Autonomous Mobility in Multilevel Networks

Natalia Chechina

Submitted in fulfilment of the requirements
of the degree of Doctor of Philosophy
at Heriot-Watt University
in the School of Mathematical and Computer Sciences
September 2011

Abstract

Autonomous Mobile Programs (AMPs) are mobile agents that are aware of their resource needs and sensitive to the execution environment. AMPs are unusual in that, instead of using some external load management system, each AMP periodically recalculates network and program parameters and independently moves to a new location if it provides a better execution environment. Dynamic load management emerges from the behaviour of collections of AMPs. AMPs have previously been measured using mobile languages like Java Voyager on local area networks (LANs).

The thesis develops an accurate simulation for AMPs on networks and validates it by reproducing the behaviour of collections of AMPs on homogeneous and heterogeneous LANs. The analysis shows that AMPs exhibit thrashing like other distributed load balancers. This thrashing is investigated in collections of AMPs, and two types of redundant movement (greedy effect) are identified. The thesis explores the extent of greedy effects by simulating collections of AMPs, and proposes *negotiating* AMPs (NAMPs) to ameliorate the problem. The design of AMPs with a competitive negotiation scheme (cNAMPs) is presented, followed by a performance comparison AMPs and cNAMPs using simulation.

To estimate the significance of the greedy effects the properties of balanced states are established, such as *independent balance*, *singleton optimality*, and *consecutive optimality*. The balanced states are characterised for homogeneous and heterogeneous networks where AMPs are analysed as the general case. The significance of the cNAMP greedy effect is established by conducting a worst case analysis of redundant movements, and the maximum number, and probability of, redundant movements are calculated for homogeneous and heterogeneous networks. One of three theorems proves that in a heterogeneous network of q subnetworks the number of redundant movements does not exceed $q - 1$.

The thesis proposes and evaluates a multilevel cNAMP architecture that abstracts over network topologies to effectively distribute cNAMPs in large networks. The thesis investigates alternatives for implementation of this multilevel architecture and proposes a *fusion-based scheme* where information is first available to neighbour nodes. These neighbour nodes modify the information and pass it to remote locations. The effectiveness of the scheme is evaluated by simulating networks with up to five levels, varying the number of locations from 5 to 336, and the number of cNAMPs from 8 to 3360. The experiments investigate the effects depending on the number of levels, topologies, number of locations, number of cNAMPs, work of cNAMPs, type of cNAMPs, speed of locations, and type of rebalancing. The architecture is found to be effective because it delivers performance close to the hypothetical, e.g. each additional level increases mean cNAMP completion time by just 2%.

To my Mother and Granny

Nasima Chechina *and* ***Antonina Moiseeva.***

Acknowledgements

This work would never be possible without help and belief of the people who supported me on the way to this thesis. First, I would like to thank my supervisors Dr Peter King and Prof Phil Trinder for their continued guidance and encouragement in every step of my PhD. It was very important for me to be confident that my supervisors always find time to discuss the research, give advice, read my writing and do some 'derussifications'.

I am grateful to my friends in Kyrgyzstan, Russia, UK, Canada, and many other countries who supported me throughout my PhD and gave me encouragement and strength to conduct the research. I am much indebted to my spiritual guide Fr Raphael Pavouris whose love and trust in God supported me from very first days at Heriot-Watt University, my flatmates Clair and Angus McAdam who turned a house into a home where it was always a pleasure to come back, and my dear friend Nurgul Kasenova who always finds time to talk to me whenever I call. I thank my teacher and friend Diana Antonenko whose enthusiasm and kindness were my source of inspiration.

I also would like to thank Heriot-Watt University for the James Watt Scholarship that made this research possible, and for the constant assistance from the staff of School of Mathematical and Computer Sciences (MACS), members of Dependable System Group (DSG), and Research Development Programs (RDP).

I am infinitely grateful to my dear Mother, Nasima Chechina, and Granny, Antonina Moiseeva, for their love, help, and belief in me.

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1. the thesis embodies the results of my own work and has been composed by myself
2. where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3. the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted
4. my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5. I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Autonomous Mobile Programs (AMPs) are mobile agents that improve execution efficiency by managing load; AMPs are aware of their resource needs, sensitive to the execution environment and periodically seek a better location to reduce completion time [DTM06]. To decide whether to move to another location or not AMPs use a cost model. Effective load management emerges from the behaviour of collections of AMPs.

It has been observed that distributed collections of AMPs exhibit redundant movements, like other distributed systems. In general redundant movements are a result of making locally optimal but globally suboptimal choices. To reduce the number of redundant movements different techniques are used, such as limiting any particular task to a maximum number of migrations [GA91], calculating the largest difference between the estimated execution time and the interprocess communication cost [EAEB97], applying market mechanisms [GR03]. These techniques aim to reduce the redundant movements where locations have a task scheduler or management agents, whereas the research

presented in this thesis aims to reduce redundant movements for AMPs that operate without a scheduler.

AMP behaviour has previously been measured using mobile languages like Java Voyager [Rec10] on local area networks (LANs) [DTM06]. The current research first investigates AMPs and their modification, so called cNAMPs, on LANs. A cNAMP design which is scalable for large networks is then proposed. The research presented in this thesis uses simulation as it provides a relatively fast and cheap way to emulate networks of different topology and specification.

1.2 Thesis Contributions

The research contributions of this thesis are as follows:

1. *Identifying two types of redundant movements in AMP systems, analysing their significance using theoretical and experimental techniques, and introducing a concept of negotiating AMPs* [CKT10]. The thesis shows that collections of AMPs exhibit redundant movements, as observed in other distributed systems, and distinguishes two types of so called greedy effect. The significance of the AMP greedy effects is analysed to show that although greedy effects have limited impact on networks with a small number of AMPs, few locations, or small AMPs, their effects increase as any of these factors scale. The analysis of the redundant movement types and the reasons they occur show that redundant movements are mainly caused by location thrashing. To reduce the number of redundant movements the negotiating AMPs are proposed and competitive Negotiating AMPs (cNAMPs) are designed (Chapter 4).

2. *Establishing balanced state properties, and investigating the cNAMP greedy effect* [CKT11]. To analyse the significance of the greedy effect in cNAMPs balanced state properties are investigated including *independent balance*, *singleton*

optimality, and *consecutive optimality*. Balanced states are also characterised for homogeneous and heterogeneous networks. The significance of the cNAMP greedy effect is established by conducting a worst case analysis of redundant movements, and the maximum number, and probability of, redundant movements are calculated for homogeneous and heterogeneous networks. One of three theorems shows that in a heterogeneous network of q subnetworks the number of redundant movements does not exceed $q - 1$ (Chapter 5).

3. *Assembling a set of consistent and rigorous definitions* about the behaviour of collections of AMPs and cNAMPs. These definitions are essential for reasoning about AMP/cNAMP behaviour. Some of definitions given in earlier papers are generalised [Den07, CKPT09] (Glossary).

4. *The design and implementation of an architecture supporting multilevel networks*. To allow effective cNAMP distribution in large networks a multilevel architecture that abstracts over network topologies is introduced. The thesis evaluates alternatives and implements a fusion-based scheme in which a node collects information from the lower level and provides only summary to the nodes of the same and upper levels (Chapter 6).

5. *Evaluating the architecture of multilevel networks*. The effectiveness of the fusion-based multilevel architecture is evaluated by estimating cNAMP completion time and number of redundant movements. The networks with up to five levels are simulated. The results show that each additional level increases the mean cNAMP completion time by 1%–3%. Although cNAMPs may have a large number of redundant movements in multilevel networks these movements do not affect completion time dramatically. The mean completion time in a five level network differs from the hypothetical value by 12% (Chapter 7).

1.3 Publications

- Natalia Chechina, Peter King, and Phil Trinder. Redundant Movements in Autonomous Mobility: Experimental and Theoretical Analysis. *Journal of Parallel and Distributed Computing*, 71(10):1278–1292, Elsevier, 2011.

- Natalia Chechina, Peter King, and Phil Trinder. Using Negotiation to Reduce Redundant Autonomous Mobile Program Movements. In *IAT '10*, pp. 343–346, September 2010. Toronto, Canada, IEEE Computer Society.

- Natalia Chechina, Peter King, Rob Pooley, and Phil Trinder. Simulating Autonomous Mobile Programs on Networks. In *PGNet'09*, pp. 201–206, June 2009. Liverpool, UK.

Chapter 2

Background

We can hardly imagine the modern world and our every day life without the Internet and advantages provided by computer networks, such as remote access, virtualization distant computers, data storage. Given the scale and complexity of the present networks autonomy is a promising means to automatically configure complex systems and distribute work among computers. This thesis investigates the effectiveness of Autonomous Mobile Programs (AMPs) as a mechanism to manage load in LANs (Chapters 3 and 4) and large multilevel networks (Chapters 6 and 7).

This chapter introduces key ideas for the understanding of AMPs and their behaviour. It starts with a discussion of computer networks (Section 2.1), and then provides information about mobility (Sections 2.2) and autonomy (Section 2.3). Next the chapter covers AMPs (Section 2.4) and discusses load management classification indicating AMP place in taxonomies (Section 2.5). The novelty of AMPs and their features are compared with other mobile agent load managers (Section 2.6), and as the research is conducted by means of simulation, network simulators are also discussed (Section 2.7). The chapter concludes with the review of the main findings (Section 2.8).

2.1 Networks

A network is a set of interconnected computers. Networks are classified according to the number of computers, their interconnections and geographical position. These factors affect transfer rate and inputs of telecommunication lines. Network scales are discussed in Section 2.1.1 and topologies are discussed in Section 2.1.2.

2.1.1 Network Scale

The literature identifies different network scales, e.g. [Tan03, Mac98]. Tanenbaum's classification [Tan03] of local area networks, wide area networks, and internetworks is found to be the most relevant to the current research.

Local Area Network. A local area network (LAN) is characterized by a relatively small geographical range, e.g. an office, a building or a small group of buildings [Nun92]. Computers are usually distributed within a few kilometres and have a relatively high data rate of 10 Mbps to 10 Gbps. A LAN is commonly used to share data and to exchange information. An example of a LAN is Ethernet.

Wide Area Network. A wide area network (WAN) covers a larger physical area than a LAN, such as a country or a continent [LE96]. It unites machines (hosts, locations) with a communication subnet. The machines are operated by users, while communication subnets are commonly operated by telephone companies or Internet service providers. A subnet typically consists of transmission lines to move data between hosts, and switching elements to link transmission lines together. Switching elements are also called routers. The speed of data transmission is usually less than within a LAN and typically ranges from 1.2 Mbps to 156 Mbps.

Internetwork. An internetwork combines a set of LANs and WANs with different hardware and software [MvS95]. This interconnection is effected by means of gateways. The gateways are specialised routers that link different networks and perform software conversion of traffic between different protocols. In contrast to a WAN, an internetwork is formed by connection of distinct networks. Another feature of an internetwork is that it has a global scale. These two properties enable computers with different architectures to communicate with each other at arbitrary distances. An example of an internetwork is the Internet.

Multilevel Network. By a multilevel network is meant a hierarchical network abstraction where processing elements are divided into groups. The reason to use a multilevel network notion is to abstract from both the network scale and the type of processing elements, i.e. to emphasise that the designed scheme is developed not for a particular type of machines or networks but rather is a general scheme. Thus, we can think of a processing element as a core in a multicore computer, or a processor in a multiprocessor architecture, or a single processor computer (location). Network scale is also not limited due to the hierarchical architecture, and may vary from a multicore computer and a LAN to an internetwork.

Multilevel networks are widely used in network managements systems to enable implementation of scalable, robust and secure networks [TF95]. Multilevel networks can be constructed in a number of ways, e.g. [Zho10] distinguishes two types of network elements within each level: network management servers that carry connection with nearest levels, and executing elements, such as laptops and printers; whereas [STD02] uses a multilevel network as a prototype for a fully connected network of processors to carry diagnosis algorithms.

In the current research processing elements are divided into groups and form $Level\ 0$. The groups are connected with each other via gateways that form the remaining levels. In the current research the number of levels depends on the dispersion of interconnection delays. Detailed discussion of simulated multilevel networks is provided in Chapter 6.

In the current thesis by a location is meant a single processor computer in a singlelevel network, and a processing unit in a multilevel network, e.g. a single processor computer or a processer in a multiprocessor computer. By a node is meant a location in a singlelevel network, and a location or a gateway in a multilevel network.

2.1.2 Network Topology

A network topology is a configuration of physical links [OO06]. The most commonly used topologies are star, ring, tree and mesh. These are also called typical topologies. The choice of a topology is determined by a range of factors, such as extensibility, reliability, and cost considerations. Small networks usually have one typical topology, and large networks assemble a set of connected typical topologies. The current thesis aims to investigate AMP distribution and AMP completion time but not precise paths of movements. Therefore, only fully connected LANs, and fully connected multilevel networks are considered, i.e. hierarchical topology where all children nodes of the same parental node are interconnected. This interconnection is an abstraction and only identifies an existence of a way to transmit data between locations. Detailed discussion of multilevel topology is presented in Chapter 6.

2.2 Mobility

The term *mobility* may refer to various concepts depending on the context. In a hardware technology context mobility implies mobile devices that can be moved, such as a laptop or a wireless PDA. In a software technology context mobility implies mobile computation over a network. To distinguish between the types of mobility the following terms are used: *mobile computation* for software mobility and *mobile computing* for hardware mobility [Car99]. In the current research *mobility* implies a technique that provides active process migration within a network [FPV98]. Thus, only an area of mobile computa-

tions is discussed in this thesis as its focus is on simulating autonomous mobile program behaviour.

An ability to move executing processes within a network has the following advantages [MDW99]:

- *Load distribution* allows processes to move from a heavily loaded node to a less loaded node to decrease process completion time.
- *Fault resilience* provides processes an opportunity to migrate away from a partly broken location and to overcome local errors.
- *Data access locality* allows programs to reduce data access time by migrating closer to the data.

