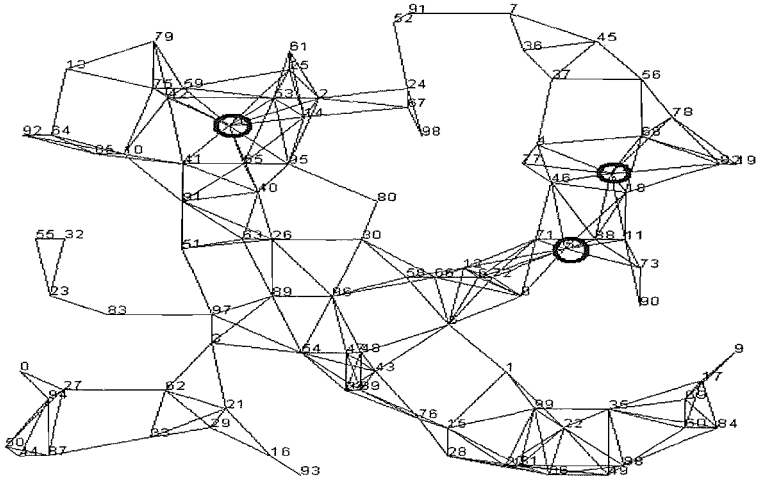


Figure 5.3: A 100-node network topology

5.5.1.1 Call Success Rate.

The relative differences in the CSR values of e-PTC and PTC are computed as (CSR(ePTC)−CSR(PTC))/CSR(PTC), where CSR(x) is the time-varying average call success rate of protocol x. Thus, the relative CSR difference is also a function of time. Figure 5.4 shows the time-varying relative CSR difference values for different call origination probabilities when the topology of figure 5.2 is used. Although e-PTC is designed to deal with failures, these results show that *even without failures, e-PTC has a higher CSR than PTC*: in figure 5.4, at load 0.8, e-PTC achieves a peak 7% CSR increase, while in figure 5.5 (showing the same results when the topology of figure 5.3 is used), for the same load value, the peak improvement of e-PTC CSR is about 11%. Without node failures, calls fail only due to node saturation which obviously happens more at higher loads. Thus, the absolute CSRs of both PTC and e-PTC decrease with increasing load. Now, e-PTC shares information after call failures. Thus, at higher loads and hence, whenever there are more call failures, e-PTC shares more information and lets the agents remain more up-to-date about their bandwidth estimates. Thus, the e-PTC CSR deteriorates less than PTC. Hence, the relative improvement of e-PTC over PTC improves with increasing load. This shows that e-PTC is inherently a more robust protocol than PTC.

5.5.1.2 Message Size.

The time-varying MS values of e-PTC and PTC are compared in a way analogous to the comparison of the CSR values of e-PTC and PTC shown previously. Specifically, figure 5.6 shows the relative increase in the average MS of e-PTC compared to that of PTC when the topology of figure 5.2 is used. It shows that, the higher the load, the larger is the e-PTC MS compared to PTC. Thus, in figure 5.6, at the end of a simulation,

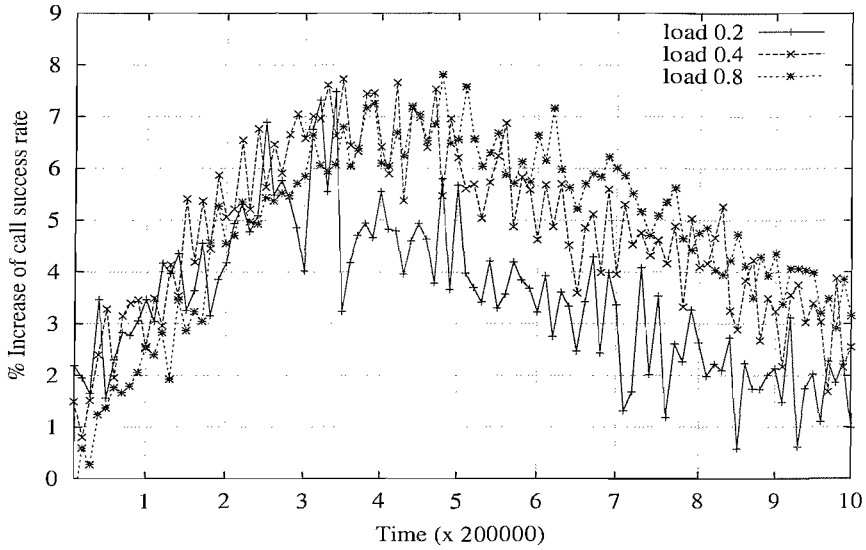


Figure 5.4: Percentage call success rate increase in e-PTC (no failures) on the 50-node topology of figure 5.2

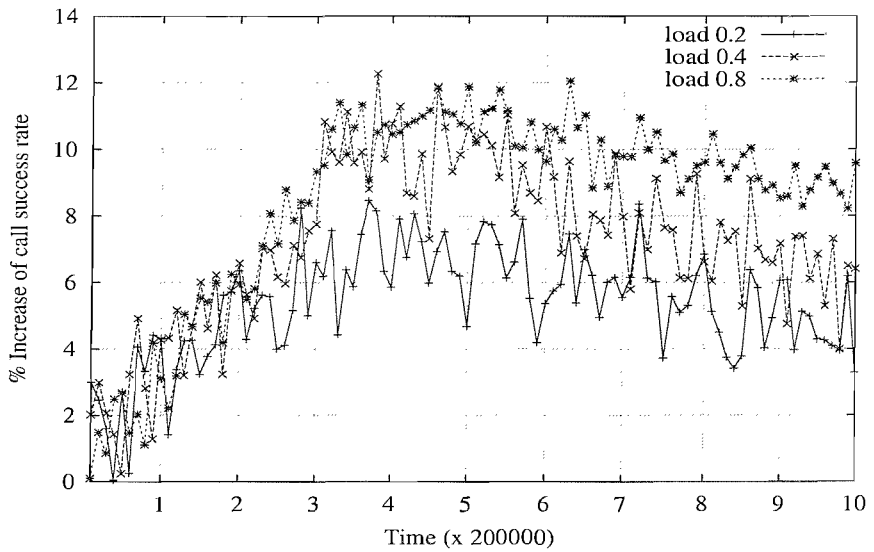


Figure 5.5: Percentage call success rate increase in e-PTC (no failures) on the 100-node topology of figure 5.3

when the load is 0.2, there is a 130% increase in the e-PTC MS over that of PTC (which corresponds to a 2.3 times increase), and at load 0.8, a 290% increase over PTC (a 3.9 times increase). Since e-PTC appends state values to `drop` messages, the number of which increases with load since more calls fail (explained previously in the CSR results), the average MS increases in e-PTC compared to PTC.

As mentioned earlier, since the `drop` and `ack` messages contain state information, they contribute significantly towards congesting the control channel which is used for transmitting these messages. Therefore, an increase of the average size of these messages

in e-PTC implies an increase in the occupancy of the control channel. For example, in this topology, using a load value of 0.8, we measured that the `drop` and `ack` messages in e-PTC occupy approximately 25% of the total control channel size. Therefore, in e-PTC, at a load of 0.8, a 3.9 times increase in the size of these messages over PTC (as shown above) indicates that the control channel occupancy has increased by approximately 18% over PTC. In this context, note that the call channel used for handling actual calls, is of fixed length (maximum size 10). It is much smaller than the average length of the control channel where all the various control messages are queued. Thus, the call channel puts a major constraint, in terms of bounded resource availability, on the call success rate. An 11% increase in the occupancy of the call channel, as shown in the CSR results in section 5.5.1.1, is, therefore, a more significant result than the increase in the control channel occupancy observed here. It shows that e-PTC is able to exploit the bounded resource availability more efficiently than PTC. Our objective, however, is to restrict the size of the `ack` and `drop` messages so that the demand on the control channel is reduced. A mechanism for achieving just this is studied in chapter 6.

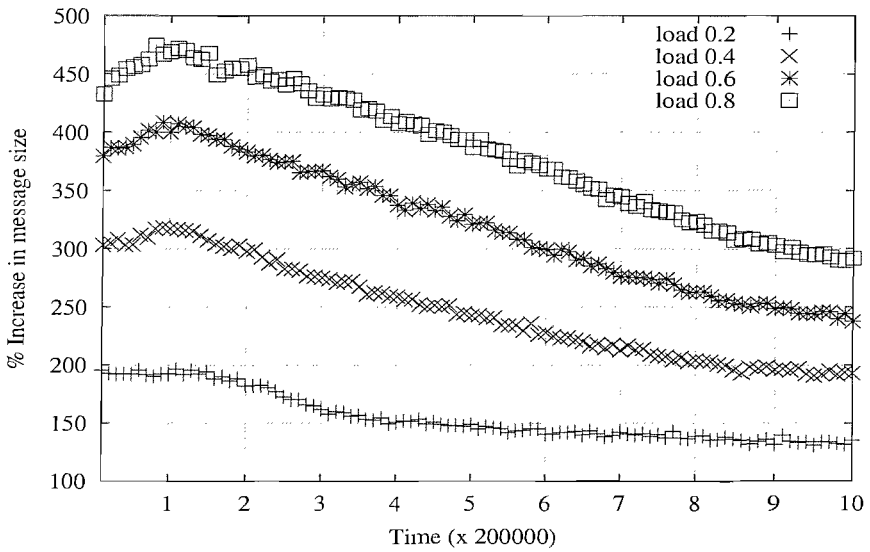


Figure 5.6: Percentage message size increase in e-PTC (no failures) on the 50-node topology of figure 5.2

In figure 5.7 we observe trends similar to those in figure 5.6 where the analogous results are plotted using the topology of figure 5.3. Thus, it can be concluded that e-PTC attains a higher CSR at a cost of larger MS.

5.5.2 With Failures

The way node failures are simulated is as follows. One node fails at a given time and multiple failures occur during a simulation run. Different combinations of nodes are selected to fail in different experiments to generalise across various network failure

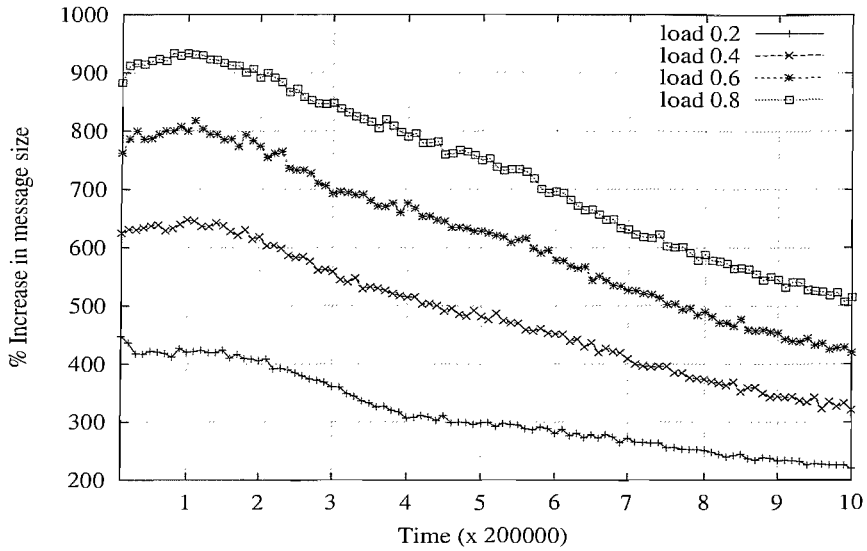


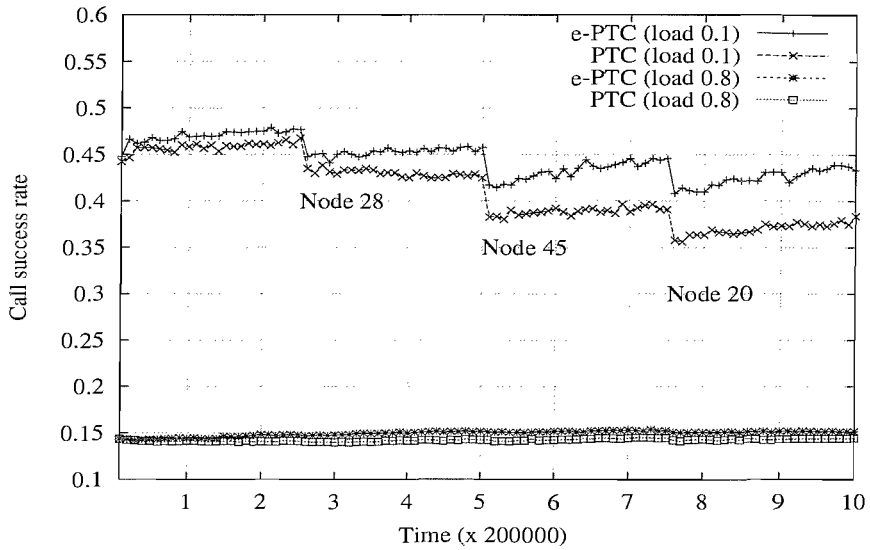
Figure 5.7: Percentage message size increase in e-PTC (no failures) on the 100-node topology of figure 5.3

scenarios. Such a regular failure interval was chosen because it simplifies our analysis. Arbitrary failure patterns would impair a clear understanding of how e-PTC differs from PTC by impacting their performances in complex ways. Thus, we use failures of: (i) three nodes with the highest edge connectivity (encircled in figures 5.2 and 5.3), and (ii) three randomly chosen nodes with average edge connectivity. Nodes with the most connectivity are critical points and heavily used for routing. Thus, (i) provides a "hard" test (since their failures cause a major disruption in the network by removing a large set of paths) for any advantage that e-PTC has over PTC. However, (ii) is a more typical scenario. Here, results are reported using (i) since those from (ii) show identical trends.

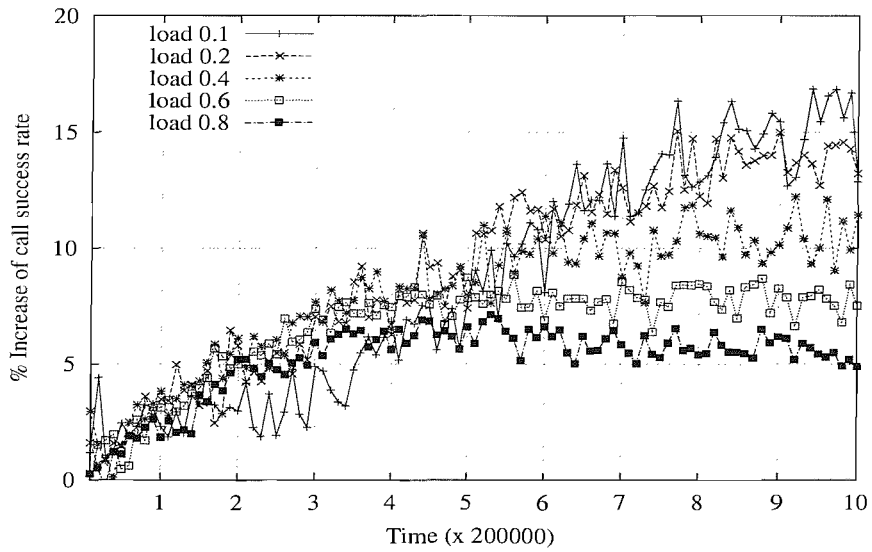
5.5.2.1 Call Success Rate.

Figure 5.8(a) shows the time variation of the CSRs of both e-PTC and PTC for different load values when the three nodes with the highest connectivity in figure 5.2 fail in succession. As explained in the results without failures, the CSRs of both e-PTC and PTC drop with increasing load: after all 3 nodes fail, e-PTC CSR is about 44% at load 0.1 compared to 16% at load 0.8. There is also a further drop due to failures. However, *the CSR for e-PTC is always higher than that for PTC, confirming the effectiveness of e-PTC over PTC.* The difference is more pronounced at lighter loads because at high loads the effects of both saturation and failures offset the advantage achieved by e-PTC (unlike the case without failures where the relative improvement was more at higher loads). Similar trends are observed in figure 5.9(a) which shows the same measurements when the topology of figure 5.3 is used.

Now, figure 5.8(b) plots the time variation of the relative difference of the CSR of

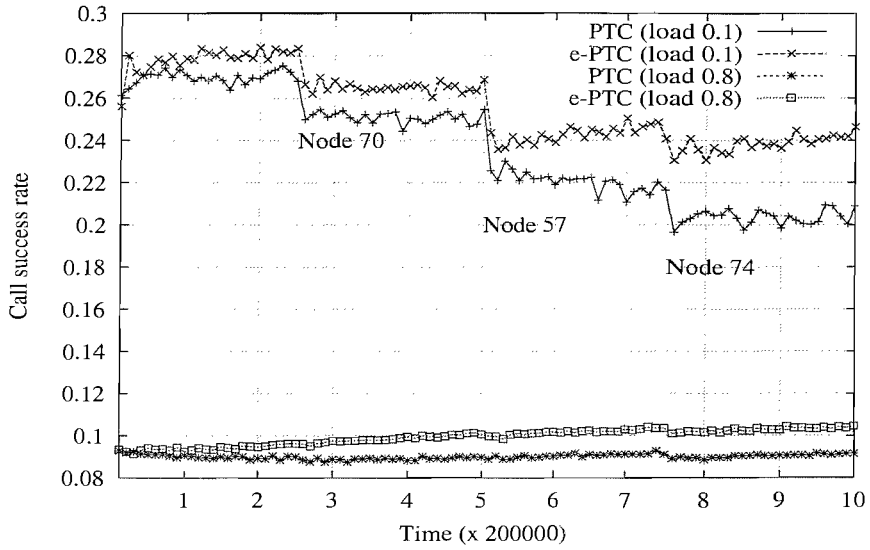


(a) Call success rate

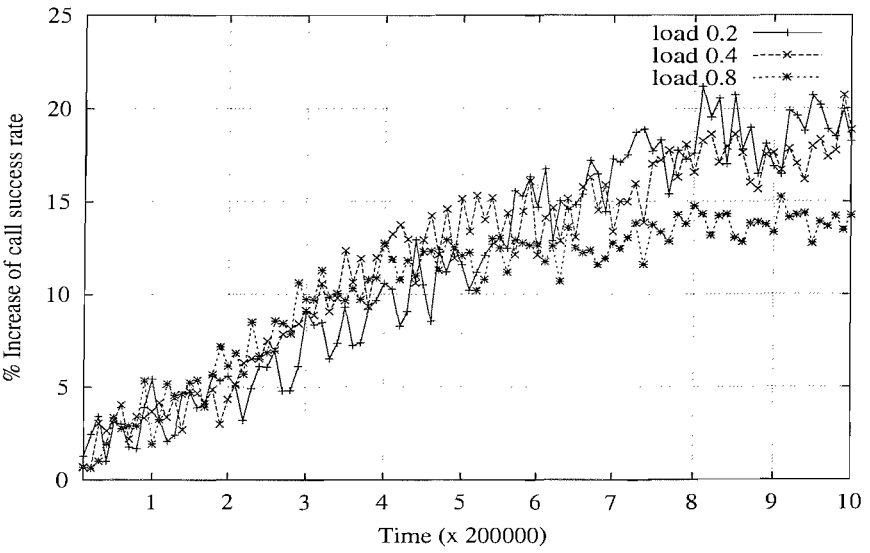


(b) Relative increase of success rate

Figure 5.8: Call success rate when 3 nodes fail: (a) - absolute CSR, (b) - percentage CSR increase in e-PTC over PTC on the 50-node topology of figure 5.2



(a) Call success rate



(b) Relative increase of success rate

Figure 5.9: Call success rate when 3 nodes fail: (a) - absolute CSR, (b) - percentage CSR increase in e-PTC over PTC on the 100-node topology of figure 5.3

e-PTC and PTC, (CSR(ePTC) − CSR(PTC))/CSR(PTC), for various loads. Here, after all three nodes fail, e-PTC CSR is about 15% higher than PTC at load 0.1 compared to 6% at load 0.8. Also, in figure 5.9(b), the relative improvement is about 20% at load 0.2 and about 14% at load 0.8. Thus, across all load values, e-PTC achieves a significant improvement in the CSR over that of PTC.

It is further observed that, *at a given load, the relative improvement increases with the number of failures.* Hence, although the CSRs of both e-PTC and PTC decrease with failures, *the rate of deterioration is lower in e-PTC. This further confirms its advantage in providing better adaptiveness against dynamic changes than PTC.*

5.5.2.2 Call Success Rate Recovery.

Figure 5.10 shows the time variation of the CSRs of the tests and the baseline at load 0.1 when the node with the highest connectivity fails in the topology of figure 5.2. Before the failure, e-PTC (test) has the highest CSR (similar to without failures). The baseline has the lowest CSR since, with one less node, it has the least amount of resources among the baseline and test cases. However, in the test cases, post failure, the CSRs of both e-PTC (test) and PTC (test) drop, from which the CSR of e-PTC (test) recovers at a faster rate than PTC (test). Figure 5.11 shows the similar trends on these measurements taken under identical conditions when the node with the highest connectivity fails in the topology of figure 5.3.

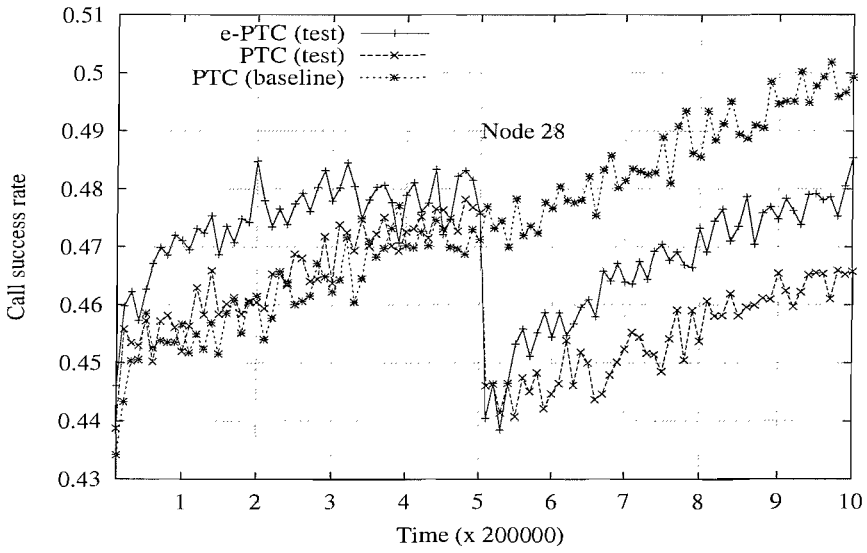


Figure 5.10: Call success rate with failure: e-PTC (test), PTC (test) and baseline (load 0.1) on the 50-node topology of figure 5.2

The deviation of the test from the baseline, ($| \mathrm{CSR}_{\mathrm{base}} - \mathrm{CSR}_{\mathrm{test}} |)/\mathrm{CSR}_{\mathrm{base}}$, obtained from figure 5.10, is plotted over time in figure 5.12. One plot corresponds to the deviation of e-PTC(test) from the baseline while the other corresponds to the deviation

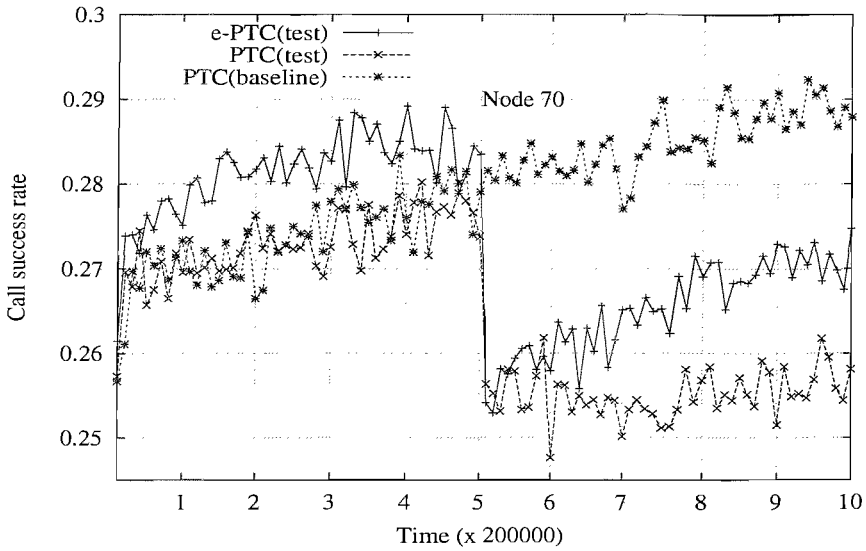


Figure 5.11: Call success rate with failure: e-PTC (test), PTC (test) and baseline (load 0.1) on the 100-node topology of figure 5.3

of PTC(test) from the baseline. It is observed that, *post failure, the relative difference of e-PTC (test) and the baseline is smaller and reduces at a rate faster than the same for PTC (test).* For example, e-PTC (test) is about 2.5% of the baseline which is 2.8 times better than that of PTC (test) which is 7% of the baseline. These trends are also observed for all other load values used. Moreover, figure 5.13 plots the relative differences of the CSRs of the tests and the baseline, obtained from figure 5.11, when the topology of figure 5.3 is used. Similar trends are observed in this figure as that in figure 5.12. Thus, *the e-PTC CSR recovers from failure with superior efficiency compared to that of PTC. This observation reinforces the claim that e-PTC achieves better adaptive behaviour against dynamic environment conditions than PTC.* The recovery is found to be more pronounced at lighter loads. Since the CSRs of all strategies suffer with increasing load, the CSR differences between the test cases and the baseline are lower at higher loads. Experiments with multiple node failures also reveal similar broad trends in the CSR recovery.

5.5.2.3 Message Size.

Figure 5.14 plots the relative increase in the average MS of e-PTC over PTC for different load values when the three highest connected nodes in the network of figure 5.2 fail in succession. It is observed that the relative increase in e-PTC MS becomes greater with load. Thus, in figure 5.14, when the load value is 0.2, there is a 180% increase in the e-PTC MS over that of PTC at the end of a simulation (that corresponds to a 2.8 times increase), and about 360% increase at load 0.8 (a 4.6 times increase). Now, e-PTC appends state values at every call failure and connection, but PTC uses only the latter.

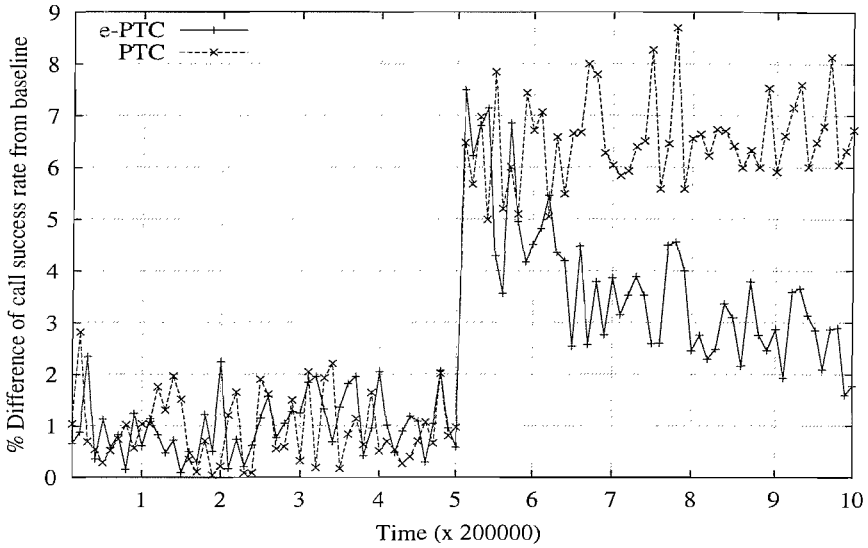


Figure 5.12: Call success rate deviation of e-PTC and PTC from baseline with node failure (load 0.1) on the 50-node topology of figure 5.2

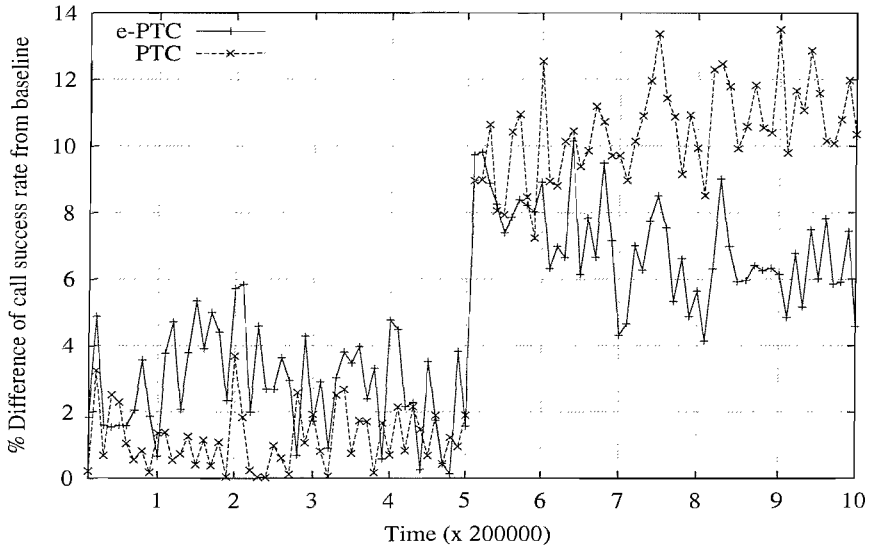


Figure 5.13: Call success rate deviation of e-PTC and PTC from baseline with node failure (load 0.1) on the 100-node topology of figure 5.3

Thus, as call failures increase due to increased saturation at higher loads coupled with node failures, so does the MS of e-PTC. PTC, however, does not incur any increase in MS due to call failures as no `drop` messages in PTC contain state information. Similar trends are observed in figure 5.15 that shows the same measurements when the topology of figure 5.3 is used. Thus, a tradeoff exists between the message overhead of e-PTC and its advantages over PTC discussed previously. At this point, a comment similar to that in section 5.5.1.2 applies to the relative significance of the increase in the call channel occupancy (a 20% relative increase in CSR is shown in section 5.5.2.1) and that in the control channel occupancy as shown here. Since a limited-size call channel