Mobile computations mechanisms can be classified using a taxonomy presented in Figure 2.1 [FPV98]. Depending on the entity that makes movement decisions a mobile computation approach can be either implicit or explicit [Boi06]. In *implicit mobile computation* a program decides itself whether to migrate or not using some algorithm. In *explicit mobile computation* a programmer controls computation placement in a network. AMPs are based on the implicit mobile computation approach and make all movement decisions themselves.

Implicit mobility can be based on mechanisms that support either a data space rearrangement *or* code and execution state mobility. AMPs require code and execution state management mechanisms.

A code and execution state management supports two forms of mobility: weak and strong. *Weak mobility* only supports code movement. Examples of programming languages that support weak mobility are *m*Haskel [BTL05] and Java Voyager [Rec10]. *Strong mobility* supports code, execution state, and data movement that allows a program to resume its execution from the place it was stopped on the previous location. Strong

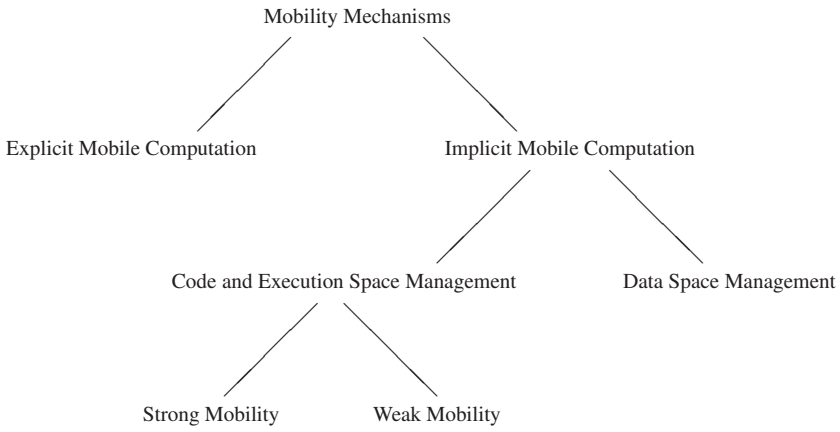


Figure 2.1: A Classification of Mobility Mechanisms [FPV98]

mobility is implemented in such programming languages like MobileML [HY00], JoCaml [MM10], and JavaGoX [SSY00]. AMPs have previously been investigated using mobile languages with both weak mobility like Java Voyager and strong mobility like Jocaml.

2.3 Autonomic Systems

The main property of an autonomic system is a capability of self-management. The self-management aims to administer a system regardless of operating system and configuration details. The four aspects of self-managing systems are as follows [KC03]:

- *Self-configuration* implies an automatic configuration of autonomic systems. A system is able to configure and register itself in the execution environment. This allows other systems to use the knowledge and to modify their own behaviour. Self-configuration is crucial for large and complex systems that require significant time and programmer resources to configure.
- *Self-optimization* is an ability to seek opportunities to enhance system efficiency. The efficiency is expressed either as a performance or as a cost. A decision can be made using a cost model, or an algorithm, or knowledge received from previous experience and learning.

- *Self-healing* implies a system ability to diagnose itself and expose bugs in software or hardware. The results can then be used to eliminate defects, e.g. installing the missing patches, or alerting a human programmer.
- *Self-protection* is a system ability to defend itself against outward attacks and inner cascade failures. Self-protection also implies making decisions using the previous system reports.

AMPs implore a self-optimization aspect, i.e. AMPs continuously analyse the network capacity to improve efficiency and reduce completion time. Other examples of self-optimising systems are [MB03, SKL89, MO04]. The decision making steps in autonomic self-optimizing systems are as follows [LWZ05]:

- *Monitoring* state information.
- *Deciding* whether to change the current state or not.
- *Executing* the decision.

2.4 Autonomous Mobile Programs

Autonomous mobile programs (AMPs) have been developed to manage load on large and dynamic networks [DTM06]. AMPs migrate within a network to raise program execution efficiency and exploit the network capacity. Most autonomous mobile agent systems adapt their computations, however AMPs adapt their coordination, i.e. *where* the program executes and not what it does.

AMPs are not only a type of autonomous agent but are mobile agents with a cost model. They are similar in conception to ethological models such as ant algorithms [MO04]. However, the difference is that ant algorithms seek the fastest path and use feedback from other mobile agents to choose the best path, whereas AMPs seek better resources

and estimate the current network information. AMPs aim to minimize completion time by seeking the most favourable location without visiting any specific one of them. AMPs are aware of network parameters and their own resources, and use this information to migrate in a network. The difference between AMPs and other mobile agent load management systems is discussed in Section 2.6.

In general any program can be encapsulated in an AMP skeleton. The research presented in the thesis considers only CPU bounded programs as it is difficult to justify moving an Input/Output (I/O) bound program away from the data it is operating on. The I/O bound program is expected to pause for I/O and, thus, lessen the load it imposes on the location. The most beneficial AMP technique will be from programs that have small size of disc space but require a lot of computation power to be executed. For example, a program that is based on recursive computations [pen08] or generic evolutionary algorithms [oxf08]. The AMPs mostly studied in the thesis perform square matrix multiplication. Experiments with other programs, e.g. coin counting and ray tracing, show consistent behaviour. Disc space of these three programs does not depend on the remaining work. Thus, in the beginning of execution, an AMP transfers as much code as in the end of its execution.

The key equations of the AMP cost model defined in [Den07, Section 3.3.2] are repeated here. Table 2.1 provides a list of the main parameters and their definitions. To estimate the program completion time, T_{total}, AMPs use the following cost model:

$$T_{total} = T_{Comp} + T_{Comm} + T_{Coord}, \tag{2.1}$$

where T_{Comp} is the computation time, T_{Comm} is the total communication time, and T_{Coord} is the total coordination time. The cost model is parametrised on the system architecture that includes location and interconnect speeds, data processing and communication costs, data size, and number of locations.

To reduce AMP coordination time each location has a *load server*. The load servers only collect state information, and we can think of them as blackboards. The load server initial design and its modification are further discussed in Chapter 4.

d	Dimension of square matrix
$gran$	Fragment of work that must be executed between searches for a better location
O	Overhead
R	AMP relative speed
S	Available speed
T_1	Execution time of one unit
T_{comm}	Time for single communication
T_{coord}	Coordination time in the load server architecture
T_{gran}	Execution time of fragment of work $gran$
T_h	Execution time on the current location
T_n	Execution time on the new location
T_{send}	Time to transmit an AMP to the new location
W	Work to execute
x_{loc}	Number of AMPs on a location

Table 2.1: Parameter Definitions

Available speed, S, is the execution speed of a single AMP on a location, i.e.

$$S = (CPU speed) \cdot (1 - non\ AMP\ load), \tag{2.2}$$

is used to differentiate the total resources of a location from the resources available for AMPs. AMP load is the number of AMPs on a location. The current research, like the previous AMP investigations [Den07], assumes that all resources of a location are available for AMPs, and the CPU speed coincides with the available speed, except in the case of the root location where the external load is higher. By the *root location* is meant the location where all AMPs start[1].

Work to execute, W, is calculated on the basis of the program coding, and is measured in units. For example, square matrix multiplication is implemented using a triple loop. Thus, the total work of a matrix multiplication program is $W \propto d^3$ where d is a dimension of a square matrix. Here, a unit consists of three operations: one operation of multiplication, one operation of addition and one operation of assigning [Den07, p. 56]. Remaining work, W_r is calculated by subtraction of the executed work from the total

[1] In [DMT10] the *root location* is also called either *initiating location* or *first location*.

work. In the simulation no real matrix multiplication is performed but only an abstraction where AMP work gradually decreases depending on the executing speed.

An *AMP relative speed*, R, is the available speed equally divided between the AMPs at the location, x_{loc}, i.e.[2]

$$R = \frac{S}{x_{loc}}. \tag{2.3}$$

An AMP executes a part of work, $gran$, before it tests the relative speeds of other locations to see if a move will improve its completion time. $gran$ is measured in units of work per recalculation, and depends on remaining work, W_r, and the number of recalculations, n [Den07, p. 70]:

$$gran = \frac{W_r}{n},$$

where number of recalculations, n, is a ratio of overhead, O, of the time for static program running on the current location, T_h, to the coordination time, T_{coord}.

$$n = \frac{O \cdot T_h}{T_{coord}}.$$

This allows to control the time an AMP spends on parameter recalculation. Thus, if an AMP never moves from the root location its overhead is within 0.

In turn T_h is remaining work, W_r, divided by relative speed at the current location R_h [Den07, p. 54]:

$$T_h = \frac{W_r}{R_h} = \frac{W_r \cdot x_{loc}}{S_h}$$

Thus, from the above three equations $gran$ can be calculated as follows:

$$gran = \frac{T_{coord} \cdot S_h}{O \cdot x_{loc}}. \tag{2.4}$$

The estimation of AMP completion time is discussed in Section 2.4.1. The time parameters for the simulation are taken from Java Voyager AMP measurements on a LAN [Den07].

[2] In [Den07] the *available speed* is called *relative speed*, and the *AMP relative speed* is called *average relative speed*.

The coordination time, T_{coord}, was determined experimentally to be 0.011s for a load server architecture. The acceptable overhead O is taken to be 5%. After executing for T_{gran} an AMP makes a request to the load server of the current location about states of other locations in the network. If an AMP decides to stay on the current location, then it continues execution for a further T_{gran} seconds, otherwise it moves to a new location taking T_{send} seconds.

The main rule on the basis of which AMPs make a decision to move to a new location is whether execution time on the current location, T_h, exceeds execution time on the next location, T_n, and communication delay, T_{comm}:

$$T_h > T_n + T_{comm}. \tag{2.5}$$

If condition (2.5) is satisfied, then an AMP moves. Here, communication time is time to transfer an AMP, i.e. $T_{comm} = T_{send}$.

2.4.1 Program Execution Time Prediction

The AMP cost model is based on knowledge of the remaining execution time of the program. The current research does not focus on the method or effectiveness of the prediction mechanism assuming that such a mechanism is implemented in AMPs and accurate predictions are provided.

The experiments presented in [Den07] use a simple time prediction mechanism. The time is calculated on the basis of remaining work and location execution speeds. To calculate the execution time of the whole program an AMP evaluates the execution time of one unit, T_1, and multiplies it by remaining work W_r, i.e. $T_{Comp} = T_1 \cdot W_r$ [Den07, p. 69]. AMPs are not limited to a particular type of execution time predictor. In general any technique that allows estimation of remaining execution time on different locations can be used, e.g. approaches based on historical information [YP98], probabilistic modelling [WhS03], case-based reasoning [NNMNA$^+$05].

2.4.2 Assumptions

Like the previous AMP investigations, the current research is based on the following three assumptions: system reliability, equal sharing of the CPU power, and sufficiency of resources.

System reliability implies that locations do not fail or disappear from the network. Communication lines are reliable and in case of some disruption data can be transferred using alternative routes. Possible packet retransmission and rout diversion are taken into account in calculation of transfer delays. AMPs are also considered to be reliable and do not fail to execute their work. Another assumption is that AMPs *equally share CPU power*, i.e. all AMPs are treated equally and while being on a location AMPs constantly conduct execution spending no time on input/output. *Sufficiency of resources* implies that when an AMP checks a possibility to move to another location it only checks execution speed on the remote location and the time it would take to transfer to that location but is not interested in available space of hard disc or available RAM of that location. These resources are assumed to be sufficient for successful AMP execution.

Some of the assumptions such as location and AMP failure and sufficiency of resources are planed to be weakened in the future research to investigate their impact on AMP performance (Sections 8.3); whereas programs that require much input/output are currently considered out of AMP scope and no future investigation in this area are planed. More discussion on the thesis limitations and future work towards weakening the assumptions are discussed in Sections 8.2 and 8.3 respectively.

2.5 Distributed Load Management

Being a promising technology to exploit computer resources and accelerate program completion, load management has occupied research interest for many years [LK87, SBK06, MI08]. The main difficulties load management approaches face are minimizing

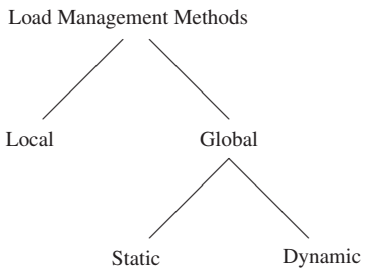


Figure 2.2: Classification of Load Management Methods [CK88]

execution time, minimizing communication delays, and maximizing resource utilization [SKH95].

The AMP approach is first classified using the general taxonomy from [CK88] presented in Figure 2.2. Then a taxonomy of dynamic task scheduling schemes [Rot94] is also used to provide further classification of dynamic load managers.

2.5.1 Local and Global Load Management

The first distinction is between local and global load managers [CK88]. The difference is in the number of processors that take part in load management. A *local load manager* implements task scheduling for a single processor, i.e. load management is achieved by scheduling a particular process within a processor [CHLE80]. A *global load manager* solves the problem of *where* a task must be executed, leaving local process scheduling to the operating system of the processor to which the task was allocated. Such cooperation of local and global scheduling increases the autonomy of individual processors in global load management.

Only global load managing classification and approaches are further examined in this section as the aim of the current research is to achieve load balancing in a network.

2.5.2 Static and Dynamic Load Management

Global load managers are divided into two types: static and dynamic [CK88]. Whether a global load manager is static or dynamic depends on the time of task scheduling, i.e. either before or during program execution. Both methods have a range of applications but efficiency depends on the data available to the scheduler.

A *static load manager* assigns tasks to locations before a program starts its execution. Therefore, all necessary information for a predictive calculation must be known beforehand, e.g. task execution times, processing resources. The fundamental drawback of static methods is the unavailability and quality of such information in many cases that often results in a poor distribution. One of the main aims of a static scheduling is to reduce program completion time at the cost of communication delays [SKH95]. This can be achieved by either allocating processes to processors or consolidation small task into larger groups. An example of static load management system that aims to minimise mean job response time is presented in [TT85].