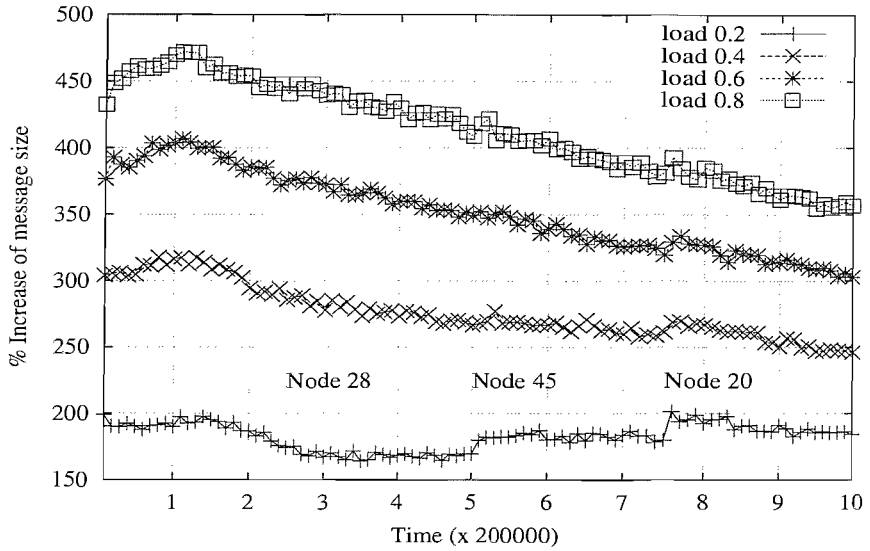


Figure 5.14: Percentage increase in e-PTC message size with 3 failures on the 50-node topology of figure 5.2

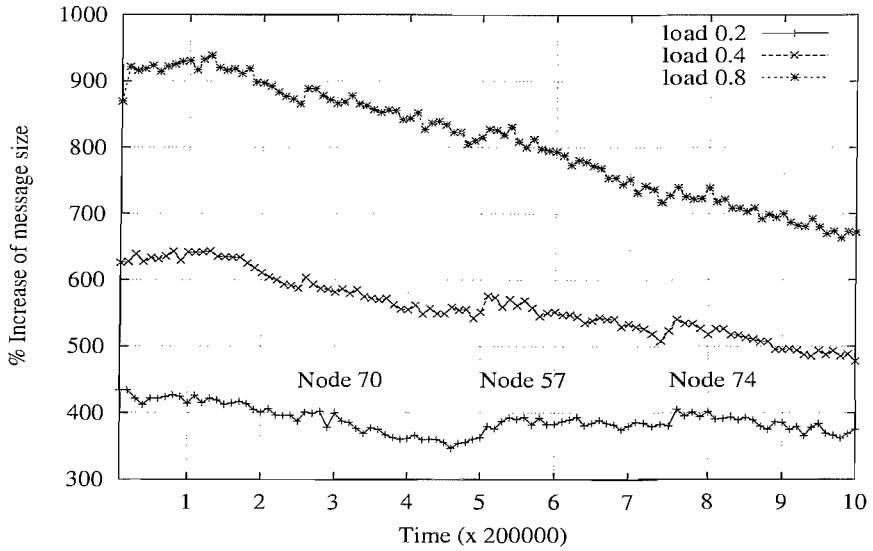


Figure 5.15: Percentage increase in e-PTC message size with 3 failures on the 100-node topology of figure 5.3

presents the largest constraint against achieving a high call success rate, a 20% relative increase in the call channel occupancy is a more significant result (indicating a far more efficient exploitation of the limited resources by e-PTC than PTC) than the increase in the control channel occupancy. It will be the topic of investigation in the following chapter whether the increase in the control channel occupancy in e-PTC can be restricted by making information transmission in e-PTC selective by letting nodes append state information only when necessary. It is envisaged that such a strategy could be able to reduce the control channel occupancy of e-PTC by reducing the sizes of the `ack` and `drop` messages.

5.6 Failure Diagnosis Using e-PTC

As discussed in section 5.2, e-PTC is designed to overcome the limitation of PTC in not being able to generate effective adaptive behaviour in dynamic environments caused by network failures. To substantiate this claim, section 5.5 shows empirically how e-PTC achieves superior bandwidth allocation than PTC when failures occur. This section describes how e-PTC facilitates the important functionality of *diagnosing* failures, a necessary prerequisite for automating distributed network maintenance. Here, diagnosis means that a node raises an alert when it interprets, from the information received, a fail condition of another node. Note that failure *diagnosis* in this section is different from the function of the failure detector model discussed in section 5.1.2. In section 5.1.2, the model describes how a *neighbouring* node of a failed node identifies the failure which then initiates information transmission via the `drop` messages. In this section, diagnosis implies *any* node in the network being able to interpret a failure using the information transmitted with these `drop` messages. In this context, we refer to the node that does the interpretation as the "monitoring" node and the one that is being interpreted as the "monitored" node. Note, any node can take either of these roles because information is potentially distributed between all nodes as calls can exist between all nodes and, thus, e-PTC distributes information between all node pairs. The aim of the diagnostic function is to ensure it is both truly indicative of a failure (no false positives), and that all failures will be detected (no false negatives). To ensure the first property, the monitoring node needs to observe a series of state values of the monitored node (explained shortly). This introduces a tradeoff between the time elapsed since a failure occurs and when it is correctly diagnosed, and the "cost" of a false alarm. The second property guarantees a detection. In the following, first, the failure diagnosis process is described. Then, an algorithm is designed for the nodes to achieve such diagnosis in a decentralised fashion using the information distributed by e-PTC.

5.6.1 The Failure Diagnosis Process

To explain the diagnosis, we label the monitoring node as n_a and the monitored node as n_b. Thus, our algorithm is used by n_a to interpret if n_b has indeed failed or not. Let T be the average interval at which n_a receives the state values of n_b that is transmitted by n_b either along with the `ack` or the `drop` messages (the e-PTC mechanism). Now, this information about n_b's state can originate either when n_b is saturated or unsaturated, or when it has failed. But, the state value of a node that is saturated and that is failed are both zeros (as discussed in section 5.3). However, the difference is that, a saturated node becomes unsaturated when one or more ongoing calls terminate, thereby freeing bandwidth, whereas a failed node continues to remain so. Therefore, after receiving the first zero state of a saturated n_b, if n_a continues to observe the state values, then it will receive a non-zero state after a *certain number of observations*, when n_b becomes unsaturated. Now, if n_a has the information about the average durations for which n_b remains alternately saturated and unsaturated, then it can compute the number of such observations using its knowledge of T (without this information, it cannot perform this computation). We assume that n_b remains alternately saturated for an average period of $\hat{t}_s$ and unsaturated for $\hat{t}_{us}$. To summarise, when n_a receives a zero-state of n_b, it assumes n_b is saturated. Then, using the knowledge of T, $\hat{t}_s$, and $\hat{t}_{us}$, n_a computes *the minimum number of observations* (termed as m) *after which it would receive a non-zero state* of n_b. If the actual number of zero-state values of n_b received exceeds m, then n_a interprets a failure by rejecting its assumption of a saturated n_b.

In the above description, out of T, $\hat{t}_s$, and $\hat{t}_{us}$, n_a can estimate online the average interval (T) at which it receives information from n_b. However, it cannot estimate online $\hat{t}_s$ and $\hat{t}_{us}$ of n_b since it does not have complete information about overall network load. Nevertheless, n_a can acquire these estimates from node traffic statistics that are usually available for most networks (FloodNet). We present the algorithm to compute m in the following section.

5.6.2 The Diagnosis Algorithm

After receiving the first zero state of n_b, n_a invokes the algorithm assuming the state value to be from a $\hat{t}_s$ interval of n_b (i.e., from a saturated n_b; it may of course be from a failed n_b). Now, knowing that it would receive state values from n_b after every T units and that n_b would remain alternately saturated for $\hat{t}_s$ and unsaturated for $\hat{t}_{us}$, n_a determines whether the next reception would be from a $\hat{t}_s$ or a $\hat{t}_{us}$ interval of n_b. This is done by comparing T against the sum of $\hat{t}_s$ and $\hat{t}_{us}$. This process is repeated until the next projected reception is from a $\hat{t}_{us}$ interval; the total number of receptions before this happens is equal to m.

Figure 5.16 shows the algorithm in detail. From our previous explanation of what the algorithm achieves, the input to the algorithm are the parameters T, $\hat{t}_s$, and $\hat{t}_{us}$ and the output is m, the minimum number of zero-state messages n_a would observe from n_b. In figure 5.16, T' denotes the sum of $\hat{t}_s$ and $\hat{t}_{us}$. The algorithm identifies the various possible relationships between T and $\hat{t}_s$ and $\hat{t}_{us}$ to decide how to compute m.

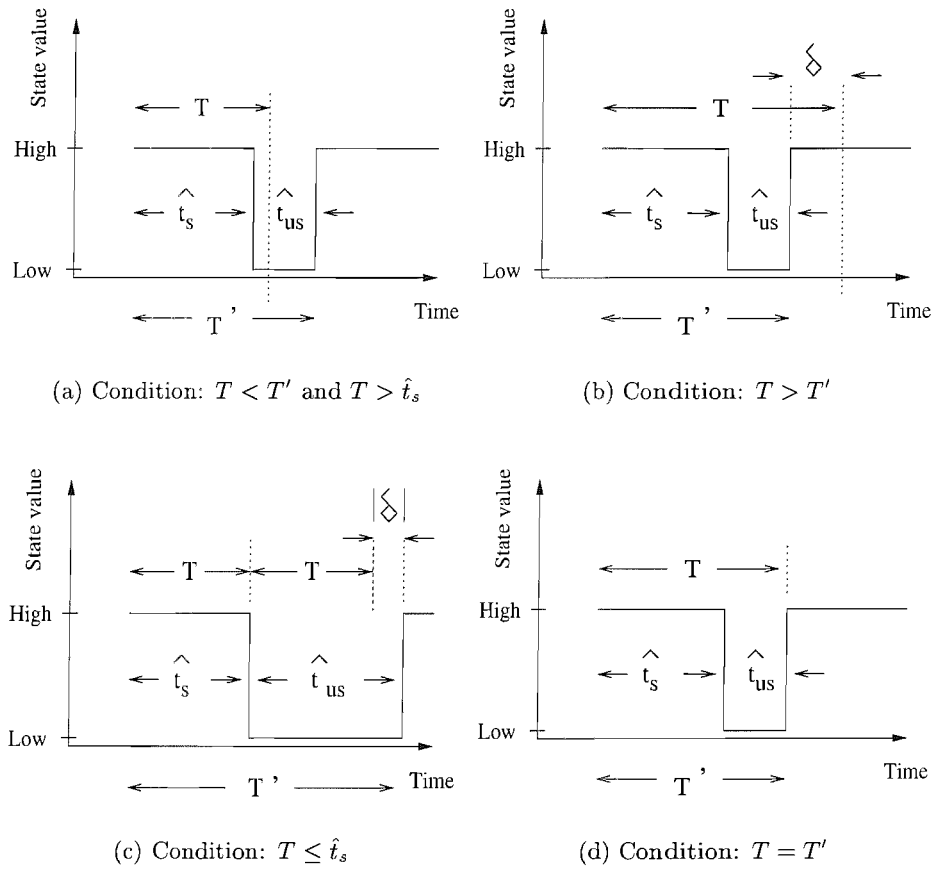
Input: T, $\hat{t}_s$, $\hat{t}_{us}$; Output: m.

0. $T' \leftarrow \hat{t}_s + \hat{t}_{us}$; $m \leftarrow 1$;
1. if($T < T'$ and $T > \hat{t}_s$)
2. $m \leftarrow 2$; //2nd msg in unsaturated interval
3. else if ($T > T'$) //next msg in a subsequent T' zone
4. $\delta \leftarrow T\%T'$; //modulus
5. while($(\delta - \hat{t}_s) < 0$)
6. $m \leftarrow m + 1$;
7. $\delta \leftarrow (T - (T' - \delta))\%T'$; //offset from start of next interval
8. else if($T \leq \hat{t}_s$) //next msg in a saturation zone
9. $\delta \leftarrow 0$;
10. do
11. $m \leftarrow m + \frac{\hat{t}_s - \delta}{T} + 1$; //integer division
12. $\delta \leftarrow T(\frac{\hat{t}_s - \delta}{T} + 1) - (T' - \delta)$; //offset from end of current interval
13. while($\delta > 0$);
14.else
15. $m \leftarrow \infty$; //to observe infinite zero-state messages; diagnosis inconclusive

Figure 5.16: Failure diagnosis algorithm

First, if $T < T'$ and $T > \hat{t}_s$ (line 1), the next state value of n_b (after an interval of T since the first reception) would be from its $\hat{t}_{us}$ interval and, hence, $m = 2$. Figure 5.17(a) shows this condition in schematic. Here, the saturated and unsaturated states of n_b are represented (only for clarification) by a pulse wave, where the "high" value corresponds to the $\hat{t}_s$ interval and the "low" value corresponds to the $\hat{t}_{us}$ interval. Along the x-axis is time and the y-axis represents state-value. Thus, from the start of the first $\hat{t}_s$ interval (corresponds to the first zero-state message that n_a received from n_b), the end of a period equal to T (which is when n_a would receive the next message from n_b) falls within the $\hat{t}_{us}$ interval; which is when n_a will receive a non-zero state value. Thus, n_a would receive a non-zero state value on the message after the first zero-state message it received from n_b. Hence, $m = 2$.

Second, if $T > T'$ (line 3), first, it is computed by how much (δ) the next reception is within the next T' of n_b (line 4). Then, this offset is recomputed (line 7) for successive observations (within the loop starting at line 5) until it is greater than $\hat{t}_s$ of the next T' (line 5), which indicates a non-zero state. Until this happens, in each iteration one more zero-state message needs to be observed by n_a (line 6). Figure 5.17(b) shows this



(a) Condition: $T < T'$ and $T > \hat{t}_s$

(b) Condition: $T > T'$

(c) Condition: $T \leq \hat{t}_s$

(d) Condition: $T = T'$

Figure 5.17: Schematic of various relationships between T, $\hat{t}_s$, and $\hat{t}_{us}$ described in the diagnosis algorithm of figure 5.16

condition in schematic.

Third, if $T \leq \hat{t}_s$ (line 8), a similar calculation is done as above. Specifically, in line 12, $T(\frac{\hat{t}_s - \delta}{T} + 1)$ gives the total duration of the consecutive observations made within an offset-adjusted $\hat{t}_s$ interval, and $(T' - \delta)$ gives the effective (offset-adjusted) T' in which the above calculation is done. Thus, the new offset from the end of the current T' is the difference of the above two values (line 12). The process is repeated until the next observation is in the $\hat{t}_{us}$ of the current T' which is when the offset is less than zero (line 13) and indicates a non-zero state. As an example of this scenario, consider the case where $T = \hat{t}_s$. Then, δ at the end of iteration 1 is (line 12): $\delta = T(\frac{\hat{t}_s - 0}{T} + 1) - (T' - 0)$, $= T(\frac{T}{T} + 1) - T', = 2T - T'$. If, $2T < T'$, then $\delta < 0$ (condition of line 13 violated) and, thus, the third message after the first zero-state message that n_a received will be from a $\hat{t}_{us}$ interval of n_b; so, in this example, $m = 3$. Figure 5.17(c) shows this example in schematic. If $2T > T'$, then the condition of line 13 holds. So, in the next iteration, an offset-adjusted $\hat{t}_s$ and T' are used (where the offset is the δ of the previous iteration) to re-compute a new δ.

Lastly, when $T = T'$ (line 14), the next state from n_b always coincides with a $\hat{t}_s$ interval (see the schematic of figure 5.17(d)). Therefore, n_a receives an endless sequence of zero states from n_b and, hence, cannot distinguish between a saturated and a failed n_b.

The above algorithm can be invoked when n_a receives a zero-state of a *saturated* n_b. However, assuming n_a has the correct estimates of T, $\hat{t}_s$, and $\hat{t}_{us}$, it will eventually receive a non-zero state of n_b on the m^{th} instance since the first zero-state instance. This is when n_b has become unsaturated due to the termination of one or more existing calls. Hence, n_a would not signal a failure which is the correct diagnosis since n_b was only saturated. Hence, the diagnosis does not generate any false positives. Alternatively, if n_b had indeed failed when n_a received the first zero-state, it would continue to be failed indefinitely. Therefore, n_a would eventually receive more than m zero-state values from n_b and then signal a failure which is also the correct diagnosis. Thus, the diagnosis does not generate any false negatives.

The above algorithm is put to empirical evaluation in simulated network failure scenarios using the topologies of figure 5.2 and 5.3. The characteristic features of the diagnosis achieved in these simulated studies are reported in the following section.

5.7 Results and Analysis

The failure detection algorithm is subjected to empirical evaluation in the simulated network routing problem. In a given network (both topologies of figures 5.2 and 5.3 are used), one node (n_a) is selected as the monitoring node and then, in turn, all other

nodes are selected to fail (n_b), each in a differect simulation with all parameters the same as before. In each simulation, only one node failure is implemented which fails at the halfway point of a simulation run. Thus, by selecting a different n_b in every simulation, an overall assessment of the characteristics of the diagnosis achieved by the selected n_a (using the algorithm of figure 5.16) can be obtained. Note that this approach of selecting one node to fail in each simulation does not necessarily limit the overall conclusions about the performance of our diagnosis algorithm. While our algorithm is theoretically capable of detecting failures independent of the number of failures occuring, selecting multiple nodes to fail might introduce a practical difficulty of state information not reaching the monitoring node due to the network getting disconnected. In such a situation, the monitoring node would not be able to compute T for some failed nodes, which is required for carrying out the diagnosis. Hence, to keep the simulation simple, yet without losing any generality of results, a single node is chosen to fail.

For every other node in the network, n_a estimates T and the simulator estimates $\hat{t}_s$ and $\hat{t}_{us}$ (as mentioned before, a node which cannot observe the load variations at other nodes in the network is, therefore, unable to estimate the $\hat{t}_s$ and $\hat{t}_{us}$ values of other nodes), which are fed to n_a using which, it can invoke the algorithm to compute m. It does this whenever it receives a zero state value of any other node. However, for all nodes other than the selected n_b, a zero state is only due to the latter being temporarily saturated. Now, n_a can correctly identify this fact since it has the values of T, $\hat{t}_s$, and $\hat{t}_{us}$. More specifically, n_a computes a value of m after it receives a zero state of any node but it receives a non-zero state value from the same node before the m^{th} subsequent reception from that node. Thus, n_a does not issue a failure alert in this case. However, after the selected n_b fails and n_a receives a zero state value from n_b, it again computes m. But this time n_a keeps receiving zero state values of n_b even after the m^{th} subsequent reception. So, in this case, n_a reports a failure of n_b. In the experiments conducted, it is observed that n_a reports a failure of only the chosen n_b and only after n_b has failed (no false positives) and in every experimental run (no false negatives). However, it should be noted that n_a *reports a failure in each case after a delay of at least* $(m-1) \times T$ *time units after the failure has actually occurred.*

In the following, we report results from the diagnosis experiments that use node 8 in the 50-node topology of figure 5.2 and node 13 in the 100-node toplogy of figure 5.3 as the monitoring nodes. A load value of 0.6 was set for these experiments (similar broad trends were observed in experiments that used other load values). Now, in each experiment, a different node (the monitored node) is set to fail. From these experiments, the average values of m and T, as computed by n_a, over the monitored nodes that are at a given distance from n_a are computed. Hence, a summary assessment of how many consecutive zero-state observations n_a requires to detect a failure at a given distance is obtained. Moreover, from these values of m and T, an estimate of the average delay

Table 5.1: Summary statistics of failure diagnosis on the 50-node topology of figure 5.2

Min dist of target	T		m		Avg. detection delay
	avg	stdev	avg	stdev	
1	420	50.7	2.8	1.2	756.0
2	1559	562.4	2.9	1.4	2962.1
3	7402	5339.3	2.3	1.2	9622.6
4	28616	15895.0	2	0	28616

Table 5.2: Summary statistics of failure diagnosis on the 100-node topology of figure 5.3

Min dist of target	T		m		Avg. detection delay
	avg	stdev	avg	stdev	
1	372	11.2	2.2	0.3	446.4
2	4148	491.1	2.0	0.1	4148.0
3	10807	658.9	2.0	0.2	10807.0
4	20265	1870.5	2.1	0.4	22291.5
5	33336	4668.6	2.5	1.3	50004.0
6	44569	15631.4	2.0	0.0	44569.0
7	63416	8510.4	2.1	0.6	69757.6
8	132917	15006.6	1.85	0.3	112979.4
9	464743	13766.6	2.0	0.1	464743.0
10	425047	21690.3	2.0	0.1	425047.0

$((m-1) \times T)$ incurred by n_a for detecting failures that are at a given distance from n_a can be evaluated.

In more detail, table 5.1 shows the average estimates of m, T, and the delay between the time a node fails and n_a actually raises a failure alert for the 50-node topology. Table 5.2 shows these values for the 100-node topology. Now, *in each experiment, involving either of the topologies, n_a has raised a failure alert only after the chosen n_b has failed. Thus, the properties of no false positives and no false negatives of the algorithm of figure 5.16 are corroborated by these empirical results.* Nevertheless, both tables show that the detection delay increases with increasing distance of the monitored nodes. This is as expected because of the increasing time interval (T) between receiving state information from distant nodes. Note, the value of T is determined by the system parameters, such as topology, load, call pattern, and the information-sharing protocol used. By reducing the value of T, such as by distributing information more frequently and along multiple paths, the failure detection delay can be reduced; a modification we intend to do in our future work. It should be highlighted that, the delay becomes significant only for those monitored nodes that are farther away from the monitoring

node. However, note that in these experiments, only one node is selected to do the monitoring of the entire network. On the other hand, by deploying multiple monitoring nodes across the network, the maximum distance from any given node (that can fail) and at least one monitoring node can be restricted such that the detection delay of any node remains within tolerable limits. The objective of this empirical study, however, is to verify the theoretical properties of the detection algorithm. From the results obtained, therefore, it can be concluded that this objective is achieved. Moreover, achieving these results by using only a single monitoring node further strengthens our claim since a single monitoring node is a "harder" test for the diagnosis algorithm compared to using multiple monitoring nodes.

5.8 Summary

In this chapter, we have improved our PTC information-sharing protocol so that agents can adapt their behaviour effectively in response to unpredictable environment changes. Such adaptive behaviour is a necessity as most distributed systems are characterised by dynamism that cannot be ascertained a priori. Specifically, such behaviour incorporates a much needed resilience in such decentralised systems that do not have a central monitoring and management center. To this end, this chapter uses empirical analysis on the network call routing domain to demonstrate that the extended PTC protocol is very effective in adapting network performance in response to changes caused by network failures and in diagnosing such failures. Hence, the extended protocol and the empirical studies both contribute by further substantiating our thesis that effective information-sharing can generate good adaptive behaviour.

Moreover, this chapter also contributes towards improving the state-of-the-art technology in handling failures in our domain of interest — limited-bandwidth, low-power mesh networks used for telephony. In such decentralised systems, ascertaining the effects of failures and detecting them are extremely challenging objectives. Thus, it is of critical importance to ensure that the network adapts to failures effectively and also provides a decentralised failure detection mechanism. So, by designing the e-PTC protocol, this chapter contributes by providing solutions to the above-mentioned problems in this domain. Specifically, the e-PTC protocol of this chapter exploits the message transmissions occuring after call failures by appending the state values of nodes to these messages. This simple mechanism allows the nodes to effectively update their estimates about the bandwidth availability changes caused by failures. By remaining more up-to-date, e-PTC achieves better bandwidth allocation quality than PTC and it supports failure detection (thus, better adaptiveness to system dynamism). These results are shown using empirical studies on various network topologies and environment settings, which show that the call success rate achieved by e-PTC is significantly higher than that

of PTC when failures occur. In fact, even when there are no failures, e-PTC is shown to achieve higher call success rates than PTC (indicating an inherent improvement in adaptive behaviour). Moreover, these empirical studies also demonstrate that, after failures, e-PTC attains a steady state call success rate faster than PTC which indicates a better responsiveness to failures by e-PTC. More importantly, perhaps, the simple mechanism of e-PTC allows the nodes to diagnose the states of other nodes and detect failures. In particular, an algorithm is designed through which a node can assess if another node has failed both with certainty and only when a failure has truly occurred. The algorithm is subjected to empirical evaluation where the capability of a node in correctly diagnosing failures of other nodes is verified. The results of these studies corroborate the claims of the algorithm.

The above-mentioned contributions provide substantial evidence towards furthering our thesis. Nevertheless, now it is our objective to improve the performance of PTC more than that achieved in this chapter. In particular, we focus on controlling the cost of communciation incurred by e-PTC in achieving the desired adaptive behaviour. Note that the advantages of e-PTC are achieved by distributing additional information between nodes. This approach results in an increase in the average size of the information messages than that of PTC and, hence, indicates an extra overhead of e-PTC. In the following chapter, we attempt to make e-PTC more effective by reducing this message size overhead but without losing the advantage of generating good adaptive behaviour.

Chapter 6

Selective Post Task-Completion Information-Sharing

So far in this research, we have provided both theoretical justifications and empirical analyses to substantiate the fact that an effective information-sharing protocol can be used for generating quality estimates and, hence, good adaptive behaviour in sequential RA tasks — the claim made in our thesis in chapter 1. More specifically, chapter 3 developed the PTC protocol and established, using a formal analysis, its significance in generating good quality state estimates in sequential RA tasks. Then, chapter 4 further substantiated this theoretical claim by way of extensive empirical evaluation of PTC against other benchmark algorithms in an exemplar sequential RA domain. Subsequently, chapter 5 improved PTC by incorporating additional information-sharing so that it can generate effective adaptive behaviour against unpredictable events such as network failures. Now, while this modification ensured significant advantages over the basic PTC by generating higher call success rates, faster responses to failures, and detection of failures, it suffered from using larger messages. This property represents an overhead incurred by the system. In particular, in limited-bandwidth networks, such as the one we study in this research, it is essential to minimise the size of control messages to save valuable bandwidth as much as possible.

This chapter aims to address the above-mentioned limitation. Specifically, the objective of this chapter is to make e-PTC more effective by reducing its message size while retaining (as far as possible) its advantages of good adaptiveness in dynamic systems. Thus, we aim to substantiate our thesis even further by developing a highly effective communication protocol capable of providing superior performance in sequential RA tasks in distributed systems.

The line of approach adopted to achieve the above-mentioned objective is to make information distribution *adaptive* in PTC. Broadly speaking, this means that an agent

transmits its local state value to another only when it deems the information is necessary for the recipient. Thus it attempts to restrict the message size overhead in the system by communicating selectively. Note, in all previous PTC heuristics, an agent transmits state values on *every* call success (both PTC and e-PTC) and failure (only e-PTC) without regards to whether the transmitted information would indeed be useful to the receiving agent.

In the rest of this chapter, section 6.1 motivates the design of the adaptive PTC protocol, details the adaptive version of PTC, and analyses the pros and cons of the design. Subsequently, section 6.2 uses this new protocol in our running network call routing problem and presents empirical results that compare its performance against e-PTC. Finally, section 6.3 reviews the contributions of this chapter.

6.1 Adaptive PTC (a-PTC)

This section motivates the design of an adaptive version of our basic PTC protocol. Moreover, a formal description of the a-PTC protocol is presented in the context of our network routing problem. Also, the possible impact on system performance that a-PTC might have is analysed.

6.1.1 Motivation for Designing a-PTC

The objective of transmitting a local state value to a neighbour is to inform the latter about its unobservable network states. This, in turn, allows the recipient to make estimates of the bandwidth availability on other nodes so that it can take judicious routing decisions. In this context, however, it can be the case that the state of a node at a given time can be identical to what it transmitted on the previous occasion. This happens if, after the last transmission, exactly as many new calls have been connected through this node as existing calls on this node have terminated and, hence, the resultant bandwidth availability on this node is the same. In that case, repeating the transmission of the same state value to the same neighbour would be redundant.[1] Therefore, a node can do without transmitting the information if it detects the above condition. Note that since state values are transmitted by being appended to information messages (`ack` or `drop`), not transmitting a state value implies that the size of the message does not increase. In this manner, therefore, by making information distribution selective, the overhead of increasing message size can be restricted.[2] Nevertheless, to determine

[1] Of course, the recipient node can identify this redundancy only if it maintains a record of the last state values received from its neighbours. More on this shortly.

[2] Note that the *number* of `ack` and `drop` messages remains the same in e-PTC and a-PTC for an equal number of successful call connections and failures. However, it is the *size* of these messages that differs in a-PTC and e-PTC.

information redundancy, the nodes require additional memory to store communication history. Such a requirement indicates a potential downside of this protocol and, hence, a performance tradeoff. The following discussion elaborates on these issues.

To implement the above-mentioned selectivity in the information distribution, the following requirements are identified:

Transmitter memory. A node should store in memory the state values it transmitted to *each of its neighbours* on the most recent transmission. It needs this information to compare its current state to that communicated on the most recent transmission to a particular neighbour so that it can decide whether the current transmission could be redundant (in case the two values are identical).

Receiver memory. A node has to store in memory the state values it receives from *any other node* on the most recent transmission from the latter.[3] Thus, when a node receives the `ack` or a `drop` message after a call connection or a failure, it uses this stored value (say, of node n_x, which is on the call path) to update its Q-estimate when it does not receive the current state from n_x. This happens when n_x detects a redundancy (as discussed before) and does not transmit its state value along with the `ack` or the `drop` message. On the other hand, when it receives the state of n_x (that is when n_x has not identified any redundancy), it uses that information to replace the last transmission it received from n_x and updates its Q-estimate with this new information.

Combining the above two requirements, this protocol necessitates a memory space of $(K_i + (N-1))$ for a node n_i, where N is the total number of nodes in the network and K_i is the total number of neighbours for n_i. Specifically, K_i is the size of the transmitter memory required to detect a redundancy in transmitted information, while $N-1$ is the size of the receiver memory used to record the most recent transmissions from other nodes. A formal description of the requirements and the mechanism associated with selective information distribution is provided in the following.

6.1.2 Formal Description of a-PTC

The following symbols are used to describe the system:

- n_i — identifier for node i in the system (same as in chapter 3).
- s_i — state of n_i (same as in chapter 3).

[3]Since the PTC protocol in the call routing problem allows state values to propagate along a call path, a node can potentially receive the state information from any other node in the network.

- $\mathcal{T}$ — set of the latest transmitted state values, each element of $\mathcal{T}$ corresponds to a neighbour. $\mathcal{T}_i(j)$ refers to the most recent state value transmitted to neighbour n_j by n_i.

- $\mathcal{R}$ — set of the latest state values received, each element of $\mathcal{R}$ corresponds to a distinct node in the system. $\mathcal{R}_i(j)$ refers to the most recent state value received by n_i from n_j.

$\mathcal{T}$ and $\mathcal{R}$ are used in the following manner to determine if a node should transmit its state information and how it would update Q from the state information received. The processes of call forwarding and message propagation are reproduced from chapter 4 in the schematic of figure 6.1 to aid this description. In this figure, a call is routed along the node sequence: $n_1, ..., n_{i-1}, n_i, n_{i+1}, ..., n_d$.

6.1.2.1 Transmitting State Value.

Referring to figure 6.1, during information propagation along with an `ack` or a `drop` message following a call connection or a failure, any node n_i $(1 < i \leq d)$ transmits its state to n_{i-1} if,

$$s_i \neq \mathcal{T}_i(i-1). \tag{6.1}$$

If condition 6.1 holds and n_i transmits its state s_i to n_{i-1}, then n_i updates $\mathcal{T}_i(i-1)$ as:

$$\mathcal{T}_i(i-1) = s_i. \tag{6.2}$$

On the other hand, if condition 6.1 is not true (i.e., s_i and $\mathcal{T}_i(i-1)$ are identical), then n_i does not append s_i to the `ack` or the `drop` message before sending it to n_{i-1}.