A *dynamic load manager* in contrast takes scheduling decisions during program execution. The movement decisions are made on the basis of system-state information, and hence cannot be made before the program starts. The advantage of dynamic load managing is that program consumptions are not required to be known beforehand. The process of dynamic load managing in general entails five phases [WT98]:

- *Load evaluation* - analysing load of processor units to estimate the degree of network imbalance.
- *Profitability determination* - detecting load imbalance and initiating a balancing process.
- *Calculation of a work transfer vector* - calculating necessary transfers to establish a balance.

- *Task selection* - preparing selected tasks to transfer.
- *Task migration* - transferring tasks from one processing element to another.

Other important requirements for a dynamic load managing algorithm are the ability to employ local communications and to make fast recalculations [RN04]. A local communication exhibits smaller communication delays which allow reduction of program execution time and improvement of efficiency. Fast calculations to relocate tasks are required to keep overheads within a reasonable value. This is especially important when a program does not move during the whole execution but stays on the initial processing element.

AMPs belong to the class of dynamic load managers as they periodically recalculate network parameters to reduce completion time. To further classify AMPs a dynamic load managing taxonomy from [Rot94] is used. The taxonomy is based on [CK88] but provides a more detailed analysis of different dynamic load managing characteristics. This taxonomy distinguishes state estimation and decision making functions of a load manager and classifies them separately. The system state estimation and the decision making are discussed in Sections 2.5.3 and 2.5.4 respectively.

2.5.3 System State Estimation

The system state estimation analyses *where* state information is collected, *with whom* it is shared, *who* initiates the exchange and *how often* it happens [Rot94]. In a formal way system state estimation, ξ, is denoted as follows:

$$\xi \in \{(Centralised, Decentralised, Hybrid) \times (Complete, Partial, Variable) \times$$
$$\times (Voluntary, Involuntary, Composite) \times (Periodic, Aperiodic, Combination)\}.$$

The detailed discussion of each parameter is provided below.

Centralized, Decentralized, Hybrid. The first group of parameters defines *where* state information is collected.

A *centralized scheme* implies a central agent that can be a physical processing element or a globally shared file. The central agent collects system state information and, therefore, must be accessed and updated by every processing element of the system. An advantage of this method is a low estimation overhead [CLHZ97]. Another advantage is a reduced communication delay which is due to 'all-to-one' communication scheme. However, a substantial defect of this method is a poor scalability, i.e. the increase of the processor number leads to the increase of the computational load and bottleneck on the central agent. Examples of the centralized information collection are a load-balancing policy with a centralised job dispatcher LBC [LR92] and a computing and communication system Andrew that was created as a result of collaboration between IBM and Carneigie-Mellon University [MSC+86].

In a *decentralized scheme* a state information collecting element is replicated on every node of the system, e.g. each location collects and estimates network state information. A decentralized organization is more scalable and more reliable compared to a centralized one. An example of decentralized information collection is a Grid computing system Messor [MMB03].

A *hybrid scheme* combines centralized and decentralized approaches. Developers compromise and derive as many advantages from both approaches as possible. Hybrid schemes are mainly based on two structures: Globally Decentralised, Locally Centralised (GDLC) and Globally Centralised, Locally Decentralised (GCLD). In a GDLC structure all locations are divided into clusters on the basis of various factors. A centralised organisation is used within a cluster, and a decentralized organisation is used between the clusters. This structure is usually used in large scale networks. An example of GDLC structure is load sharing system Utopia developed at the University of Toronto [ZZWD93]. In a GCLD structure locations exchange state information with their neighbours in a decentralized manner but also send their local state information

to the central monitor. An example of GCLD structure is a decentralised job scheduler based on Bayesian decision theory presented in [Sta85].

In the AMP approach information collection is implemented by load servers on each location in a decentralised manner. In a multilevel organisation each location collects information from sibling locations and the gateway (i.e. upper level), a gateway in tern collects information from its sibling, parental, and daughter nodes also in a decentralised manner (Chapter 6). Thus, AMP information collection has a decentralised scheme.

Complete, Partial, Variable. The second group of parameters defines *with whom* locations share their state information. Depending on a number of nodes that take part in a state information exchange a system state estimation can be either complete, partial, or variable. In a *complete* approach all nodes exchange state information with each other, whereas in a *partial* approach a node exchanges its information only with a subgroup of nodes. A *variable* approach combines complete and partial approaches. In the AMP implementation on a LAN a load server exchanges state information with all other load servers, and in the AMP multilevel implementation a gateway exchanges state information with sibling, parental, and child nodes (Chapter 6). Therefore, AMPs use a partial approach.

Voluntary, Involuntary, Composite. The third group of parameters defines *who initiates* the information exchange process. In a *voluntary* approach a node sends state information on the basis of its inner conditions, whereas in an *involuntary* approach the information is sent as a response to a request from another node. A *composite* approach is a combination of voluntary and involuntary approaches.

In the initial AMP implementation load servers exchange state information in an involuntary manner. However, this was modified and a cNAMP approach introduced in Section 4.4 uses a voluntary scheme.

Periodic, Aperiodic, Combination. The fourth group of parameters defines how often the information is exchanged. AMPs use a periodic scheme.

2.5.4 Decision Making

The decision making function examines *where* a movement decision is made, *who* initiates it, and *what main rule* it uses. In a formal way the decision making, δ, denoted in [Rot94] is as follows:

$$\delta \in \{(Centralised, Decentralised, Hybrid) \times (Sender\ initiated, \\ Receiver\ initiated, Symmetric) \times (Simple, Model\ based)\}.$$

The characteristics of each parameter are discussed below in this section.

Centralised, Decentralised, Hybrid. This group of parameters defines *who* and *where* makes movement decisions. A *centralised* load management implies scheduling by a single server, whereas in *decentralised* load management this function is distributed among several nodes. A *hybrid* scheme combines centralised and decentralised schemes. AMPs belong to the extreme version of the decentralised decision making scheme as each AMP decides itself where and when to move.

Sender Initiated, Receiver Initiated, Symmetric. A decision making policy which is also called a transfer policy depends on *who* initiates a migration. In a *sender initiated* approach a task or task maintainer searches for a remote location with better parameters. In a *receiver initiated* approach a node searches for work to execute. A *symmetric* approach combines both sender and receiver initiated approaches. AMPs use a sender initiated approach, i.e. each AMP periodically recalculates its parameters to seek for a better location.

Simple, Model Based. The last group of parameters defines the type of transformation, i.e. the way the best location for the further execution is chosen. A *simple* transformation is performed by the first fit and the best fit approaches. A *model based* transformation involves more complicated computations using such approaches as stochastic learning automata, Bayesian decision theory. To reduce the recalculation cost AMPs use simple transformation with the best fit approach, i.e. an AMP chooses a location that can provide the minimum execution time adjusted for communication time.

Thus, AMP state estimation, ξ, and decision making, δ, functions are classified as follows:

$$\xi = \{(Hybrid) \times (Partial) \times (Involuntary) \times (Periodic)\}$$

$$\delta = \{(Decentralised) \times (Sender\ initiated) \times (Simple)\}$$

2.6 Mobile Agent Load Management

Researchers have investigated different approaches to manage load effectively in a network using mobile agent techniques. Some important techniques are probabilistic, market-based, community-based, and biologically inspired approaches.

In a *probabilistic* approach the movement decisions are made on the basis of a probability [OGP98, SBK06]. One of the first mobile agent load balancing systems that uses this approach is FLASH [OGP98] designed for heterogeneous clusters. The balancing is implemented by means of agents, i.e. node, system, and user agents. User agents make movement decisions using local and global information provided by node and system agents respectively.

A well-known *market-based* approach representative is Dynasty [BPZ96]. The load is performed by means of brokers that are organized in a hierarchical manner. Each task has its own funds which are supplied by the root task. To migrate to another location a task must be able to cover such expenses like rent for utilising computer resources, brokerage

for getting assigned to a target host, fees for migration and transport services. The more levels of broker hierarchy are involved and the larger the location selection range, the higher price for task assignment. The request for brokering is initiated by a task and contains information about amount of money the task is willing to pay, the task computation costs, and the list of requirements concerning the desired target host. Examples of other load balancing systems that use a market-based approach are [WHH+92, KKP+04]

An example of a *community-based* approach is a Community-based Load Balancing system (CLB) [MI08] where agents in the system have an analogy with humans. Agents are placed according to a community they belong in to minimize inter-agent communication time, i.e. when agents from different servers communicate with each other the servers exchange information to improve the balance and if possible to place the agents at the same server. CLB deals with two problems: placing frequently communicating agents as close as possible to reduce communication time *and* balancing load between the servers.

Biologically inspired approaches reflect biological processes and animal behaviour on mobile agents to manage load, e.g. ant colonies in Traveler [WX99], bee honey collecting in MATS [GHCN99], cloning in [CB04]. To discuss biologically inspired approaches MATS is taken as an example. The MATS agents cooperate as a team where each agent performs a particular role: Hive, Queen, Scout, or Worker. Hive is the initial program which interacts with a user. Hive decomposes the initial program into processes, Queens. A Queen is a mobile agent that implements a part of the whole program, and Scouts are mobile agents that collect information. The Scouts are dispatched by the Hive. The workers perform some part of computation and are dispatched by the Queen.

AMPs belong to the community-based approach. AMPs differ from other mobile agent systems designed to balance load in that each AMP is both autonomous and self aware, i.e. it knows key information like remaining execution time and program size. Another difference is that the approach does not split a program into subtasks as in [MI08, KKP+04, CB04]. An AMP is the whole program. Thus, AMPs represent a radical point in distributed decision making when the agent itself decides where and when to move

rather then the decision being taken by a location server [BPZ96], a load balancing coordinator [SKA06], or a cluster manager [KKP+04].

2.7 Network Simulation

The research presented in the thesis is conducted using simulation. To analyse AMPs such simulation advantages like on demand access to arbitrary number of locations for unlimited time where locations may have any required capacity are employed. Other important simulation advantages are the possibility to repeat experiments using the same random seeds and the ability to manage simulation pace, i.e. from slow-motion to express. Thus, to conduct the research one of the key tasks is to find a suitable simulator. This section examines properties of a desired network simulator (Section 2.7.1), explores the classes of simulators (Section 2.7.2), and discusses the features of the selected simulator, i.e. OMNeT++ [Var10] (Section 2.7.3).

2.7.1 Properties of Network Simulation

The main properties the simulator should possess to analyse AMP behaviour on a network are as follows:

Discrete event simulation. The simulator must provide processing of discrete events to simplify the model and to allow an accurate analysis. Continuous event simulation is not practical in the current research due to the complexity of the system.

High level network simulation. As the aim of the current research is to analyse AMP behaviour in a network, the simulator should be able to simulate processes at an application level and abstract the lower level details of the network.

Reasonable time of completion. An opportunity to evaluate results on the basis of multiple runs is one of the most important simulation modelling properties. However, if

each run takes too much time to execute, the process of data collecting becomes time consuming. Therefore, a capability of reasonable completion time is critical to evaluate and further improve a simulation model.

Free Software. Since there are no funds to purchase a commercial simulator there are pragmatic reasons to use freely distributed simulators. This also provides a freedom to modify the simulator for the problem at hand, if this would be necessary.

2.7.2 Simulation Packages

Classes of Simulation Packages

Simulators are classified as a continuous and discrete event [SS06]. *Continuous* simulation is preferable when object behaviour in a system can be described by means of equations. *Discrete* simulation assumes passing transaction-flow (programs) by a discrete sequence of steps in the simulation model (network). As program implementation is very difficult and often impossible to predict a discrete event simulator is more effective.

In turn, discrete event simulators are classified into areas of application, e.g. business planning, weather forecasting, molecular modelling, network simulation. Only network simulators are discussed further in this section.

Network Simulators

A wide range of network simulators allows users to model and analyse projects before their real implementations, e.g. JiST/SWANS [Bar04], OMNeT++ [Var10], NS2 [AcbraUBP11], OPNET [opn11], Sim++ [CF11], Traffic [Sof11], and NCTUns [nct11]. The most common areas of using network simulators are as follows:

- *Simulating routing and multicast protocols* is supported by network simulators like NS2 [AcbraUBP11] which is popular in academia due to its extensibility, open source and detailed documentation. NS2 is a discrete event network simulator based on C++ and OTcl [otc11]. It allows simulation of both wired and wireless networks. Dynamically loadable libraries simplify NS2 extensibility and do not require modification of the core simulator.

- *Modelling complex queuing problems* is the main purpose of simulators like Traffic [Sof11] which was specifically developed for tasks that cannot be solved with Erlang equations [Dev87]. A user can define object behaviour without using a scripting language. Results can be exported in a spreadsheet and a word processor format to conduct a deeper data analysis.

- *Analysis of wireless networks* is supported by simulators like JiST/SWANS [Bar04]. SWANS is a Scalable Wireless Ad hoc Network Simulator that was built atop of a simulation engine JiST (Java in Simulation Time). SWANS combines system- and language-based approaches, and provides such models as reception, noise, signal propagation, fading, node mobility. Unmodified network applications can also be run over the simulated network. SWANS supports components for physical, link, network, transport and application layers.

2.7.3 Simulation Package Selected

Analysis of network simulators shows that OMNeT++ [Var10] most closely satisfies the requirements presented in Section 2.7.1. The initial release of OMNeT++ is dated 1996 and developers continue its improvement at present. The current research is conducted using the simulator versions 3 and 4 released in 2005 and 2010 respectively. The simulator is under Academic Public License and is able to work in Dos/OS2/Windows/Unix environment. This allows programs to be easily transferred from one computer to another and to run models on computers with different operating systems.