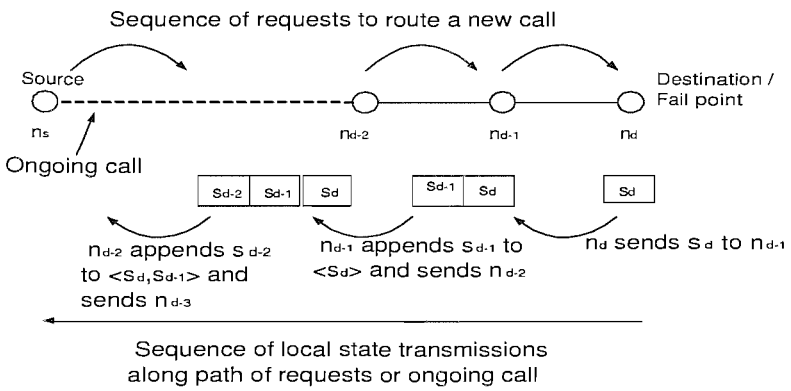


Figure 6.1: Schematic of call setup and sequence of node state transmissions

6.1.2.2 Updating Q.

Again, referring to figure 6.1, any node n_i $(1 \leq i < d)$ uses the following set of state values of the downstream nodes to update its Q-estimate.

$$\{s'_{i+m} \mid m = 1, ..., (d-i) \wedge s'_{i+m} \text{is the latest of } \mathcal{R}_i(i+m) \text{ and } s_{i+m}\}. \tag{6.3}$$

In the above set of state values (given by 6.3), n_i determines the "recency" of state information by checking if the state value of node n_{i+m} is present in the message. If it is, then n_{i+m} has transmitted this value by determining that this transmission is not redundant (using condition 6.1). Then, n_i uses this state value to update its Q-estimate and also updates $\mathcal{R}_i(i+m)$ as:

$$\mathcal{R}_i(i+m) = s_{i+m}. \tag{6.4}$$

On the other hand, if n_i does not find the state value of n_{i+m} in the message, then it uses the last state value received from n_{i+m}, $\mathcal{R}_i(i+m)$, to update its Q-value.

6.1.3 Implications of Selective Information Distribution

While state information is useful to an agent for updating its Q-estimates of the network states, it has the downside of producing larger messages. Against this background, selective information distribution, as described in the previous section, is envisaged to affect the following: (i) the memory requirement of a node, (ii) the size of the `ack` and the `drop` messages, and (iii) the quality of Q estimates. Each of these factors can, in turn, affect the performance of the network. Therefore, it is crucial to analyse the impact of selective communication on these factors and on the system performance in general. The following descriptions elaborate on each of these:

Memory requirement. As discussed before, each node n_i needs a memory of size $K_i + N - 1$ to store the $\mathcal{R}$ and $\mathcal{T}$ values. Thus, the memory requirement *increases linearly* with the number of nodes in the network and is constant for a given network (the memory requirement does not increase at run-time). Hardware memory is typically cheap and, thus, can be easily incorporated into the nodes. *However, here we are primarily concerned with improving bandwidth usage efficiency since bandwidth is a valuable resource and is very limited on wireless nodes.*

Message size. The larger the size of a message, the more node bandwidth it uses. The size of a message increases only if a node appends its state value to it. Using

selective information distribution, therefore, the message size cannot be any more than without selective distribution. In fact, node states remain unchanged over time intervals when new calls do not originate and when existing calls are ongoing. Therefore, there would be instances where an agent finds its state to be the same as the previous transmission where selective communication would prevent it from appending its state value. *Hence, it is envisaged that the size of* `ack` *and* `drop` *messages would be reduced significantly by using selective distribution.*

Quality of Q-estimates. The objective of communicating state values is to ensure that the agents are up-to-date about the bandwidth availability information across the network (as far as is possible). Now, since selective distribution prevents state information from being propagated, the concern is whether it adversely affects the quality of Q-estimates. In this context, note that selective communication prevents information propagation only if that information is identical to that in the previous transmission. Since the agents store in memory the latest transmissions ($\mathcal{T}$), any new information is guaranteed to be transmitted. *Hence, this protocol ensures that the Q estimates learned are up-to-date in exactly the same way as would be achieved with the basic PTC protocol (without selective distribution).*

6.2 Experimental Evaluation

As stated in section 6.1, the design objective of a-PTC is to reduce the message overhead of the e-PTC algorithm without sacrificing its advantage of improved bandwidth allocation. In this section, therefore, we carry out experiments to test if our design of the a-PTC algorithm, as defined in section 6.1, satisfies that objective. In particular, a-PTC is compared against e-PTC along the following dimensions:

- *Savings in message size overhead:* Given identical sets of environmental parameters and the same network topology, by how much (if any) the size of the `ack` and the `drop` messages are reduced using a-PTC than e-PTC.
- *Loss in call success rate:* Given identical parameter values, by how much (if any) does the call success rate reduce using a-PTC than e-PTC.

Experiments are conducted to measure the performance differences of a-PTC and e-PTC along the above two dimensions by varying a "selectivity threshold" (Δs) for communication. This threshold is the amount of change in the local state that a node needs to observe before it initiates a transmission of state information. In more detail, a threshold of Δs_i for node n_i implies that n_i communicates its local state information to its neighbour n_j only if:

$$| s_i - T_i(j) | \geq \Delta s_i. \tag{6.5}$$

It can be seen that the higher the value of Δs, the more the degree of communication selectivity, and the lower the number of information transmissions. This is because a large value of Δs implies that nodes communicate only when there is a large state change. However, when Δs is small, nodes communicate for both large and small state changes (when Δs equals zero, then we revert to e-PTC). Thus, the higher the value of Δs, the lower is the frequency of communication. Therefore, Δs acts as a parameter to set the degree of information-sharing in a distributed system. This implies that Δs has an effect on the size of the `ack` and the `drop` messages. Also, since the information distributed is used to learn the Q-estimates, Δs can, in turn, affect the quality of learning (equivalently, the CSR of the network). Thus, it is envisaged to influence the performance of a-PTC along both the above-mentioned dimensions. To evaluate this hypothesis, experimental results providing more insight about the impact of a-PTC on the system performance are presented and analysed in the following.[4]

As was the case in the previous empirical analyses (see sections 4.5 and 5.5), the two algorithms are evaluated by varying the call origination probability. The results from using the topologies of figures 4.3(a) and 4.3(b) are reported since the same broad trends are observed in other sample topologies. Experiments were run using call origination probabilities (p_c): $\{0.1, 0.2, 0.4, 0.6, 0.8\}$. For each of these values, the selectivity threshold Δs was varied between $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.[5] All nodes use the same value of Δs. The results reported are statistically significant at the 95% confidence level.

6.2.1 Impact on Call Success Rate

Both e-PTC and a-PTC were run using the above-mentioned parameter values and the average CSR was computed for both algorithms. For this case, figure 6.2 shows the time-variations of the CSRs of both algorithms for the topology of figure 4.3(a) at a p_c value of 0.2 and Δs equal to 0.1. The value of 0.1 is chosen for Δs since a node state only takes on values from {0.0, 0.1, ... 1.0} and, therefore, the value of 0.1 represents

[4]Note that we used an absolute state-change in equation 6.5. State changes occur either due to new calls being set up (using up bandwidth, hence, lowering state value) or ongoing calls terminating (increasing state value). An absolute measure, therefore, captures both these possibilities and allows communication in either case. Since the purpose of communication is to share information to allow the agents to learn the node states accurately, the absolute measure is the suitable approach. This is in contrast to using a relative value which can capture either a state increase (if using $(s_i - T_i(j)) \geq \Delta s_i$) or a decrease (if using $(T_i(j) - s_i) \geq \Delta s_i$). Hence, absolute state-change is used as the communication decision condition.

[5]Here, Δs is expressed as the fractional bandwidth available, hence it is expressed in the same units as the node state (see section 4.1 for the definition of node state).

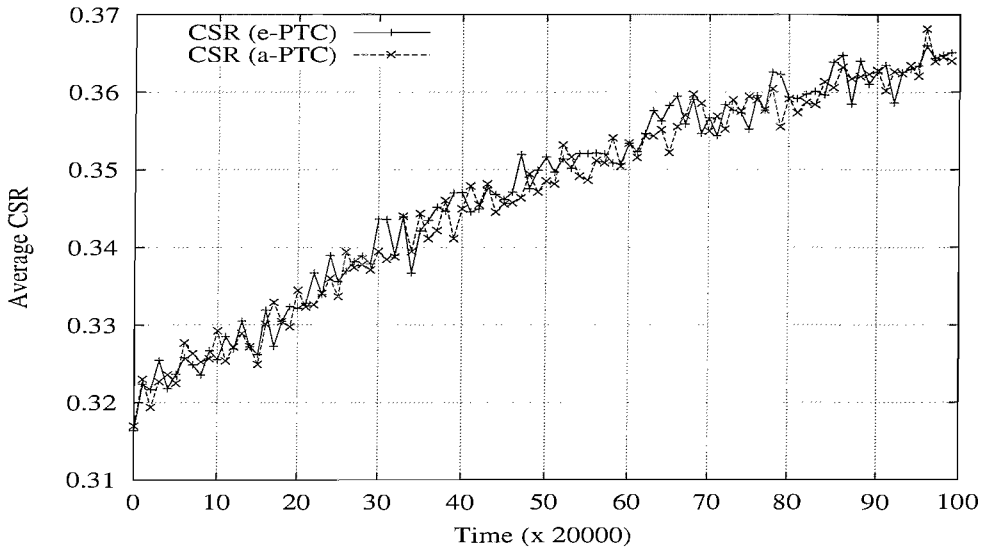


Figure 6.2: Time variation of average call success rates of e-PTC and a-PTC using the 50-node topology of figure 4.3(a), $p_c = 0.2$, $\Delta s = 0.1$

the minimum non-zero value of Δs for which the condition 6.1 holds. Thus, this value of Δs accounts for the information transmissions possible with all other values of Δs and thus, represents a "baseline" a-PTC.

As can be seen, this figure demonstrates that *using the adaptive PTC algorithm the CSR has not degraded by any significant amount when compared to that of e-PTC.* For example, e-PTC CSR is 0.365 at the end of the simulation, while it is 0.363 for a-PTC. Thus, in spite of restricting communication (the impact of such selectivity on the message size is described in the following subsection), a-PTC retains the superior bandwidth allocation quality of e-PTC. Figure 6.3 shows the analogous results when the topology of figure 4.3(b) is used where identical trends are observed. Moreover, these trends also hold for all values of p_c. These results, therefore, provide evidence to substantiate one of the design objectives of a-PTC.

Now the CSR performances of a-PTC and e-PTC are compared across the range of call origination probability and the selectivity threshold values. To summarise the trends in these results, the relative differences of the average steady state CSR (as defined in section 4.5) of a-PTC and e-PTC are used. More specifically, for every pair of Δs and p_c values, the value of $(CSR(ePTC) - CSR(aPTC))/CSR(ePTC)$ (where $CSR(x)$ represents the average steady state call success rate for protocol x) is computed. In figure 6.4, these values are plotted when the topology of figure 4.3(a) is used over a surface defined by the Δs and the p_c axes. The following trends are observed:

Observation 1: *At small selectivity threshold values, there is no significant degradation of a-PTC CSR compared to e-PTC for all call origination probabilities.* For example, in figure 6.4, for values of Δs less than 0.4, the CSR of a-PTC is within 4.0% of that

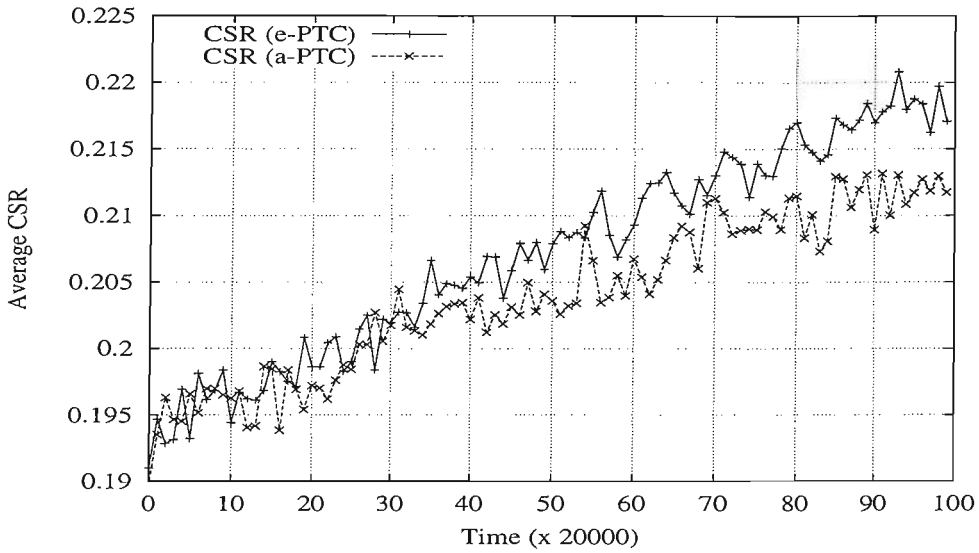


Figure 6.3: Time variation of average call success rates of e-PTC and a-PTC using the 100-node topology of figure 4.3(b), $p_c = 0.2$, $\Delta s = 0.1$

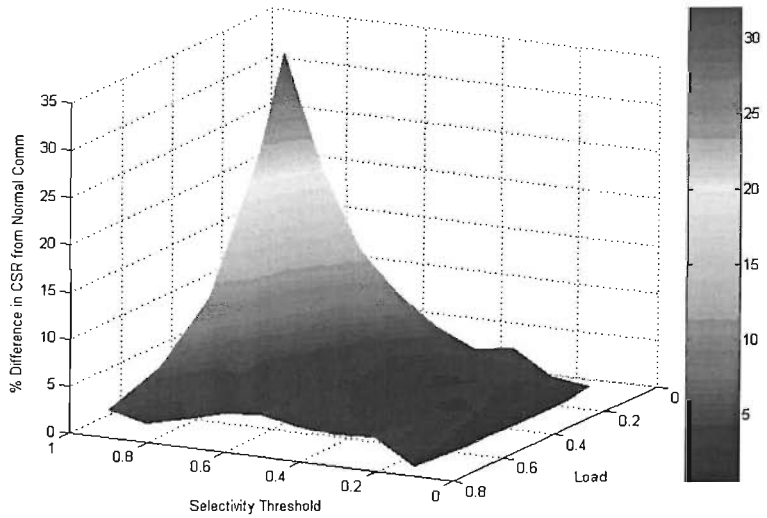


Figure 6.4: Comparing the call success rate profiles of a-PTC and e-PTC on the 50-node topology of figure 4.3(a)

of e-PTC for any value of p_c. A small value of Δs implies that communication of state information happens for both small and large changes in a node's state. Small state changes are more common than large ones at any given value of p_c. This is because, at any node, the origination and the termination of one or a few calls are more frequent than of a large number of calls. Moreover, the larger state changes occur more at higher p_c values. Thus, for any p_c, a low Δs value allows frequent communication of state information. This allows the nodes to remain updated fairly well about the network states and generate good call success rates (almost as well as that of e-PTC). Hence,

the CSR of a-PTC does not degrade significantly compared to that of e-PTC.

Observation 2: *At large selectivity threshold values, the a-PTC CSR degrades significantly compared to e-PTC when the call origination probability is low. However, at higher call origination probabilities, the a-PTC CSR is not significantly lower than e-PTC.* For example, in figure 6.4, at $\Delta s = 0.9$, there is a relative difference of about 32.0% in the CSRs of e-PTC and a-PTC (the e-PTC CSR being higher) when $p_c = 0.1$, but it is only about 3.0% when $p_c = 0.8$. Here, a high value of Δs implies that state information is communicated only when large state changes occur. Now, at small values of p_c, when few calls originate in the network, such large node state changes are rare (since all nodes are only sparsely occupied). Therefore, communication of state information is also infrequent; the nodes are not effectively informed about the state changes in the network. Hence, the Q-estimates are far from up-to-date which, in turn, adversely affects the CSR performance of a-PTC. On the other hand, when the value of p_c is larger, large state changes occur more frequently than when p_c is smaller. So, having a high selectivity threshold implies that the nodes communicate the state-change information more effectively at larger p_c values than at smaller p_c values. The Q-estimates, therefore, are more effectively updated and the CSR performance of a-PTC is not significantly degraded compared to that of e-PTC.

The trends observed in figure 6.5, which shows the same measurements using the topology of figure 4.3(b), are similar to those discussed above. These results confirm that the call success rate of a-PTC is not significantly lower than that of e-PTC over a wide range of environments — at all load values when the selectivity threshold is small and at high load values when the selectivity threshold is large.

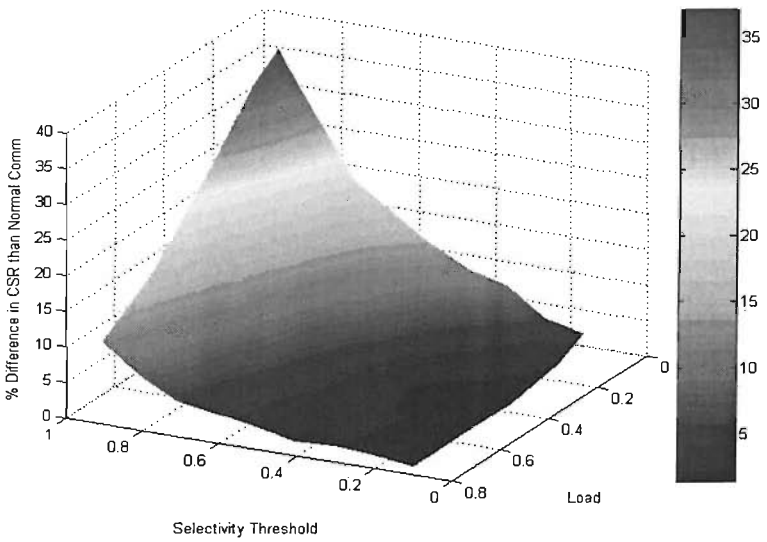


Figure 6.5: Comparing the call success rate profiles of a-PTC and e-PTC on the 100-node topology of figure 4.3(b)

6.2.2 Impact on Message Size

Analogous to the CSR results of e-PTC and a-PTC, figure 6.6 plots the time-variations of the average message size (considering all `ack` and `drop` messages generated by all nodes) of e-PTC and a-PTC using the topology of figure 4.3(a) at a p_c value of 0.2 and Δs equal to 0.1. Again, results are reported with $\Delta s = 0.1$ since this value represents a "baseline" case (as is explained in section 6.2.1). At $\Delta s = 0.1$, a-PTC is the least selective about transmitting information and, hence, a saving in message overhead over that of e-PTC would imply savings for all other (higher) values of Δs.

It has been shown in section 6.2.1 that the CSR of a-PTC is not significantly different from that of e-PTC under the given experimental setting. However, *figure 6.6 demonstrates that the MS of a-PTC is significantly lower than that of e-PTC (significance testing is performed at the 95% confidence level).* For example, the MS of e-PTC is 3.0 at the end of the simulation, while it is 2.0 for a-PTC: a 33% reduction in the MS relative to e-PTC.[6] Therefore, a-PTC generates a far lower message overhead than e-PTC. Figure 6.7 shows identical trends when the topology of figure 4.3(b) is used. Similar trends are observed for all other values of p_c. Hence, these results provide evidence to substantiate the second design objective of a-PTC. In this context, it is also important to note that, the `ack` and `drop` messages occupy a significant portion of the control channel.[7] Thus, a reduction in the size of these messages due to the selective communication mechanism is indeed a significant contribution towards reducing communication overhead (see discussions in sections 5.5.1.2 and 5.5.2.3).

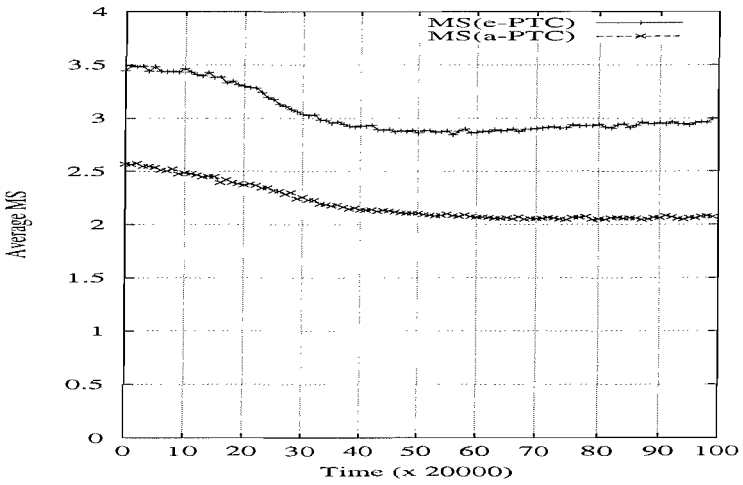


Figure 6.6: Time variation of average message size of e-PTC and a-PTC using the 50-node topology of figure 4.3(a), $p_c = 0.2$, $\Delta s = 0.1$

[6]Note that there is a slight upward trend in the MS of e-PTC in figure 6.6. However, it settles down to a value close to 3.02 after about 2.5×10^6 time steps.

[7]In fact, these messages have been observed to occupy about 25% of the total control channel bandwidth at a load of 0.8 in the topology of figure 4.3(a) using e-PTC.

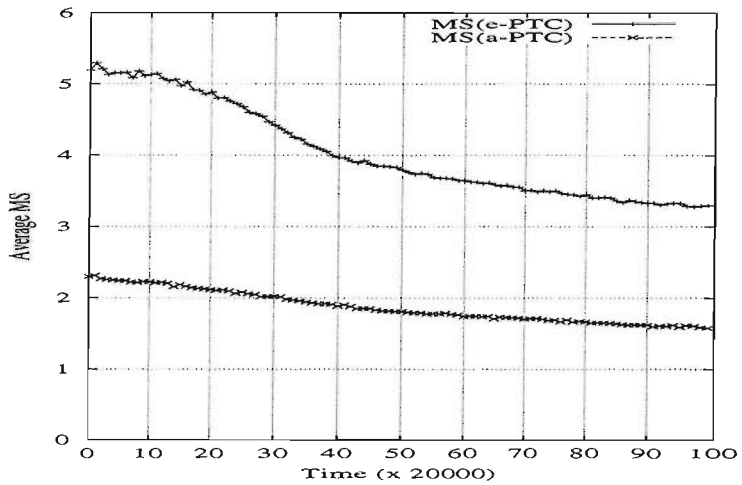


Figure 6.7: Time variation of average message size of e-PTC and a-PTC using the 100-node topology of figure 4.3(b), $p_c = 0.2$, $\Delta s = 0.1$

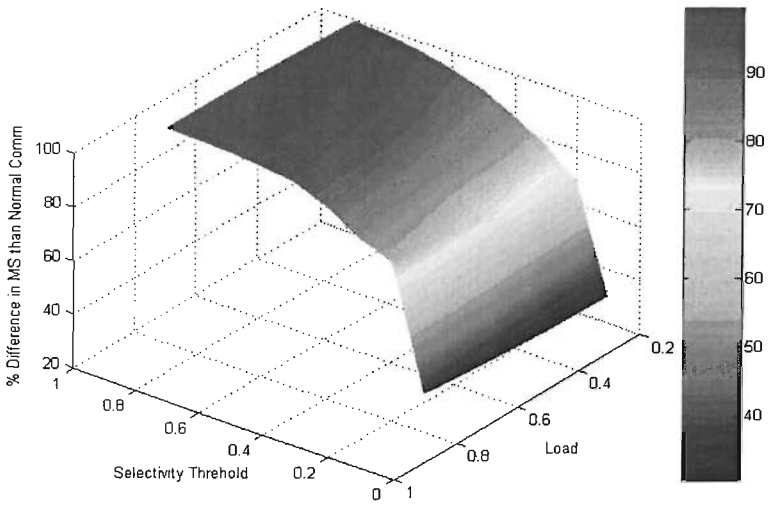


Figure 6.8: Comparing the message size profiles of a-PTC and e-PTC on the 50-node topology of figure 4.3(a)

Now the MS performances of a-PTC and e-PTC are compared across the range of call origination probability and selectivity threshold values. Similar to section 6.2.1, summary trends of the relative difference of the average steady state MS of a-PTC and e-PTC are used. Thus, for every pair of Δs and p_c values, the value of $(MS(ePTC) - MS(aPTC))/MS(ePTC)$ (where $MS(x)$ represents the average steady state message size for protocol x) is computed. In figure 6.8, these values are plotted when the topology of figure 4.3(a) is used over a surface defined by the Δs and the p_c axes. The following trends are observed:

Observation 3: *At a given call origination probability, the larger the selectivity thresh-*

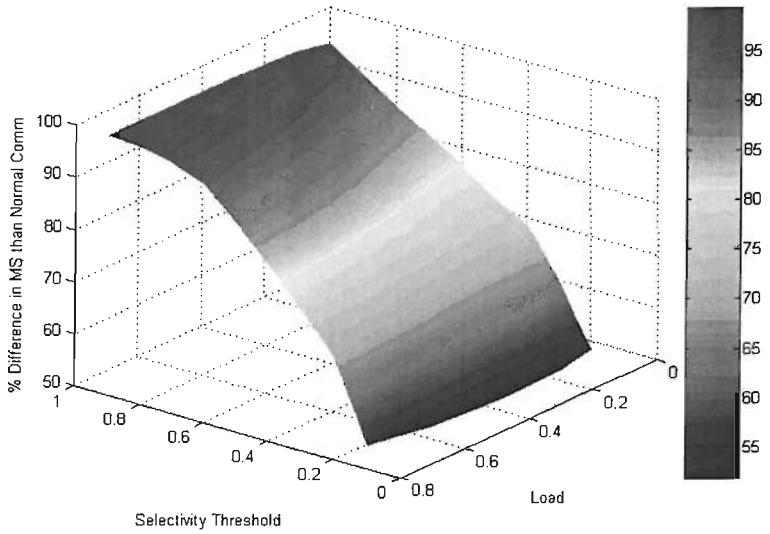


Figure 6.9: Comparing the message size profiles of a-PTC and e-PTC on the 100-node topology of figure 4.3(b)

old, the larger the relative difference in MS of e-PTC and a-PTC. This is because, at a given value of p_c, information transmission is more frequent when the value of Δs is small than when it is large (as explained in section 6.2.1). Thus, at a given p_c, the MS of a-PTC decreases as Δs increases. Hence, the relative difference in the MS of e-PTC and a-PTC increases as Δs increases.

Observation 4: *For a given selectivity threshold, a change in the call origination probability does not have any significant impact on the relative difference in MS of a-PTC and e-PTC.* The MS of any protocol increases as p_c increases because more calls originate as p_c increases and thus, more calls connect and fail leading to more information transmissions (larger MS). Thus, given a value of Δs the MS of a-PTC increases with p_c and so does that of e-PTC. However, in figure 6.8, the uniformity in the relative difference of the MS of e-PTC and a-PTC implies that the rate of increase of the MS of a-PTC is lower than that of e-PTC with increasing p_c. This is because the relative difference is measured as $(MS(ePTC) - MS(aPTC))/MS(ePTC)$, where MS(ePTC) is larger than MS(aPTC) at any given p_c (see previous results on the MS time series in this section). So, if both MS(aPTC) and MS(ePTC) increase with increasing p_c, the relative difference can be uniform (over p_c) only if the rate of increase of MS(aPTC) with p_c is smaller than that of MS(ePTC).

The trends observed in figure 6.9, which shows the same measurements using the topology of figure 4.3(b), are similar to those discussed above. These results, therefore, establish the advantage of the adaptive PTC algorithm in significantly reducing the message overhead of e-PTC.

6.3 Summary

This chapter contributes by improving the effectiveness of the PTC protocol further than that achieved in chapter 5. In particular, such an advancement is done by reducing the communication overhead of the e-PTC algorithm of chapter 5 but without sacrificing the quality of solutions generated for the sequential RA tasks. Hence, this work further establishes our thesis (as stated in chapter 1) that an effective information-sharing protocol can be used to generate high quality adaptive behaviour in sequential RA tasks in distributed systems.

The specific way of reducing the PTC message overhead adopted in this chapter is by making information-sharing an adaptive process. In the context of our application domain of network call routing, this is achieved by providing the nodes with memory to keep track of the most recent state values transmitted and received so that they can detect if transmitting the current state value will be redundant to the recipient and, in that case, not transmit it. Thus, such adaptivity results in a more controlled communication protocol in the sense that no messages are generated whenever a redundancy is detected. It ensures that the message overhead is restricted; but, since it is only when there is a redundancy that information is not shared, its effect on the system's performance in terms of successfully placing calls (i.e., the system's capability is successfully completing the sequential RA tasks) is insignificant. In this way, the effectiveness of the PTC protocol is improved further. Moreover, this chapter provides a novel advancement to the state-of-the-art in this domain by designing an adaptive mechanism for information-sharing that generates highly effective allocation performance but only incurring a low message overhead. Empirical analyses show that the simple modification (of adding a storage space which is only linear in the number of network nodes and does not increase at run time) is very effective by generating call success rates that are not significantly lower than that of e-PTC. Moreover, these studies also show that the message overhead of the adaptive algorithm is significantly lower than that of e-PTC.

Chapter 7

Conclusions and Future Work

In this chapter, we present our concluding remarks regarding the work presented in this thesis (section 7.1) and identify the avenues of future research (section 7.2).

7.1 Conclusions

In this research, we had proposed our thesis (see chapter 1) that an effective information-sharing mechanism is essential for the generation of good quality estimates and, hence, adaptive behaviour, required for efficiently processing distributed sequential RA tasks. To establish this thesis, we have used a series of theoretical and empirical studies that have provided conclusive evidence about the validity of our thesis. In order to achieve this goal, we had to overcome three important research problems (see chapter 1): (i) to design an effective information-sharing protocol; (ii) to show that this protocol can be used for estimate generation in sequential task domains; and (iii) to show that such estimates can then generate adaptive behaviour in dynamic environments involving this task domain. In addition, in so doing, our work has contributed by advancing the state-of-the-art in the field of cooperative MAS used for solving distributed sequential RA problems. In the following, we summarise the contributions of each chapter in this thesis and explain how they contribute towards establishing our thesis.

In chapter 3, we start by developing an information-sharing protocol — the post task-completion (PTC) protocol — that allows agents to acquire the values of the states they cannot directly observe in distributed systems where sequential RA tasks are executed. This protocol is both novel in its principle of application and, to the best of our knowledge, the most effective among currently available information-sharing protocols (used in efficiently solving distributed sequential RA problems) in terms of generating highly accurate estimates. In chapter 3, we prove this property of the PTC protocol using a formal analysis in a sequential RA task environment. Hence, this chapter establishes

the usefulness of an information-sharing mechanism for the generation of high quality state estimates in distributed systems — a claim made in our thesis in chapter 1.