OMNeT++ has its own library that is an extension of C++. A simulation model is constructed on the basis of modules. These modules have a hierarchical structure where compound modules are composed of simple modules and other compound modules. The module functionality is programmed in C++. Modules from different levels cannot interact directly with each other. A network configuration is specified in a language called NED, and parameters can be assigned either in corresponding NED files or in an configuration file omnetpp.ini. A compiler translates NED into C++. The simplest and the most useful component in the simulator is a message of class cMessage. The library supports message sending, receiving, scheduling and termination. Messages transfer from one module to another through gates that also are called ports in other systems. Thus, a message leaves a module through an output gate and enters through an input gate. The connection between modules of the same level is implemented by means of links. Message delays in the links can be set either ones for a particular link in this case all messages that transfer via this link have the same delay *or* every time a messages leaves a node, here the delay is set via channels. Currently, OMNeT++ does not support parameter arrays in messages but this can be overcome by emulating arrays via strings.

OMNeT++ provides a wide range of graphical event presentation. The graphical part helps us to visualize and understand topology, connections and process implementation. Events can cause output both in textual and graphical forms. Variables defined in the model can also be inspected through a graphical user interface. Another OMNeT++ property is error checking and reporting. An error report contains both cause and location of an error. This helps to locate mistakes quicker and saves time for productive programming. To help in revealing mistakes an experiment repeatability is implemented. OMNeT++ allows to change a seed from 0 to 2^{32}.

The last but not the least important factor is detailed documentation that helps us to understand basic system concepts. Step by step descriptions of simple examples allow users to easily learn the main simulator capabilities.

2.8 Discussion

This chapter has explored the key AMP related background information. The chapter has discussed the reasons of the network type and topology abstraction, and introduced network and AMP simulation features (Section 2.1). It has also covered two core AMP concepts: mobility and autonomy – that allow AMPs to migrate within a network to reduce completion time. AMPs are based on implicit mobile computation, employ code and execution space management, and have previously been investigated using both weak and strong mobility [Den07] (Sections 2.2). From the four aspects of a self-management AMPs employ a self-optimization aspect (Section 2.3). AMPs are unusual in that, in place of some external load management system, each AMP periodically recalculates network and program parameters and may independently move to a better execution environment. Load management emerges from the behaviour of collections of AMPs (Section 2.4).

AMPs are global dynamic load balancers [CK88]. AMPs have decentralised state estimation scheme that involuntary and periodically collects partial information, *and* decentralised, sender initiated, simple decision making policy [Rot94]. AMPs take all movement decisions themselves, independently of other AMPs. An AMP relocates when the predicted time to complete at the current location is greater than the predicted time at the best available remote location plus the communication time to reach the remote location. AMPs have a decentralised state estimation system, and to recalculate parameters they require partial state information that is sent in a periodic involuntary way [Rot94]. To collect state information each location has a load server that allows a reduction of AMP coordination time (Section 2.5).

The chapter has discussed AMP difference from other mobile agent load balancers. Thus, the way AMPs search for a better location may seem similar to iterative algorithms [Cyb89, LRRV04]. Indeed, both approaches aim to distribute load among nodes, and the load is estimated in the number of tasks (or programs) per node. There are two distinctions: firstly, in iterative algorithms locations make the movement decisions, but with

AMPs programs make the decisions themselves. Secondly, while in iterative algorithms the locations aim with each iteration to approach some mean load, AMPs seek only to reduce the program completion time (Section 2.6).

The chapter has examined requirements to the simulator and then explored the alternatives. For the current research the OMNeT++ network simulator has been chosen (Section 2.7). The simulation network construction and the simulation result validation against the real experiment results will be provided in Chapter 3.

Chapter 3

AMP Simulation Design and Validation

To analyse AMPs on large networks an accurate simulation is constructed and validated against experiments with AMPs on real networks. These real experiments were conducted on LANs in [DTM06] using AMPs with weak mobility (e.g. Java Voyager [Rec10]) and strong mobility (e.g. JoCaml [MM10]). The validation uses simulated AMPs with parameters appropriate to weak mobility and compares the simulated and measured results. The chapter only investigates AMP distribution after a system enters a balanced state to validate the simulation model against results presented in [DTM06]. Detailed movements of AMPs are discussed in Chapter 4.

This chapter first outlines the implementation features of AMPs that use weak mobility (Section 3.1) and provides information about experiment set up (Section 3.2). Then it compares the distributions of simulated and real AMPs over locations in homogeneous networks (Section 3.3), and heterogeneous networks (Section 3.4). Finally, the chapter summarises results and briefly defines future research directions (Section 3.5).

3.1 Simulation Design

This section covers features of AMPs that use weak mobility (Section 3.1.1), and justifies the design decisions for the simulated networks that are constructed using the OMNeT++ network simulator (Section 3.1.2).

3.1.1 Weak Mobility

Weak mobility allows only code migration as discussed in Section 2.2. That is after a migration the execution state is not preserved, and the program restarts its execution [FPV98]. AMPs with weak mobility in [Den07] are implemented in such a way that although a program restarts its execution on the new location it places a bookmark on data which allows it to continue the calculation from the stop point. Figure 3.1 shows the coordination behaviour of AMPs with weak mobility [Den07]. The description is as follows:

1. *Preparation to move.* A program initiated at $Loc\ 1$ decides to move to $Loc\ 2$.
2. *Building a reference.* The program sends code to $Loc\ 2$ and creates a reference from the code on $Loc\ 2$ to the data on $Loc\ 1$.
3. *Returning results.* The code on $Loc\ 2$ completes a part of computations and returns results to $Loc\ 1$.
4. *Final state.* The code on $Loc\ 2$ recalculates parameters, and either migrates to another location or continues execution on the current location. Thus, the data at $Loc\ 1$ never moves.

For example, to perform matrix multiplication $A \times B = C$ an AMP contains of program code and two matrices B and C whereas matrix A stays on the root location and its

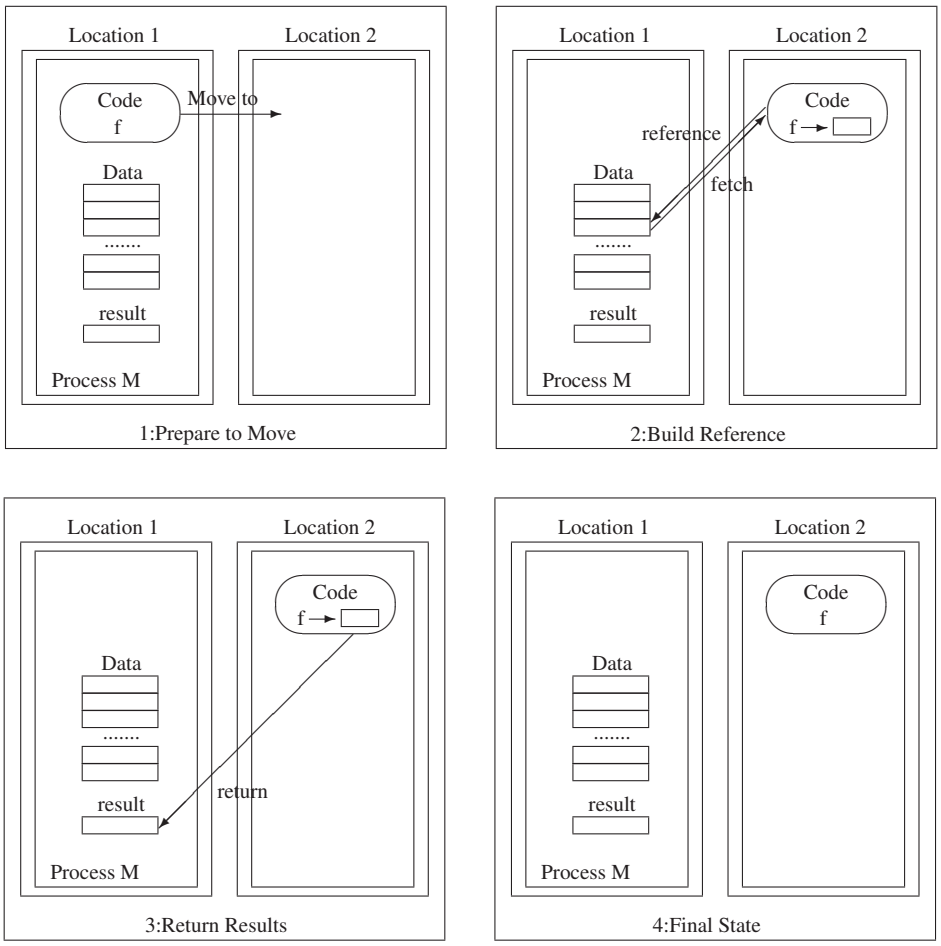


Figure 3.1: Mobile Behaviour of Java Voyager AMPs with Weak Mobility [Den07]

rows are periodically transferred to the location where the AMP is currently executed. In the simulation delays related to periodic data transfer from the root location to migrated AMPs are taken into account in AMP computation speed. Experiments with Java Voyager showed that the root location only had 50% of its capacity available to serve the AMP workload [Den07]. This property is believed to result from the overhead of allowing the AMPs migrated to other locations to access the data that has remained at the root location. Therefore, the root location is assigned a speed of 50% of its clock rate.

3.1.2 Simulation Network Design

The section covers features of simulating computers and the network topology.

Location. A single location (computer) in the simulated network is represented by a `Location` with the three main parts depicted in Figure 3.2: `Generator`, `Queue` and `Switch`. The functionality of each part is discussed below.

`Switch` is the core part of the simulated location, and is connected to `Generator` and `Queue`. `Switch` keeps state information tables and performs corresponding actions when messages arrive. For example, when a state message arrives `Switch` renews state information about the remote location, picks the next location from its list of locations, and sends the state message to that location. `Generator` is only used to generate new messages, e.g. AMPs and state information. `Queue` is used to simulate AMP execution time. Other inner location delays, such as coordination time, are implemented by means of channels. Figure 3.3 shows a program fragment that implements connections between parts of `Location` using the NED language [Var10].

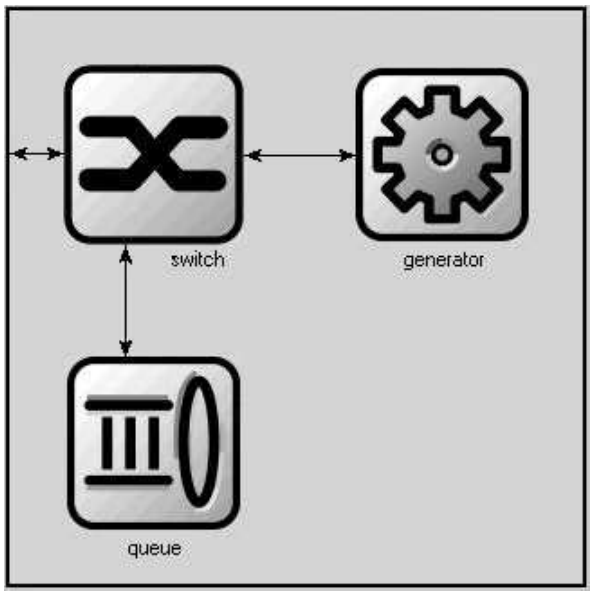


Figure 3.2: Interconnections in a `Location`

```
// Connection to other Locations
for i=0..sizeof(out)-1 {
     switch.out[i] --> out[i] if i!=ownIndex;
     switch.in[i] <-- in[i] if i!=ownIndex;

// Connection between Switch and Queue
switch.out[sizeof(out)] --> queue.in[0];
switch.in[sizeof(out)] <-- { delay = timeCoord; } <-- queue.out[0];

// Connection between Switch and Generator
switch.out[ownIndex] --> generator.in;
switch.in[ownIndex] <-- generator.out;
```

Figure 3.3: Interconnections in a `Location` Using NED

The parameter $sizeof(out)$ contains the size of the `Switch` gate vector, i.e. the number of single gates, through which the connection to `Generator`, `Queue`, and other locations is implemented. In the current simulation it depends on the total number of locations, $numLocations$, i.e.

$$sizeof(out) = numLocations + 1$$

The parameter $timeCoord$ is the coordination time required to collect information from the load server and recalculate parameters. The parameter $ownIndex$ is the `Location` ID number in the network.

Topology. To investigate AMP behaviour a number of fully connected networks are simulated. A snapshot of location interconnections in a simulated network of 15 locations is presented in Figure 3.5. A fragment of code presented in Figure 3.4 imple-

```
for i=0..numLocations-1, for j=0..numLocations-1 {
      location[i].out[j] --> timeComm --> location[j].in[i] if i!=j; }
```

Figure 3.4: Location Interconnection

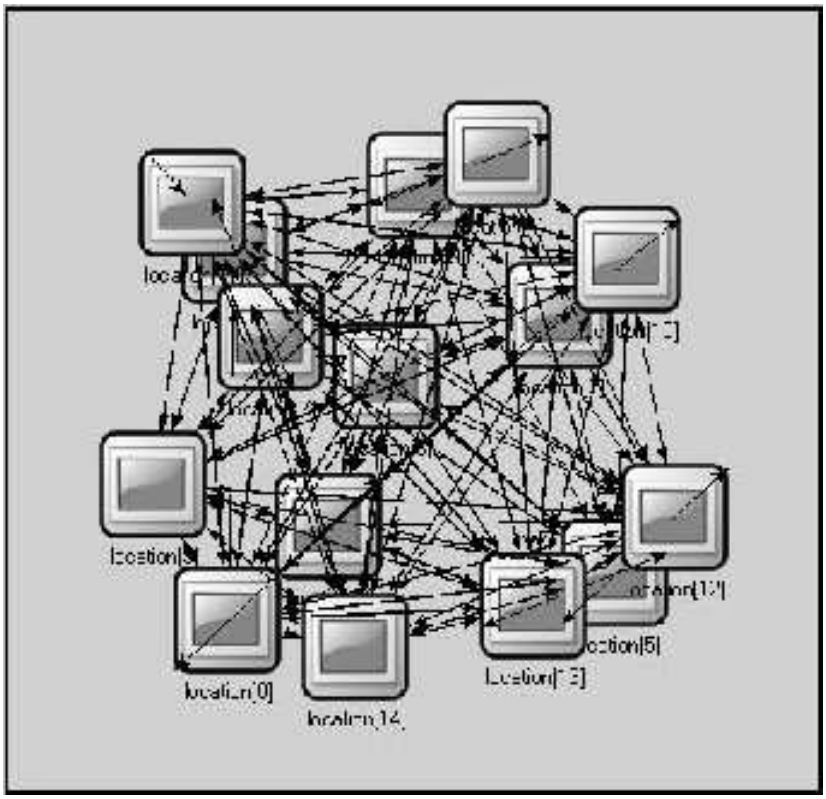


Figure 3.5: Interconnections in a LAN

ments connections between locations of the simulated network in NED language where $numLocations$ is the number of locations in the network, $timeComm$ is the communication time required to move a message (i.e. an AMP or state information) from one location to another.