In chapter 4, we design a set of communication heuristics based on PTC and empirically evaluate their performance in an exemplar application involving distributed sequential RA tasks — call routing in a mesh telephone network. In particular, Q-learning is used along with PTC to provide a flexible and robust mechanism for learning state estimates. The experimental results indicate significant improvements in the performance of the PTC-based heuristics compared to a variety of state-of-the-art algorithms used in this domain. Specifically, PTC is shown to achieve up to 60% improvement in call success rates, up to 1000% improvement in its capability to connect long-distance calls, and as low as 0.25 of the message overhead of the benchmark algorithms. The experiments were conducted under dynamic environments simulated by varying the network load. Thus, these results indicate that the PTC-based heuristics can adapt better than the benchmarks. These results, therefore, provide empirical evidence to confirm the theoretical claims of chapter 3. Hence, this chapter contributes by further establishing the validity of our thesis that effective information-sharing can be used for estimate generation and good adaptive behaviour.

In chapter 5, we focus on improving the performance of PTC under highly dynamic environment conditions. Note that chapter 4 has shown that PTC performs better than the benchmarks under changing load conditions (one source of environment dynamics). Chapter 5 furthers this observation by subjecting PTC to environments with even greater dynamism. This approach is significant since it helps us to confirm our thesis which states that the communication protocol should be able to generate effective adaptive response in dynamic systems. Specifically, in chapter 5, system dynamism is implemented using network failures in the call routing application, where failures can occur anywhere in the network without prior indication. To counter the effects of such events, the communication heuristics are extended to share information following failed attempts to complete tasks. While this method incurred more communication overhead (almost 3 times more), experimental results demonstrate that it performed much better than the heuristics of chapter 4 by providing greater adaptiveness when failures occur (up to 3 times faster recovery of the call success rate were observed after failures occur). More importantly, the extended heuristics were shown to be capable of performing decentralised failure diagnosis, a key functionality for automating the maintenance of decentralised systems. In particular, a diagnosis algorithm is developed for correct diagnosis of failures every time one occurs (no false negatives) and only if they occur (no false positives). Using this algorithm in an experimental study with simulated failures, it is shown to diagnose failures correctly, thereby verifying the theoretical claims of the algorithm. These results, therefore, establish that PTC is indeed capable of generating very effective adaptive behaviour in response to dynamism caused by network failures.

While chapter 5 successfully improved the performance of PTC by making it adaptive in highly dynamic systems, it achieved this advantage by generating larger messages. Now, most real applications are resource-bounded systems and, hence, a large message size can be a performance bottleneck. Thus, in chapter 6, we focus on improving the performance of PTC by reducing this message overhead but without sacrificing the advantage it achieved in chapter 5. This chapter, therefore, contributes to further confirm our thesis that effective adaptiveness can be generated by a very effective information-sharing mechanism. To this end, chapter 6 shows that by storing in memory the history of communication, the agents can communicate selectively. In particular, they can identify whether a certain communication would be redundant by comparing against the history of communication with the other agents. In case a redundancy is detected, they can refrain from communicating and, hence, save message overhead. Specifically, using empirical analyses on the call routing problem, it is shown that a significant amount of communication overhead is saved (up to 90% savings observed) without degrading the allocation quality significantly (call success rate with selective communication is observed to be within 3% of that obtained by using the heuristics of chapter 5). Thus, experimental studies have corroborated our theoretical predictions in this chapter.

In the context of the above discussion, it can be justifiably claimed that this work has established our thesis on a solid foundation supported by both rigorous theoretical analyses and extensive empirical evaluation. Thus, our thesis has a significant contribution towards extending the state-of-the-art in cooperative MAS research. Furthermore, by successfully applying our insights into practice in the call routing application, this thesis also contributes by advancing the technology currently available for this domain.

7.2 Future Work

In this section, we identify a set of possible avenues for future extensions to this thesis.

7.2.1 Effect of Network Topology on Performance

Our information-sharing algorithm, PTC, is designed in chapter 3 independent of the structure of the network on which it is evaluated. However, the empirical studies conducted to evaluate its performance in a simulated network routing problem used different network topologies. In these studies, although the general trends were broadly identical across topologies, the extent to which the performance of PTC differed from that of the benchmark algorithms was different in each topology. This observation indicates that the performance of an information-sharing algorithm is affected by network structure. Given this, we could have some general conclusions regarding the influence of topology on performance. Such insight could then be used as guidance for network

designers and users providing them with approximate bounds on the performance of a given information-sharing algorithm. To this end, we envisage the need to categorise networks based on their structural properties, such as connectivity pattern and network diameter. By doing so, we would be able to evaluate our algorithm on multiple instances of each of these categories and, thus, create profiles of the effects of topology on performance.

7.2.2 Cost of Communication

The key focus of this research has been to design an effective communication protocol that a MAS can use in practice for improving its performance in sequential RA tasks in distributed systems. To this end, we have successfully developed such a protocol (chapter 3), evaluated it using empirical analyses on an exemplar network call-routing test bed (chapter 4), and worked towards improving its effectiveness both in dealing with highly dynamic environments (chapter 5) and controlling its message overhead (chapter 6). However, in this context, we identify that communication always incurs a "cost". This is simply because sending any message is an overhead. Thus, although in chapter 6, we have introduced adaptive communication, the decision-making used to adapt communication is based on the notion of redundancy of information. While this has been demonstrated as a very useful strategy in controlling the message overhead, we envisage further improvements if communication decisions were based on, in addition to the notion of redundancy, an explicit cost measure. As an example of communication cost, we can introduce in the agents' decision-making, an expected value of the message size that it is currently transmitting. Such an estimate can be correlated with the dynamics of the system, such as the rate of call origination, traffic patterns, previous estimates about message sizes transmitted to specific destinations, among others. This would then allow the agents to make even more judicious communication decisions. It remains to be investigated if such cost-based communication decisions can reduce communication overhead while retaining good call allocation performance.

7.2.3 Other Resource Allocation Domains

Our thesis has been proposed to show that a cooperative information-sharing mechanism can be useful for estimate generation, and, thus, adaptiveness in dynamic systems processing *sequential* RA tasks. While we have been successful in establishing our thesis in the course of this research, we note that RA tasks of other modalities also exist in practice in distributed systems. Thus, among the examples cited in chapter 1, the Grid RA is not sequential. Rather, in this application, a set of computational resources are allocated for a job to successfully execute. This type of allocation, although done in a distributed fashion (involving resources across distributed systems), does not necessarily

involve a *sequence* of allocations. Nevertheless, all these applications share some common properties. These are, distributed agents, with only localised views of the system, individually process parts of an overall task. So, in all these systems, the agents need to use estimates of the non-local states for efficient task processing. Thus, to do so, cooperative information-sharing can be useful in these applications. Now, we have analysed a specific information-sharing protocol — PTC — in the context of a specific task domain — *sequential* tasks. However, the underlying motivation of sharing information is to allow distributed agents with localised knowledge to generate effective state estimates, independent of the nature of the task (sequential or otherwise). So, from the success of PTC in the context of sequential tasks, it is reasonable to hypothesise that suitable information-sharing protocols can be designed for efficient distributed task processing in other RA domains.

7.2.4 Hierarchical Control

The specification of our PTC protocol only indicates that information is to be shared among those agents that jointly complete a task (section 3.1). Now, this works well in the network routing domain which, in its current form, functions effectively as a peer-to-peer system. However, in case of much larger systems, we envisage that such peer-to-peer interactions may become somewhat inefficient. Specifically, it can incur long latencies for information to be shared across agents and, hence, affect the quality of estimates learned.

However, in many multi-agent domains, the flow of control among agents occurs in a hierarchical fashion. For example, in network routing, clusters of nodes are often used with one or more "head" nodes that act as the cluster representatives. Thus, inter-cluster routing occurs at the level of the corresponding cluster heads (Perkins, 2000). Similarly, in a multi-agent-based monitoring system used for aging people, sensor agents report to a monitor and processing centre, which, again, might report to medical personnel (Cortes et al., 2003). Such a hierarchy of control and communication mitigates the scalability issues identified before.

In this context, we envisage that using PTC to share information in a hierarchical fashion would ensure that it scales effectively to large multi-agent systems. However, a hierarchy can define different organisational structures depending on different problem instances. Therefore, it is essential to define cooperative groups in the context of hierarchical systems, such that PTC can be used to distribute information among these groups. This may then involve sharing of information among agents across different hierarchical levels.

7.2.5 Extended Representation of State Dynamics

In this thesis, we have modelled the state dynamics by representing the available bandwidth as a function of the call destination and the neighbour node to whom the call is to be forwarded. Then, for effective call allocation, the Q-estimates are learnt appropriately. But, we envisage that for very large networks, this representation can lead to long delays for estimates to all possible destinations to be learnt. This is because, information is transmitted between agents along the call paths following call successes and failures. Therefore, for large networks, this mechanism would necessitate long delays for the information to be shared (and, hence, learnt) between all node pairs. To rectify this, we envisage that a modified Q-function is needed. In particular, we can use techniques for segmenting the state-space to define "compound" states and use Q-learning on this augmented state-space. The motivation for such state aggregation is similar to that of hierarchical control: to address the dimensionality problem. In this context, we can build upon the results from other works on state segmentation for hierarchical routing (Ramanathan and Streenstrup, 1998; Shacham and Westcott, 1987).

7.2.6 Analysis of Algorithm Stability

The effectiveness of our information-sharing algorithm has been demonstrated by way of extensive empirical evaluation on a simulated call routing problem. This problem represents a complex distributed system and its behaviour is governed by several parameters: network topology, call origination rate, distribution of call sources and destinations, call duration, node capacity, information-sharing algorithm, being some of the most important ones. It is important to note that all the above-mentioned parameters affect the learning of the Q-estimates (which the agents use to route calls). Now, the central activity of this system is that of distributed agents asynchronously choosing neighbouring agents to forward calls. Note that this behaviour is solely influenced by the Q-estimates and, hence, by the above-mentioned parameters. In this context, therefore, it is important to study if such a complex distributed system always performs within "acceptable bounds". In other words, we are interested in answering the question whether this system would always remain stable. Noticing that the agents can only handle a task when they have free bandwidth (otherwise, a task is dropped and reserved resources are recovered) and the Q-estimates are strictly bounded in the range [0.0,1.0], our hypothesis is that the behaviour of each agent is always "bounded" and, thus, the system always remains stable. This hypothesis has so far been supported by the numerous experiments we have conducted where no instablity has ever occured. Nevertheless, a formal analysis of this hypothesis could be useful for strengthening the claim that our algorithm is useful in practice.

Appendix A

Evaluation of a Broadcast Protocol for Sharing Information

In chapter 1, we identified information-sharing as a necessary prerequisite for generating a MAS solution for efficiently solving sequential RA problems. We demonstrated the viability of this claim by designing a novel protocol for sharing information (in chapter 3) and then demonstrating using empirical evaluation (in chapter 4) that our protocol achieves a better solution than a range of state-of-the-art algorithms when used in a distributed call routing problem in mesh networks. In this context, a broadcast protocol of sharing information was not considered as a viable benchmark. This is because it is impractical in applications such as network call routing due to its excessive bandwidth requirement caused by indiscriminate communication. Moreover, in large distributed systems, it is the right information about the right, often partial, portions of the global state (rather than indiscriminate communication used in broadcast) that generate the desired performance.

In this appendix, we design a broadcast protocol for sharing information in the call routing application and compare its performance against that of our PTC heuristic. Our observations from this study justify the above-mentioned arguments. To this end, we first describe the broadcast protocol. Subsequently, the issue of a variable-length message queue is discussed that is used to determine how changing the amount of communication impacts performance. Then, we present experimental results comparing the performances of the broadcast protocol against that of PTC-M (as discussed in section 4.3). Finally, the observations of this study are summarised.

A-1 The Broadcast Protocol

In the broadcast protocol, each node transmits its state value to all other nodes in the network whenever there is a change in its local state. By so doing, it attempts to inform all other nodes about its new state. Note, however, that a broadcast message reaches other nodes not by point-to-point communication but by a multi-hop forwarding process. To explain in more detail, consider a node n_x with a set of neighbours $\mathcal{K}_x$ broadcasting its state s_x at time t. Thus, at time $t+1$ (assuming a receiving node in a direct communication actually receives a message after one time tick in the global clock), all nodes in $\mathcal{K}_x$ will receive s_x. Each of $\mathcal{K}_x$ will then transmit that s_x to their neighbours. Note that in so doing, s_x would be transmitted back to the originator n_x. However, it can detect that this message was being transmitted by itself before and, hence, reject it. This check is possible by including the message originator's id n_x and the origination time t along with the message.

Against this background, we now describe how the nodes using the broadcast protocol reinforce their Q-estimates. To explain this, the broadcast message, in addition to containing s_x (the originator's state), n_x (the originator's id), and t (the origination time), also contains the states of the intermediate nodes that transmitted it. The nodes append their state values to the broadcast message before transmitting it to the neighbours. This is similar to what happens while an m_c type message is transmitted from the call destination towards its source (see section 4.2). Thus, a node n_i receiving a broadcast message also receives the set of state values of those nodes along the path that the message traversed. Hence, similar to the PTC-M mechanism, n_i computes the minimum of these state values and updates its Q function. By using the same aggregation operator as PTC-M — minimum — a fair comparison of the broadcast-based learning and the PTC-based learning is ensured. Thus, if n_i receives a broadcast message, originating at n_x, from a neighbour n_j, it updates it Q-estimate for n_x and n_j as: $Q_i(x,j) \leftarrow (1-\alpha)Q_i(x,j) + \alpha R$. Here, R is the aggregate state of the path taken by the message from n_x to n_i via n_j.

A-2 Effect of Message Queue Size

A broadcast protocol requires each node to queue all incoming messages. However, we observed that even with a reasonably small topology (the network of figure 4.2) and a lightly loaded network (call origination probability of 0.1) the number of messages to be queued increased rapidly and the simulator ran out of memory on a machine with a dual 2.2 GHz AMD Opteron processor and 2GB memory. This is a clear indication of the unsuitability of using broadcast in practice due to the indiscriminate communication it uses. However, to carry out the simulation-based evaluation of the performance of

broadcast without running out of memory, we decided to use message queues with a maximum size limit. A tuning parameter like the queue size, provides us with a tool to study the impact the amount of communication has on performance. In this context, we set the maximum size of the queue used for broadcast messages as a multiple of the maximum call-channel capacity (a value of 10, see section 4.4.2). The multiple value is varied between 1 and 50; the higher the value, the closer is the protocol to pure broadcast (with no restrictions on message queue size). Obviously, the greater the size of the queue, the more memory and time it requires for the broadcast simulation to complete. On the other hand, the message queue size has insignificant effect on the memory consumption and execution time of the PTC-M protocol. This is because being a selective mechanism for information-sharing, the PTC protocol has the advantage of generating a limited number of messages. Hence, even a small message queue is able to hold all messages in PTC-M. In the following section, we discuss the call success rate performance of PTC-M and the broadcast protocols using various message queue lengths.