3.2 AMP LAN Experimental Set Up

AMPs were measured on a range of languages and LANs in [Den07]. Sections 3.3 and 3.4 reproduce the substantial Java Voyager experiments. AMPs are matrix multiplication programs. The measurements are conducted using locations of the following CPU speeds: 3193 MHz, 2168 MHz, and 1793 MHz. To correlate work, W, and location CPU speeds the measurements of matrix multiplication computation time provided in Table 3.1 are used. The analysis of ratio of work to the computation time

from Table 3.1 shows that CPU speed of 3193 MHz corresponds to execution speed of 21385417 units per second. Applying proportion gives the following: 2168 MHz corresponds to 14520383 units per second and 1793 MHz correspond to 12008786 units per second for matrix multiplication programs. Recall that units are used to calculate program work W and depend on the type of program (Section 2.4).

The real experiments were conducted on the Linux operating system using version 3.3 of Java Voyager. The locations were connected into Ethernet networks with 100Mbps communication rate. The time to transfer an AMP that executes matrix multiplication was correlated to matrix dimension, d, i.e.

$$T_{send} = 0.029 + 5.07 \cdot 10^{-6} \cdot d^2. \tag{3.1}$$

Each experiment was run eleven times. The duration of an experiment depends on the following three parameters: simulation mode, the number of AMPs, and the number of locations. The simulation mode indicates the amount of output data during the experiment, i.e. fast mode provides detailed simulation results and express mode only provides main simulation results, such as the total simulation time and the number of generated messages. Thus, *Experiment 1* from Section 3.4 lasts for 180 simulated seconds which in real time takes 15 minutes in fast mode, and 2 minutes in express mode. The express mode was also programmed to output data in a file of such events like an AMP movement from a location, an AMP arrival to a new location, and AMP termination.

Dimension, d	Work, $W \propto d^3$ (unit)	Computation Time, T_{comp} (sec)	CPU Speed (MHz)
300	$27 \cdot 10^3$	1.27	3193
400	$64 \cdot 10^3$	3.00	
500	$125 \cdot 10^3$	6.03	
600	$216 \cdot 10^3$	10.20	
700	$343 \cdot 10^3$	16.00	
800	$512 \cdot 10^3$	23.80	
900	$729 \cdot 10^3$	33.40	
1000	10^6	46.20	

Table 3.1: Computation Time of Matrix Multiplication Programs [Den07, Table 4.11]

3.3 Homogeneous Network

A *homogeneous network* is a set of locations with the same available speed, except for the root location which may have reduced speed, because of the communication with the remote processes that have migrated away from the root location.

Four types of experiments are reproduced for homogeneous networks [Den07, Subsection 5.3.1]: optimal balance (Section 3.3.1), near-optimal balance (Section 3.3.2), adding AMPs (Section 3.3.3) and removing AMPs (Section 3.3.4). This section gives only a brief introduction to *balanced states*, i.e. states where no AMP can gain a greater AMP relative speed by moving [Den07]. The detailed discussion of the balanced state properties and features is provided in Sections 5.1 and 5.2. The number of locations in the experiments ranges from three to five, and between five and thirteen AMPs start at the root location. The CPU speed of all locations is 3193 MHz.

3.3.1 Optimal Balance

The first type of experiment tests the distribution of AMPs in an *optimal balanced state*, i.e. a state where locations with the same available speed have equal number of AMPs. AMPs start execution on the root location and after some time distribute themselves over the network. The number of AMPs and the number of locations was chosen to allow equal numbers of AMPs on all locations except the root. The results of the real [Den07, Table 5.25] and the simulated experiments are presented in Table 3.2. For each number of locations in Table 3.2, the first row is the distribution in the real experiments, and the second row is the distribution in the simulated experiments. The two simulated results that differ from the real experiments are highlighted in bold.

The results show that the simulation model reflects the real balancing of AMPs in the network, except for two cases: 9 AMPs on 3 locations, and 13 AMPs on 4 locations. This mismatch is a result of using 50% of capacity at the root location. When the root

	5 AMPs	7 AMPs	9 AMPs	10 AMPs	13 AMPs
3 Locations					
Real	1/2/2	1/3/3	1/4/4	-	-
Simulated	1/2/2	1/3/3	**2/3/4**	-	-
4 Locations					
Real	-	1/2/2/2	-	1/3/3/3	1/4/4/4
Simulated	-	1/2/2/2	-	1/3/3/3	**2/4/4/3**
5 Locations					
Real	-	-	1/2/2/2/2	-	-
Simulated	-	-	1/2/2/2/2	-	-

Table 3.2: Optimal Balanced Distribution in Real and Simulated Experiments

location has two AMPs and a non-root location has four AMPs the relative speeds of the locations become equal. The AMP does not move because the communication time would be added to its total execution time, i.e.

$$\frac{W_r \cdot 2}{0.5 \cdot S} \not> \frac{W_r \cdot 4}{S} + T_{comm}. \tag{3.2}$$

If the workload of 48% for the root location is used in the simulation model, then simulated results agree with the real experiment results in Table 3.2, i.e.

$$\frac{W_r \cdot 2}{0.48 \cdot S} > \frac{W_r \cdot 4}{S} + T_{comm}. \tag{3.3}$$

3.3.2 Near-Optimal Balance

The second type of experiment investigates *near-optimal balance*, i.e. a state when the total number of AMPs makes it impossible for equal numbers of AMPs to be at each location, but the discrepancy between locations should be at most one AMP [Den07]. Table 3.3 shows distribution of six AMPs between three locations and five AMPs between two locations. For each number of locations in Table 3.3 the first and the second rows represent distribution in the real [Den07, Figures 5.56, 5.57] and the simulated experiments respectively. The results are identical.

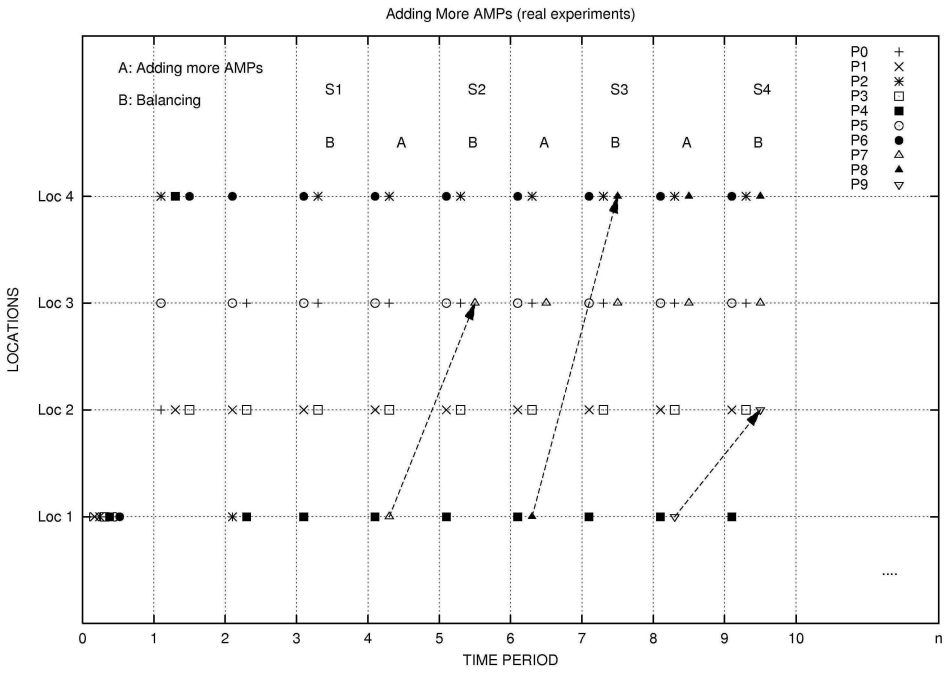
	5 AMPs	6 AMPs
2 Locations		
Real	2/3	-
Simulated	2/3	-
3 Locations		
Real	-	1/2/3
Simulated	-	1/2/3

Table 3.3: Near-Optimal Balance

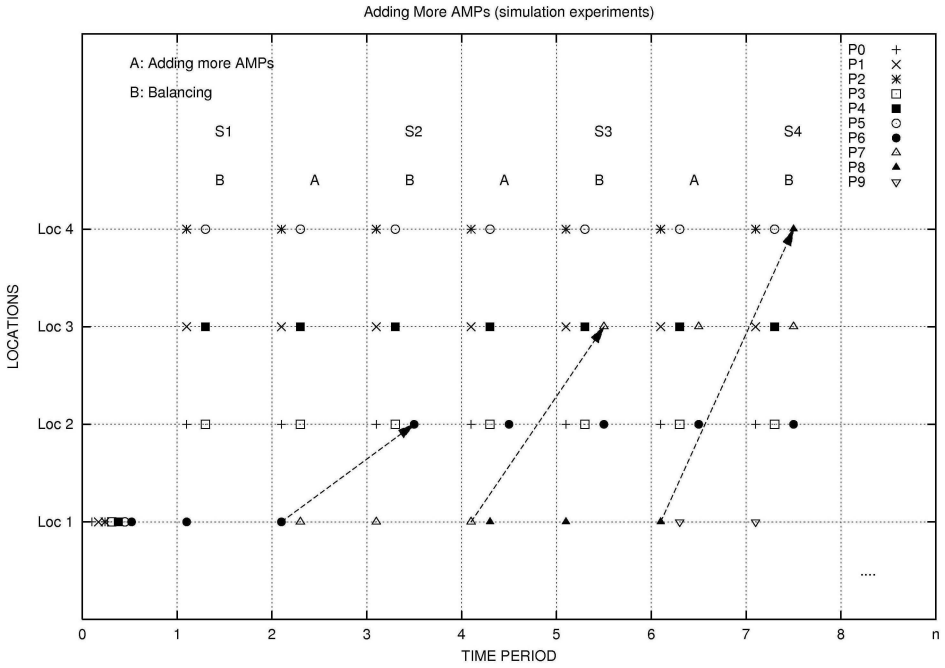
3.3.3 Adding Autonomous Mobile Programs

Two experiments were conducted to analyse AMP distribution after adding more AMPs [Den07, p. 114]. The first experiment has four locations and ten AMPs in total. Initially, seven AMPs start on the root location. When the system enters a balanced state three more AMPs are added one by one. A time period after adding each AMP is chosen to allow the system to enter a balanced state before adding the next AMP. Figures 3.6(a) and 3.6(b) show AMP initial distribution and rebalancing in the real and simulated experiments respectively. States S1, S2, S3 and S4 depict the balanced states the system enters before (i.e. S1) and after (i.e. S2, S3, S4) adding AMPs. Although the real and simulated results in Figures 3.6(a) and 3.6(b) may seem a little different in fact they are identical because locations in the experiments are homogeneous, *and* the number of AMPs in the balanced states S1, S2, S3 and S4 for the real experiment corresponds with the simulated experiments.

The second experiment has three locations and five AMPs that start execution at the root location ($Loc\ 1$), then four more AMPs are added sequentially. Figures 3.7(a) and 3.7(b) depict AMP movements between locations in real and simulated experiments respectively. Identical balanced states are marked S1, S2 and S3. However, after adding the third AMP, the movements of simulated AMPs are not the same as of the real AMPs. The mismatched states are marked S4, S5 and K4, K5. The cause of the mismatch is again the 50% root location workload, as discussed in Section 3.3.1.

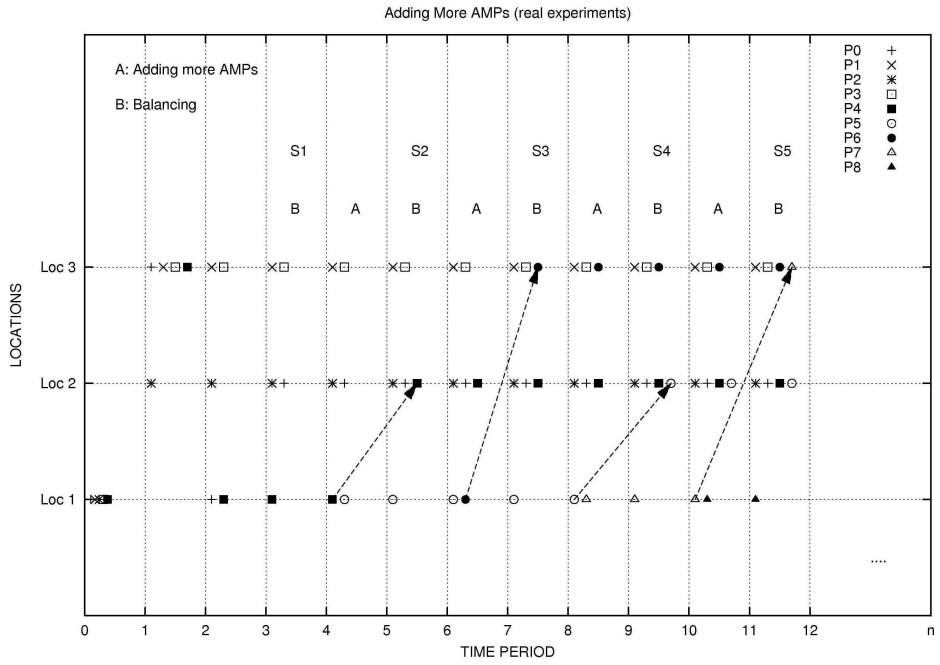


(a) Java Voyager Experiment [Den07]

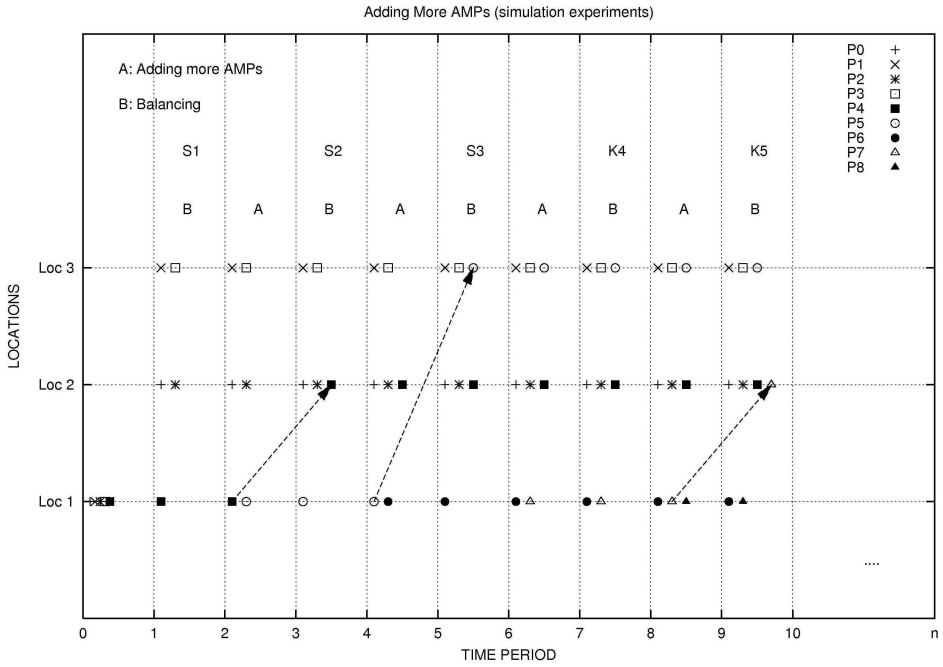


(b) Simulated Experiment

Figure 3.6: 7 AMPs Adding 3 More AMPs on 4 Locations



(a) Java Voyager Experiment [Den07]



(b) Simulated Experiment

Figure 3.7: 5 AMPs Adding 4 More AMPs on 3 Locations

Balance States	Per cent of experiments
S1 (2/4/4)	100%
K1 (2/3/4)	41%
S2 (2/3/3)	23%
K2 (1/3/4)	70%
K3 (1/3/3)	94%
S3 (1/2/3)	100%
S4 (1/2/2)	100%

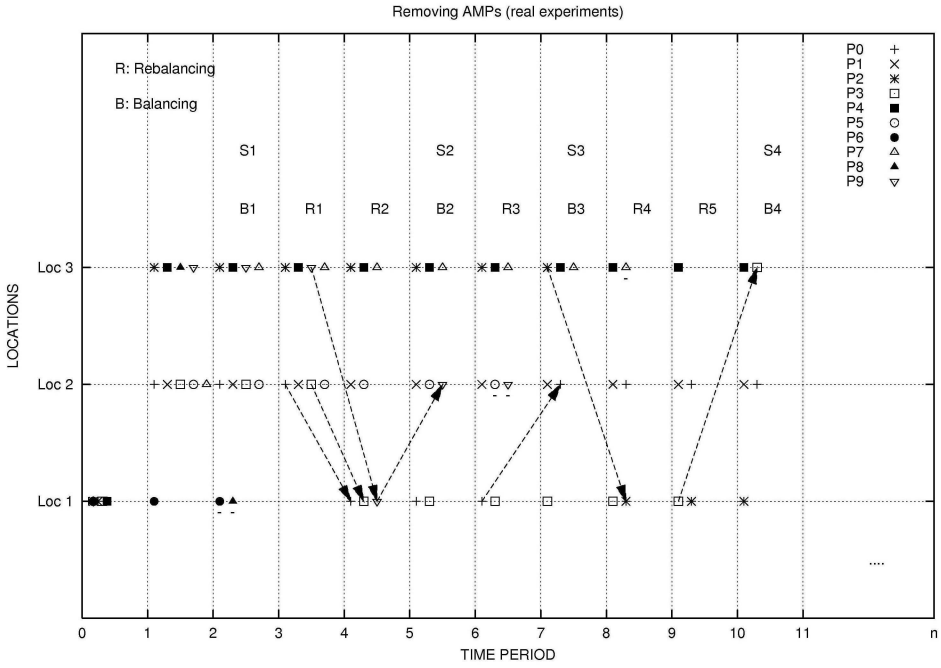
Table 3.4: Balance States of Simulated Experiments

3.3.4 Removing Autonomous Mobile Programs

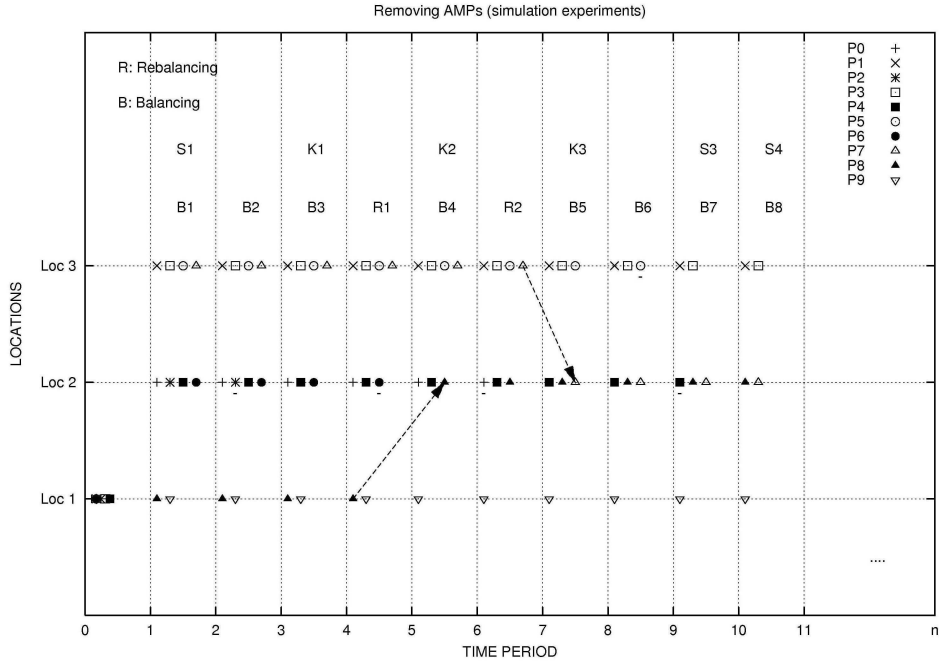
The fourth type of experiment examines AMP behaviour after termination of some AMPs. The experiment analyses distribution of five large and five small AMPs on three locations [Den07, p. 115]. Large AMPs are programs of matrix multiplication of size 1000×1000, and small AMPs are programs of matrix multiplication of size 500×500. All AMPs start at the root location ($Loc\ 1$) and then the AMPs randomly distribute themselves over the network.

Figures 3.8(a) and 3.8(b) show AMP distribution in real and simulated experiments respectively. The sign '-' is used to indicate AMP termination. States S1, S2, S3 and S4 are balanced. AMPs of all simulated experiments enter states S1, S3 and S4. However, only 18% of the experiments enter state S2. This is due to communication time and random distribution of large and small AMPs, i.e. a small AMP may terminate from any location. Depending on which locations AMPs terminate and on which locations other AMPs detect the available processing power the states S2 (2/3/3) or K2 (1/3/4) may result. Both states are balanced. Table 3.4 shows the percentage of experiments the corresponding states have occurred. States K1, K2 and K3 are also balanced states the system may enter.

The real experiments exhibit two additional unbalanced states: R0 and R2 on Fig. 3.8(a) that are not typical for the current simulation model. These states are due to delays in



(a) Java Voyager Experiment [Den07]



(b) Simulated Experiment

Figure 3.8: Removing AMPs

transmitting state information. Thus, if two or more AMPs discover an opportunity to move to the same location to reduce completion time simultaneously, they move and then rebalance again. This is a so called greedy effect that is thoroughly examined in Chapter 4.

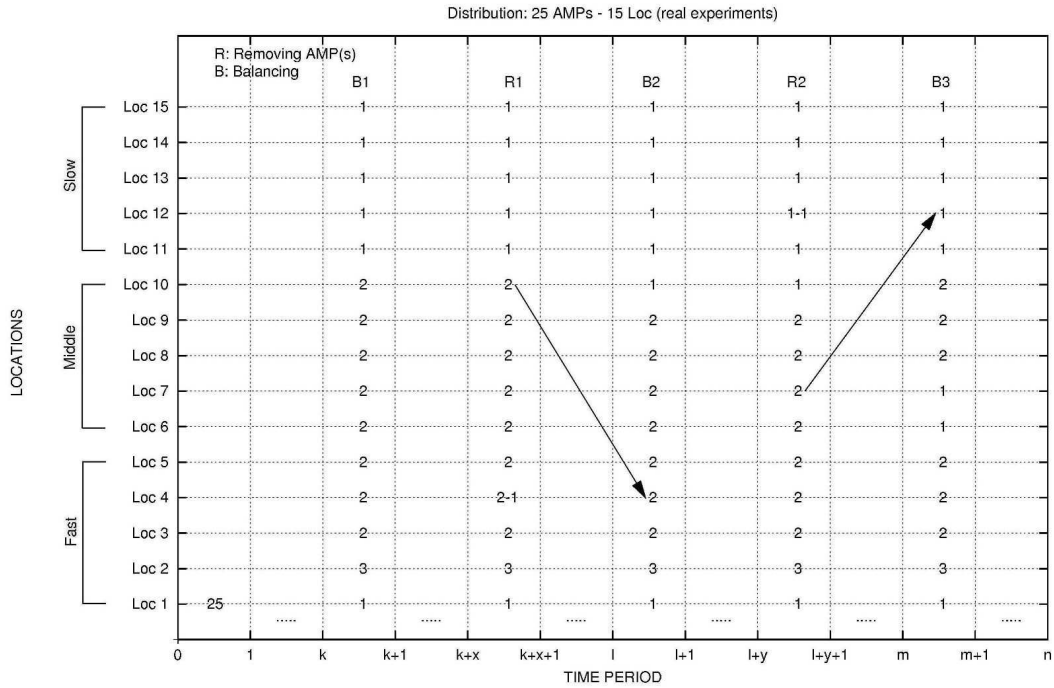
The comparative analysis shows that simulated AMPs enter similar balanced states to real AMPs in homogeneous networks. The difference in balanced states is caused by the use of a constant value for the workload at the root location *and* by the random initial distribution of large and small AMPs.

3.4 Heterogeneous Network

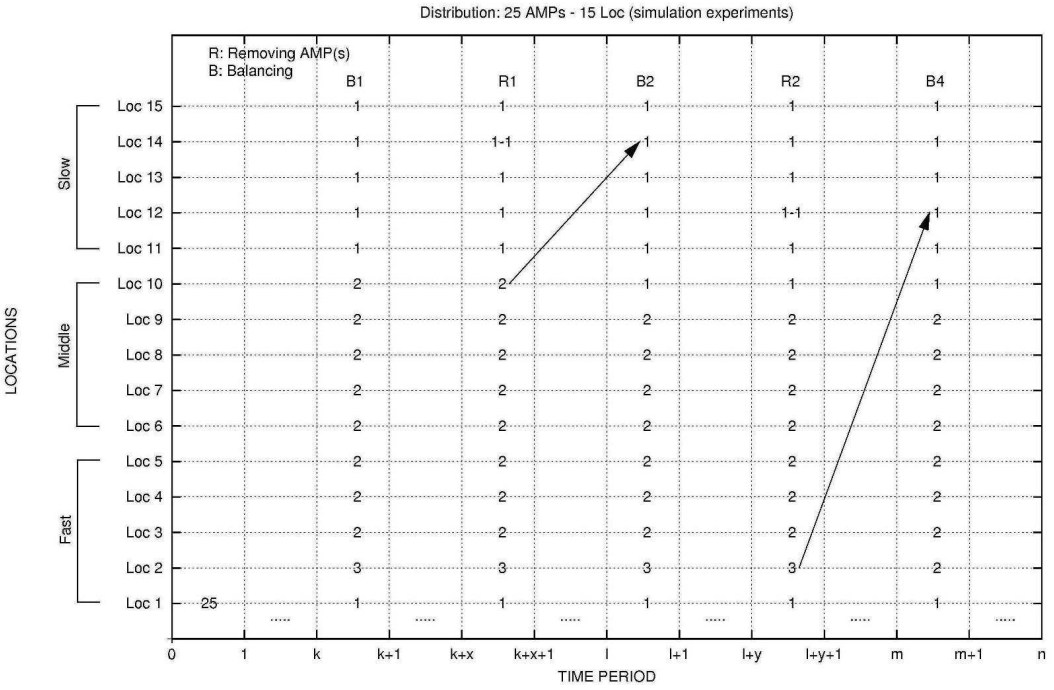
A *heterogeneous network* is a set of locations with different available speeds. For heterogeneous networks two experiments are reproduced [Den07, Section 5.3.2]:

- *Experiment 1:* 25 AMPs and 15 locations with CPU speeds 3193 MHz ($Loc1 - Loc5$), 2168 MHz ($Loc6 - Loc10$) and 1793 MHz ($Loc11 - Loc15$).
- *Experiment 2:* 20 AMPs and 10 locations with CPU speeds 3193 MHz ($Loc1 - Loc5$), 2168 MHz ($Loc6$) and 1793 MHz ($Loc7 - Loc10$).