A-3 Experimental Results

In this section we present and analyse the experimental results obtained by using the broadcast and the PTC-M protocols on the topology of figure 4.2. Both the message queue size and the call origination probability values were varied. For a given value of the message queue size and the call origination probability, the average steady state call success rate (as defined in section 4.5) is computed for both protocols. Figure A-1 shows these values against the various message queue sizes when the call origination probability is 0.1. The following observations are made:

Observation 1: *The call success rate of PTC-M remains almost unaffected with changing size of the message queue, while that of the broadcast protocol initially increases and then stabilises as the message queue size increases.*

PTC-M is a controlled mechanism for distributing information. It allows one message to be generated per call success and that message is distributed to only the nodes on the call route. Thus, the number of information messages generated are sufficiently limited to be accommodated effectively even when the message queue size is small (corresponding to a value of 1 along the x-axis of figure A-1). So, obviously, for queues of even larger sizes, all messages of PTC-M would also be accommodated. Now, we know that the call success rate depends on how well the Q-estimates are learned, which, in turn, depends on how effectively information is shared. Since, for all message queue lengths, the information messages are effectively distributed among the nodes, their Q-learning is not affected by the varying queue length. Hence, the call success rate performance of PTC-M remains almost unaffected.

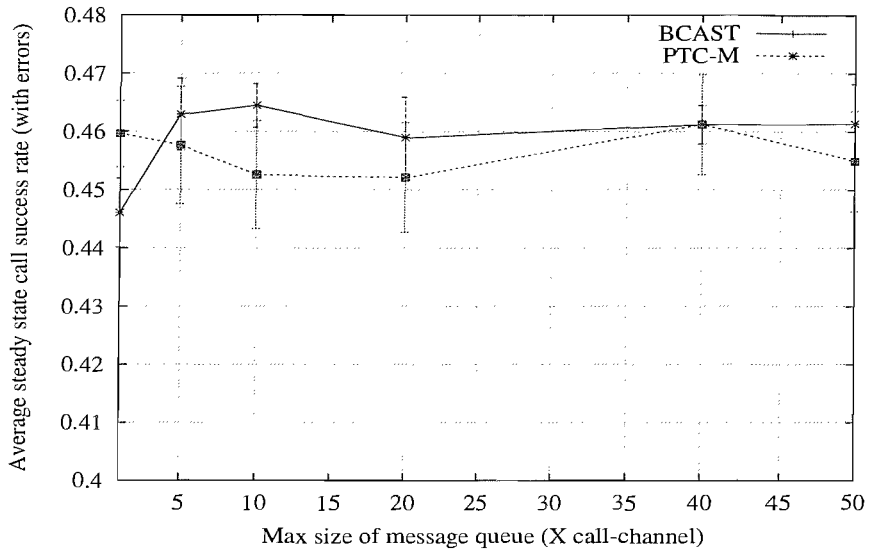


Figure A-1: Average steady-state call success rates of broadcast and PTC-M using various message queue lengths, topology of figure 4.2, load 0.1

On the other hand, the broadcast protocol generates messages far more frequently than PTC-M — every time there is a state change in any node. A node's state changes if a new call gets placed (thus, using up bandwidth) or an existing call terminates (thus, freeing up bandwidth). Now, since calls can originate on any node on a continuous basis, the broadcast protocol results in generating a high volume of messages. Therefore, when the message queue size is small, a large proportion of these messages are lost. Note that, in PTC-M, for the same value of the message queue, all information-bearing messages get successfully transmitted to the target nodes. In broadcast, however, losing messages that are targeted for every node results in a significant loss of information that could have helped a lot of nodes to learn the network state estimates. This is why the call success rate of broadcast suffers when the queue length is small. However, with larger queues, this problem is alleviated and the call success rate recovers. Also, the broadcast call success rate stabilises at large values of message queue size. This indicates that communicating more and more information (when the communication approaches pure broadcast) does not necessarily improve the network's performance. So, it is selective controlled communication (such as that of PTC) that is enough to achieve a sufficiently good performance.

Observation 2: *For small message queues, PTC-M generates better call success rates than broadcast, while the differences between them are statistically insignificant for larger queues.*

As explained previously, the large proportion of information-bearing messages lost in broadcast when small message queues are used causes the Q-learning to suffer. However, since the desired information messages are transmitted successfully in PTC-M, the nodes

learn effectively even when the queues are small. Thus, the call success rate of PTC-M is higher than that of broadcast. With larger queue lengths, however, as the broadcast can accommodate more and more messages, its learning performance improves, and so does the call success rate. In fact, we notice the broadcast call success rate to be slightly higher than PTC-M, although the differences are not statistically significant at the 95% confidence level.

In this context, it is important to note that the degree of dynamism and inherent latency in the flow of information in a distributed system dictate that simply increasing communication does not enable the agents to remain accurately up-to-date with the network states at all times. Coupled with this is the fact that task processing takes a finite time from start (the node originating the call) to finish (the call getting routed to the destination) during which *different* agents take routing decisions based on their *individual* estimates. This decentralised protocol is always offset from the ideal (and, infeasible in practice) situation where all nodes were perfectly synchronised. In this network routing scenario, the amount of communication achieved by PTC-M is shown to suffice and generate equally good performance to that of unrestricted communication. A formal analysis could provide us with a theoretical insight regarding the amount of communication that is required to achieve a certain performance in a given complex system; however, that is part of our future work. For experiments with other values of the call origination probability, similar observations were made.

A-4 Summary

In this appendix, we have experimentally demonstrated that broadcasting information among nodes is an impractical strategy to be used in real applications due to its excessive communication overhead. Moreover, indiscriminate communication does not necessarily generate better performance than our PTC algorithm. This supports our claim that in distributed, dynamically changing systems where the agents have very localised views of the global states, communicating ever more information may not improve performance more than transmitting information selectively in a timely and accurate manner.

Bibliography

I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications*, pages 102–114, 2002.

J. Baxter and P. L. Bartlett. Direct gradient-based reinforcement learning: I. Gradient estimation algorithms. Technical report, Research School of Information Science and Engineering, Australian National University, 1999.

C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 478–485, 1999.

J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 671–678, 1993.

Jeffrey M. Bradshaw. *Software Agents.* AAAI Press/The MIT Press, Menlo Park, CA, 1997.

W. L. Brogan. *Modern Control Theory.* Prentice Hall, 3rd edition, 1990.

K. M. Carley and L. Gasser. Computational organisation theory. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330. The MIT Press, Cambridge, MA, 1999.

CCGRID. The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2003), May 2003. http://www.ccgrid.org/ccgrid2003.

S. Chandler, S. Braithwaite, and H. Mgombelo. A distributed rural telephone system for developing countries. In *Proceedings of the IEE Conference of Telecommunications*, 1993.

CNES. Pleiades: Satellites and means constellation for earth observation. http://smsc.cnes.fr/PLEIADES/, 2001.

P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 35(4):487–512, 1991. Special Issue on Cognitive Science and Artificial Intelligence.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, chapter 24: Single Source Shortest Paths. MIT Press, 2nd edition, 2001.

U. Cortes, R. Annicchiarico, J. Vazquez-Salceda, C. Urdiales, L. Canamero, M. Lopez, M. Sanchez-Marre, and C. Caltagirone. Assistive technologies for the disabled and for the new generation of senior citizens: the e-Tools architecture. *AI Communications*, 16(3):193–207, 2003.

W. H. E. Davies and P. Edwards. The communication of inductive inferences. In G. Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning: Learning in Multi-Agent Environments*, Lecture Notes in Artificial Intelligence 1221, pages 223–241. Springer-Verlag, 1997.

K. S. Decker. *Environment centered analysis and design of coordination mechanisms.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1995a.

K. S. Decker. TAEMS: A framework for environment centered analysis and design of coordination mechanisms. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 16. Wiley Inter-Science, 1995b.

K. S. Decker and V. R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-agent Systems*, pages 73–80, San Fransisco, July 1995.

K. S. Decker and J. Li. Coordinating mutually exclusive resources using GPGP. *Autonomous Agents and Multi-Agent Systems*, 3(2):133–157, 2000.

B. Denkena, M. Zwich, and P. Woelk. *Holonic and Multi-Agent Systems for Manufacturing*, volume 2744/2004 of *Lecture Notes in Computer Science*, chapter Multiagent-based process planning and scheduling in context of supply chains, pages 100 – 109. Springer-Verlag, Heidelberg, January 2004.

E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, Sept - Oct 1991.

P. S. Dutta, S. Dasmahapatra, S. R. Gunn, N. R. Jennings, and L. Moreau. Cooperative information sharing to improve distributed learning. In *AAMAS-04 workshop on Learning and Evolution in Agent-Based Systems*, pages 18–23, 2004.

P. S. Dutta, C. V. Goldman, and N. R. Jennings. Efficient communication using selective information exchange in resource-constrained multiagent systems. Submitted to the

Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), 2005a.

P. S. Dutta, N. R. Jennings, and L. Moreau. Cooperative information sharing to improve distributed learning in multi-agent systems. *Journal of Artificial Intelligence Research (JAIR)*, 24:407–463, October 2005b.

P. S. Dutta, N. R. Jennings, and L. Moreau. Sharing information for Q-learning-based network bandwidth estimation and network failure detection (poster). In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1107–1108, 2005c.

E. Feinberg and A. Schwartz. *Handbook of Markov Decision Processes: Models and Applications.* Kluwer Academic Publishers, August 2001.

FloodNet. Floodnet: Pervasive computing in the environment, 2004. http://envisense.org/floodnet/floodnet.htm.

I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1998.

M. S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):140–150, January 1981.

D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, 1991.

J. R. Galbraith. *Organization Design.* Addison-Wesley, 1977.

M. A. Gibney, N. R. Jennings, N. J. Vriend, and J. M. Griffiths. Market-based call routing in telecommunications networks using adaptive pricing and real bidding. In S. Albayrak, editor, *Intelligent Agents for Telecommunication Applications — Proceedings of the Third International Workshop on Intelligent Agents for Telecommunication (IATA'99)*, volume 1699, pages 46–61. Springer-Verlag: Heidelberg, Germany, 1999.

C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research (JAIR)*, 22:143–174, 2004.

B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

N. Haque, N. R. Jennings, and L. Moreau. Resource allocation in communication networks using market-based agents. *International Journal of Knowledge Based Systems*, 18(4-5):163–170, August 2005.

Carl Hewitt. *The foundation of artificial intelligence — a sourcebook*, chapter The Challenge of Open Systems, pages 383 - 395. Cambridge University Press, 1990.

N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

M. Jin, M. Girvan, and M. E. J. Newman. The structure of growing social networks. *Phys. Rev. E*, 64, 026118:381–399, 2001.

P. Jones. *Resource Allocation, Control, and Accounting for the Use of Network Resources (RFC 1346).* Joint Network Team, UK, June 1992. `http://www.faqs.org/rfcs/rfc1346.html`.

D. Julian, M. Chiang, D. O'Neill, and S. Boyd. Qos and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks. In *Proceedings of the IEEE InfoComm*, pages 1 - 10, 2004.

J. R. Kalagnanam, A. J. Davenport, and H. S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, 1(3):221–238, July 2001.

M. Kalech and G. A. Kaminka. Diagnosing a team of agents: Scaling-up. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 249–255, 2005.

H. Kautz, B. Selman, and M. Shah. Referralweb: Combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997.

P. Klemperer. *Auctions: Theory and Practice.* Princeton University Press, 2004.

T. Krag and S. Buettrich. *Wireless Mesh Networking.* O'Reilly Wireless Devcenter, January 2004. `http://www.oreillynet.com/pub/a/wireless/2004/01/22/wirelessmesh.html`.

T. B. Lee, J. Hendler, and O. Lassila. A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. In *Scientific American.* May 2001.

M. Lemaitre, G. Verfaillie, H. Fargier, J. Lang, N. Bataille, and J. M. Lachiver. Equitable allocation of earth observing satellites resources. In *Proceedings of the 5th ONERA-DLR Aerosapce Symposium (ODAS'03)*, 2003.

V. R. Lesser and D. D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, Fall 1983.

V. R. Lesser and L. D. Erman. Distributed interpretation: A model and experiment. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 120–139. Morgan Kaufmann, San Mateo, CA, 1988.

N. Lynch. *Distributed Algorithms.* Morgan Kauffman, 1996.

J. G. March and H. A. Simon. *Organizations.* John Wiley & Sons, 1958.

J. Maynard-Smith. *Evolution and the Theory of Games.* Cambridge University Press, 1982.

R. A. McCallum. Hidden state and reinforcement learning with instance-based state identification. *IEEE Transactions on Systems, Man and Cybernetics*, Part B (Cybernetics26):464 — 473, 1996.

P. Milgrom. *Putting Auction Theory to Work.* Cambridge University Press, 2004.

N. Minar, K. H. Kramer, and P. Maes. Cooperating mobile agents for dynamic network routing. In *Proceedings of the Software Agents for Future Communications Systems.* Springer-Verlag, 1999. ISBN 3-540-65578-6.

T. M. Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Stanford University, 1978.

T. M. Mitchell. *Machine Learning*, chapter 13: Reinforcement Learning. McGraw-Hill, 1997.

L. Moreau, S. Braithwaite, N. Jennings, and D. DeRoure. Case for support: Mohican: Mobile handsets in cooperative agents network, 2001. `http://www.ecs.soton.ac.uk/-lavm/mohican/case.html`.

R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1997.

R. Nair, M. Tambe, M. Roth, and M. Yokoo. Communication for improving policy computation in distributed pomdps. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 1098–1105, 2004.

R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 705–711, 2003.

C. E. Perkins, editor. *Ad Hoc Networking.* Addison Wesley, December 2000.

L. Peshkin and V. Savova. Reinforcement learning for adaptive routing. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2002.

B. Rachlevsky-Reich, I. Ben-Shaul, N. T. Chan, A. W. Lo, and T. Poggio. GEM: A global electronic market system. *Information Systems*, 24(6):495–518, 1999.

R. Ramanathan and M. Streenstrup. Hierarchically organized, multihop mobile wireless netwoks for quality-of-service support. *ACM/Baltzer Mobile Networks and Applications Journal*, 3(1):101 — 119, 1998.

C. Rich and C. L. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the First International Conference on Autonomous Agents (Agents '97)*, pages 284–291, 1997.

T. Sandholm and S. Suri. Market clearability. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1145–1151, 2001.

N. Shacham and J. Westcott. Future directions in packet radio architectures and protocols. In *Proceedings of the IEEE*, volume 75, pages 83 — 99, 1987.

H. A. Simon. *Models of Man.* Wiley, New York, 1957.

S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995.

S. Soon, A. Pearce, and M. Noble. Adaptive teamwork coordination using graph matching over hierarchical intentional structures. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 294–301, 2004.

P. Stone. TPOT-RL applied to network routing. In *Proceedings of ICML 2000*, pages 935–942, 2000.

P. Stone and M. Veloso. Team partitioned, opaque transition reinforcement learning. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 206–212, 1999.

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy-gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7: 83–124, 1997.

R. Viswanathan and P. K. Varshney. Distributed detection with multiple sensors: Part I - fundamentals. In *Proceedings of the IEEE*, volume 85-1, pages 54–63, January 1997.

C. J. C. H. Watkins. *Learning from delayed rewards.* PhD thesis, Psychology Department, University of Cambridge, 1989.

C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8: 279–292, 1992.

B. Widrow and M. E. Hoff. Adaptive switching circuits. In *WESCON Convention Record Part IV*, pages 96–104. 1960.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229 — 256, 1992.

M. Wooldridge. *An Introduction to Multiagent Systems.* John Wiley and Sons, Chichester, England, February 2002.

P. Xuan, V. R. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 616–623, Montreal, 2001.

B. Yu, M. Venkatraman, and M. P. Singh. An adaptive social network for information access: architecture and experimental results. *Applied Artificial Intelligence*, 17(1): 21–38, January 2003.

F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.