In *Experiment 1* 13 large and 12 small AMPs start on $Loc1$. The locations are classified into slow ($Loc1 - Loc5$), middle ($Loc6 - Loc10$) and fast ($Loc11 - Loc15$) locations according to the CPU speed. Figures 3.9(a) and 3.9(b) show AMP distributions in real and simulated experiments respectively. The reason the real experiment enters state B3 and the simulated experiment enters state B4 is due to the type of locations where an AMP first discovers the opportunity to move. In the real experiment an AMP moves from a middle speed location, and in the simulated experiment an AMP moves from a fast speed location. Here, both states that real and simulated systems enter are *stable*, i.e. states where no AMP can reduce its completion time by moving. Table 3.5 shows



(a) Java Voyager Experiment [Den07]



(b) Simulated Experiment

Figure 3.9: AMP Distribution in a Heterogeneous Network

percentage of experiments that enter different stable states after the first and the second AMP termination. The '1/3...' represents the states with 3 AMPs at one location, the '1/2...' represents the states where no location has 3 AMPs.

As the locations where small AMPs should initially move are not specified, the small AMPs may terminate from any location, i.e. fast, middle, or slow speed in states R1 and R2. However, 6% of simulated experiments enter the same states R1 and R2 as the real experiments.

If, after an AMP removal, a single movement of an AMP does not result in a stable state, a second movement may occur. It depends on the type of location from which an AMP discovers a better location first, and can be observed in Figure 3.10. The solid lines show the optimal AMP movements, and the dotted lines show the movements that actually occurred. This is another type of the greedy effect (Section 3.3.4).

As an example of this greedy effect, consider the state change from R1 to B2 shown in Figure 3.10. An AMP terminates at $Loc14$ leaving it with excess capacity. All middle speed locations, $Loc6$–$Loc10$, have 2 AMPs and the fast locations $Loc2$–$Loc5$ all have 2 AMPs except $Loc2$ which has 3 AMPs. The optimal movement pattern would involve an AMP moving from a middle speed location *or* the fast speed location that has 3 AMPs ($Loc2$) to $Loc14$. However, one of the fast locations with 2 AMPs ($Loc5$) detects the availability of $Loc14$ first, and an AMP moves from that location, leaving only 1 AMP. The second movement occurs when the fast location with only 1 AMP is detected by either a middle speed location or $Loc2$ that has 3 AMPs.

In *Experiment 2* 10 small and 10 large AMPs start on the root location ($Loc1$). The results are similar to the results of *Experiment 2* (Appendix A).

Type of distribution	After 1^{st} rem.	After 2^{nd} rem.
1/3...	88% of exp	41% of exp
1/2...	12% of exp	59% of exp

Table 3.5: Type of Stages after AMP Removing

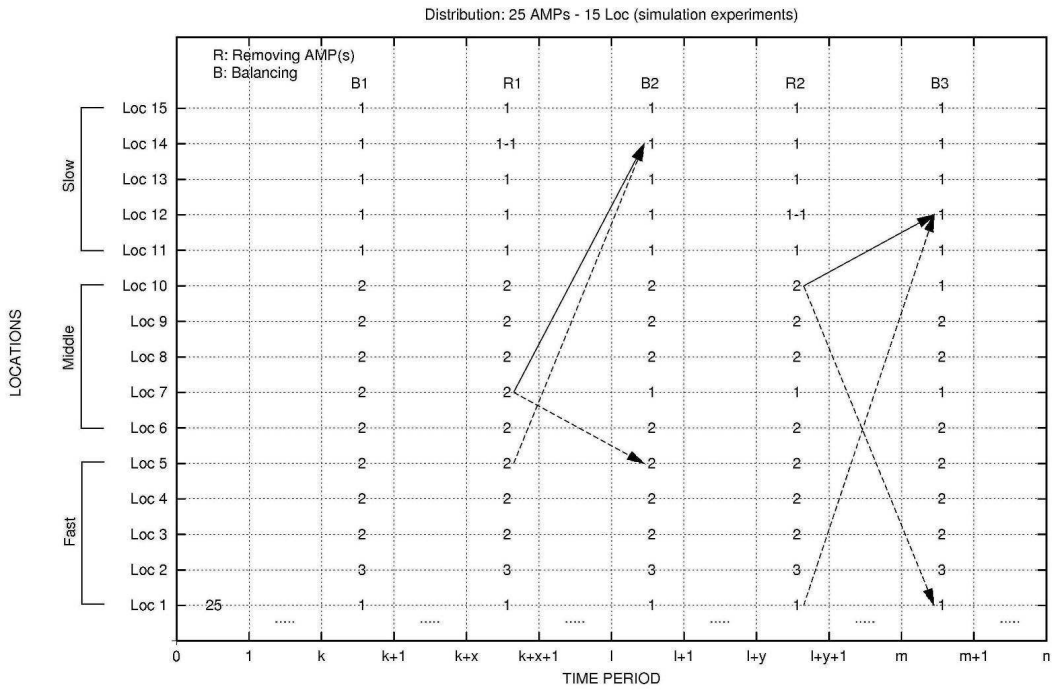


Figure 3.10: AMP Rebalancing: Greedy Effect in Simulated Experiments

The above analysis of AMP behaviour in heterogeneous networks shows that simulated AMPs reflect real AMP distribution. The minor differences are result of random initial allocation of large and small AMPs and random location from which AMPs detect the location with extra capacity first.

3.5 Discussion

To examine AMP behaviour in homogeneous and heterogeneous networks the simulated networks have been constructed and a set of experiments that compare the simulation with Java Voyager AMPs on a LAN have been implemented. The comparative analysis of real and simulated experiments for optimal and near-optimal balancing, adding and removing AMPs in homogeneous networks shows the following:

Optimal balance. All distributions in simulated experiments are matched with the distributions in the real experiments except two cases. The reason of the mismatch is that

the communication workload in the simulated experiments is 50%, whereas in the real experiments the workload varies between 48% and 51%.

Near-optimal balance. Real and simulated experiments enter identical states.

Adding AMPs. Simulated and real experiments enter the same states. The cause of the only difference is the same as in the optimal balanced experiments.

Removing AMPs. All simulated experiments enter three of four balanced states of the real experiments, i.e. S1, S3, S4. In 18% of the simulated experiments AMPs enter all states of the real experiments. 23% of simulated experiments have state S2, and 70% have state K2 that is also a balanced state.

In *Experiment 1* and *Experiment 2* on heterogeneous networks the results are as follows:

- In the 41% and 58% of simulated experiments states B1, B2 and B3 coincide with the same states of the real experiment in *Experiments 1* and *2* respectively. Here, other states that simulated experiments enter are also stable states.
- In 6% of simulated experiments in *Experiment 1* AMPs terminate from the same locations as in the real experiment.
- The greedy effects that are observed in the real experiments, are also observed in the simulated experiments.

Other than a small number of explainable deviations *the current simulation is an excellent model of AMPs on LANs*. This gives confidence to use the model as the basis for further greedy effect investigation. Chapter 4 will analyse and estimate the greedy effects, propose some improvements to reduce redundant movements, and again estimate the greedy effects in the collection of modified AMPs.

Chapter 4

Redundant Movements in Autonomous Mobility

The previous chapter showed that collections of Autonomous Mobile Programs exhibit thrashing, or greedy effects, like other distributed load balancing systems, e.g. [NXG85, GR03, SBK06]. These greedy effects are a phenomenon that results in redundant AMP movements during the balancing of loads between locations, and are a result of locally optimal choices made by each AMP. The greedy effect is estimated by using simulation in the current chapter and then theoretically in Chapter 5. This is the first substantial investigation of thrashing behaviour for collections of distributed agents, and the results of using an agent-based technique, namely negotiation, to ameliorate them.

The investigation starts with identifying two forms of greedy effects in collections of AMPs (Section 4.1). Then adaption of the AMP cost model and simulation to facilitate an investigation of the greedy effects are provided (Section 4.2). The AMP greedy effects are examined on initial distribution, rebalancing, and AMP completion time (Section 4.3). Analysis of the types of movements shows that the majority of redundant movements occur because an AMP is unaware of the intentions and movements of other AMPs. So, the chapter discusses ways to reduce the greedy effects and proposes the

concept of negotiating AMPs (NAMPs) that communicate their intentions with a view to reducing redundant moves. While a number of negotiation schemes are possible, AMPs with a competitive scheme (cNAMPs) have been designed and simulated (Section 4.4). cNAMPs announce their intentions to move and compete with each other for the opportunity to transfer to the new location.

An analysis of simulated cNAMP results shows that even this simple negotiation significantly decreases both the number of redundant movements and the time to rebalance. For example, cNAMPs make no redundant movements during initial distribution, and initial balancing is at least three times faster than for AMPs (Section 4.5). The chapter concludes by summarising results (Section 4.6).

4.1 Greedy Effects

An *optimal rebalancing* is a sequence of AMP movements that is the minimum number of AMP movements needed to enter a stable state.

The AMP *greedy effects* are the result of a non-optimal AMP rebalancing which differs from the optimal rebalancing in having additional redundant movements, and is a result of the AMP making a locally optimal choice, i.e. AMPs do not possess sufficient and accurate state information to make the optimal movement decision. There are two types of the AMP greedy effects: location thrashing and location blindness. Both location thrashing and blindness are observed in real and simulated AMP experiments (Chapter 3).

Greedy effects also occur in other dynamic load balancing systems; other terms are *processor thrashing* [Kuo85], *task thrashing* [GA91], *task dumping* [NXG85, RM90], *transmitting dilemma* [LM82], *El Farol problem* [SBK06].

4.1.1 AMP Distribution Scenarios

To illustrate the greedy effects the following AMP scenarios are introduced. The scenarios specify the number of AMPs and locations, and types of locations:

Scenario 1: 25 AMPs on 15 locations with CPU speeds 3193 MHz ($Loc1 - Loc5$), 2168 MHz ($Loc6 - Loc10$) and 1793 MHz ($Loc11 - Loc15$).

Scenario 2: 20 AMPs on 10 locations with CPU speeds 3193 MHz ($Loc1 - Loc5$), 2168 MHz ($Loc6$) and 1793 MHz ($Loc7 - Loc10$).

Scenario 3: 10 AMPs on 3 locations with CPU speeds 3193 MHz.

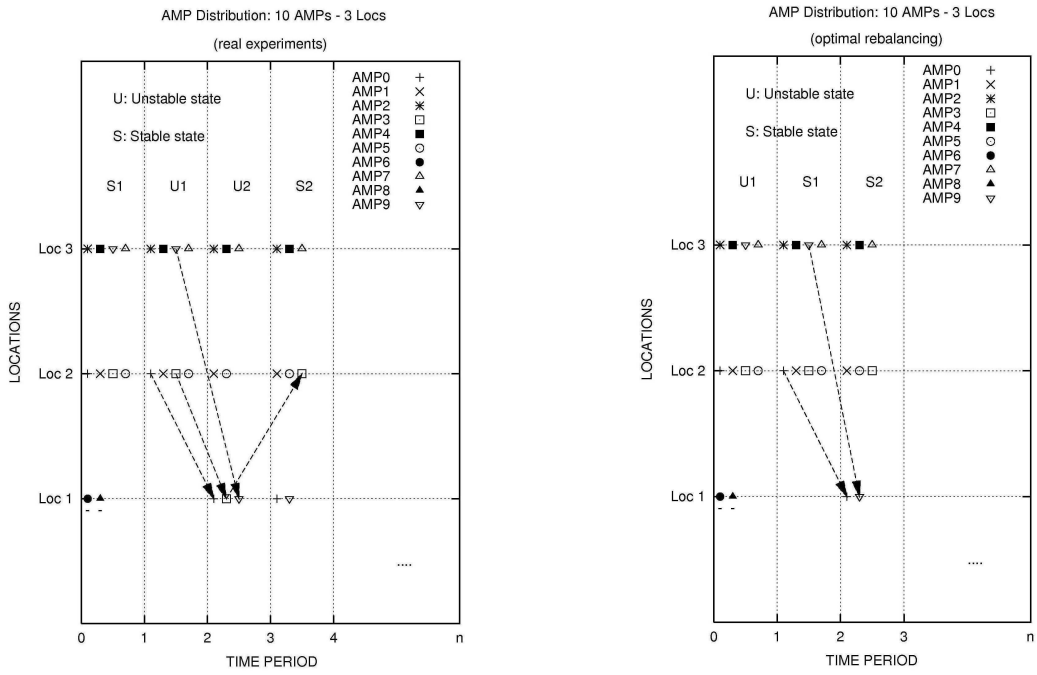
The scenarios are chosen from those discussed in Sections 3.3 and 3.4. For all scenarios $Loc1$ is the root location. Here, large and small AMPs of matrix multiplication are again employed as in Section 3.3.4.

4.1.2 Location Thrashing

Location thrashing is the greedy effect resulting from an AMP's lack of information about other AMPs intending to move to the same location. That is, two or more AMPs decide to move on the basis of the same information about the target location that causes further AMP retransmission.

Location thrashing is illustrated in Figure 4.1(a) which shows AMP movements in the experiments with the real system [DMT10], based on scenario 3 (Section 4.1.1). In Figure 4.1(a) each icon denotes an AMP. The locations are specified on the vertical axis and the horizontal divisions represent time intervals. The time intervals are of different lengths, showing states that the system enters as it attempts to reach a stable state.

After the termination of two AMPs from $Loc1$ in state S1, state U1 is entered and two AMPs from $Loc2$ and one AMP from $Loc3$ discover a better opportunity for execution



(a) Redundant Rebalancing

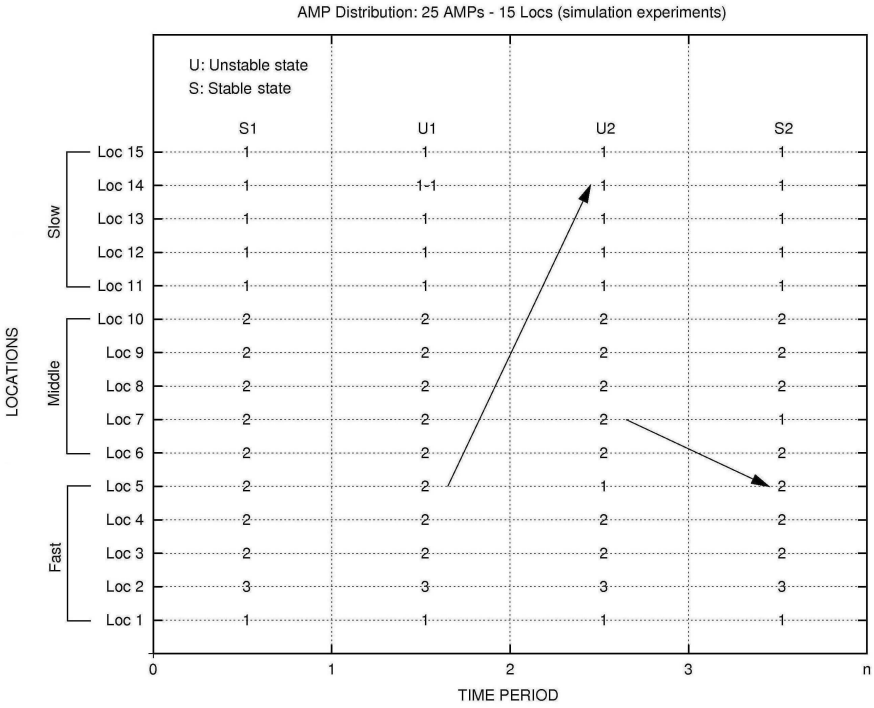
(b) Optimal Rebalancing

Figure 4.1: Location Thrashing Greedy Effect [DTM06]

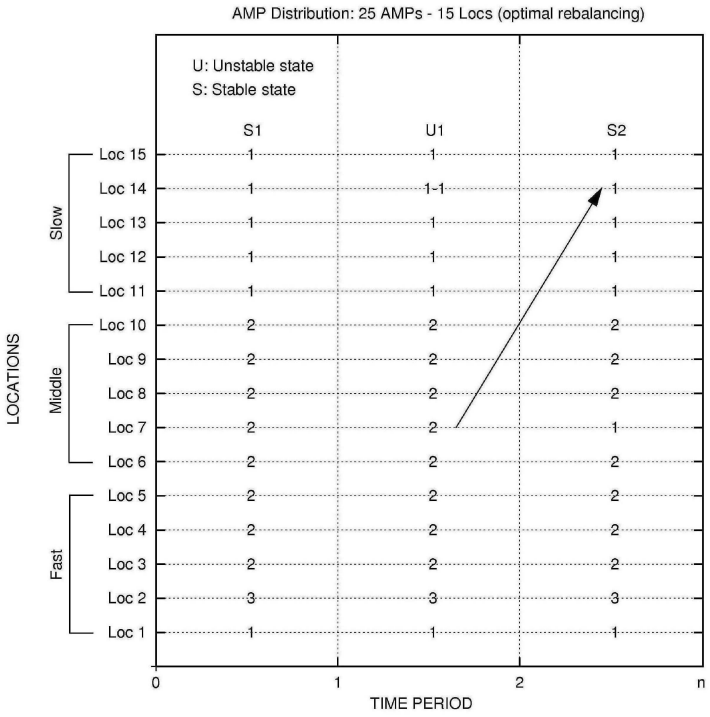
on $Loc1$ simultaneously. These three AMPs move to $Loc1$ (state U2), and then one AMP moves back to $Loc2$ to enter a stable state (state S2). In this case, an optimal rebalancing from the state U1 to the state S2 can be reached by one AMP movement to $Loc1$ from each of $Loc2$ and $Loc3$ as Figure 4.1(b) shows. Note that location thrashing incurs two performance penalties, namely the cost of additional communication and the cost of slower execution. By *additional communication cost* only additional AMP movements during a rebalancing are meant and not communication time they may take.

4.1.3 Location Blindness

Location blindness is the greedy effect resulting from an AMP's lack of information about the remaining execution time of other AMPs. The problem is not with poor runtime predictions, but rather an inability to obtain accurate AMP runtime predictions at distributed locations, i.e. the more accurate information that is required, the more *ex-*



(a) Redundant Rebalancing



(b) Optimal Rebalancing

Figure 4.2: Location Blindness Greedy Effect (Section 3.4)

pensive it becomes to collect and process the information in a distributed system [CK87].

Figure 4.2(a) shows an example of location blindness in a simulated AMP experiment using scenario 1 (Section 4.1.1). Numbers identify the number of AMPs on a location. After an AMP terminates on $Loc14$ in state S1 and the system enters state U1, an AMP from $Loc5$ discovers the opportunity for faster execution first and moves (state U2). Then an AMP from $Loc7$ discovers the opportunity for faster execution on $Loc5$ and also moves (state S2). Figure 4.2(b) shows optimal AMP rebalancing. In contrast to the location thrashing, location blindness only causes redundant communication and causes no additional computation cost. Each AMP will have improved its environment by moving.

Of the two types of greedy effect location thrashing is more harmful, because it causes an increase in AMP completion time, and hence decreases AMP efficiency. To estimate the number of redundant movements and time to balance the greedy effects are simulated and analysed further in this chapter.

4.1.4 Location Thrashing vs. Location Blindness

To better understand the difference between location thrashing and location blindness an analogy between a network and a building is drawn. Here, a room represents a location, a person represents an AMP, a blackboard in a room represents a load server, and corridors represent communication lines between locations.

Blackboards keep information about other rooms and people in their own room independently from each other. A blackboard keeps the most recent information about available speed and the number of people in other rooms, and the number of people in its own room. Each blackboard has a messenger. The duties of the messenger are as follows: to go to another room, take information from the blackboard about the room available speed and the total number of people in that room, return to the home room, rewrite

information about the visited room on the blackboard, and go to the next room. The process of collecting state information by a messenger is an infinite loop.

A person in a room executes a granule of work, checks information on the blackboard, recalculates their parameters, and on the basis of recalculations decides whether to move to another room or to stay in the current one. If the person decides to go to another room, they reduce the number of people written on the blackboard by one, and move to the target room not telling anyone about the destination.

Location thrashing occurs when the same vacant place is spotted by a number of people at the same time, and all of them move to the target room without informing anyone about the destination. Thus, when all people arrive at the same room, some of them should move again, i.e. rebalance.

Location blindness occurs in the following situation. Assume that a building has four rooms: A, B, C, and D where rooms are listed in order of comfort, i.e. room A is the most comfortable and room D is the least comfortable. A person from room A resigns (leaves the building) leaving a vacant place in room A. Then before people from other rooms spot the place person $P1$ from room B discovers the vacant place in room A and moves there. Then person $P2$ from room C discovers a vacant place in room B and also moves (here, a place in room A is already taken). After that person $P3$ discovers a vacant place in room C and also moves. Thus, although all people who moved improve their execution environment, the rebalancing is globally non-optimal because the same number of people in the rooms would be if person $P3$ move from room D to room A. In this case there would be much less movements in the corridors.

4.2 Adapting Simulation to Investigate the Greedy Effects

The experiments in Chapter 3 show that the simulation closely models real AMP distributions on LANs, and hence can be used as an effective tool to analyse AMP behaviour.

This section provides minor modifications to the previous model. The modifications are essential to investigate the greedy effects. The changes include bounds on AMP transfer time (Section 4.2.1) and delays on the state information transfer (Section 4.2.2).

4.2.1 Transferring AMPs

A load server decreases its number of AMPs as soon as an AMP makes a decision to move to a new location. In turn, the new location increases its number of AMPs after the full AMP has arrived [Den07].

In the simulation model AMP transfer time is exponentially distributed with the mean given by T_{send}. However, due to limited data transfer capabilities AMP transfer time cannot be less than a certain value. Therefore, the lower bound for an AMP transfer of $0.6T_{send}$ is introduced. Thus, if the exponential distribution in the simulation provides a value below the lower bound, it returns the value $0.6T_{send}$. The lower bound is taken in accordance with the measurements presented in [Den07].

4.2.2 State Information

The collection of state information from the other locations in the network is done by a load server that sends requests to locations in a round robin sequence [Den07]. The time taken to send a request to a remote Java process and receive a response, T_{req}, has been measured using Java Voyager, and is equal to 0.25s [Den07, p. 80]. Thus, in a network of N locations a load server completely renews state information about other $N-1$ locations every T_{renew} seconds:

$$T_{renew} = T_{req}(N-1). \tag{4.1}$$

As T_{req} is the time to send a request and receive the response, then the time to send a request one way, T_{res}, is

$$T_{res} = \frac{T_{req}}{2}, \tag{4.2}$$

i.e. $T_{res} = 0.125$ second. In the simulation experiments exponential distribution is used with the mean given by T_{res}. The values are restricted by lower and upper bounds. The lower bound is $0.7T_{res}$, and the upper bound is $1.3T_{res}$. So, if the exponential distribution provides a value below the lower bound, it returns the value $0.7T_{res}$, and if the value is above the upper bound, it returns the value $1.3T_{res}$. As before the lower and upper bounds are taken in accordance with measurements presented in [Den07].

The state information collection algorithm is as follows. A load server sends a request to a location from the list. The request arrives to the target location in T_{res} seconds. Then the target location reports its number of AMPs and the response goes back to the initial location, again taking T_{res} seconds. The load server of the initial location renews the state information and sends a request to the next location from the list.

A Unified Modelling Language (UML) diagram involving locations, load servers, AMPs, and auxiliary messages is presented in Figure 4.3.

4.3 AMP Greedy Effect Experiments

This section investigates the frequency and significance of greedy effects in the three scenarios from Section 4.1.1 on homogeneous and heterogeneous networks (Section 4.3.1). Then the redundant movements are classified and the primary cause is identified (Section 4.3.2). Each experiment is repeated eleven times. As the experiments with the real system did not investigate the greedy effects, systematic comparisons with the real experiments cannot be made. However, the results are consistent where they have been compared.

4.3.1 Experiments and Results

Experiment 1 (E1): Initial distribution. This experiment investigates the greedy effects as large AMPs distribute over the network from a single location. Column *Initial dis-*

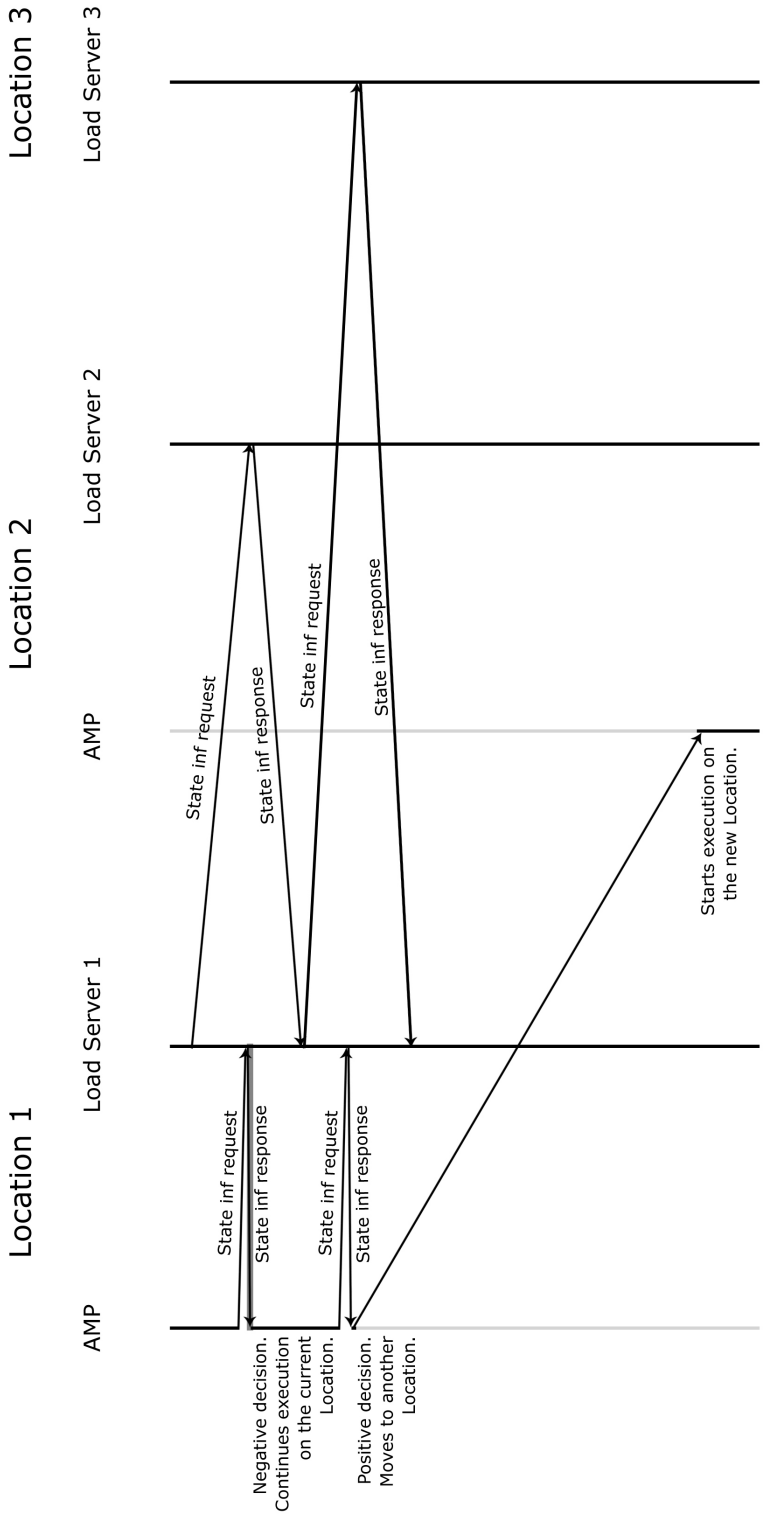


Figure 4.3: AMP UML Diagram

tribution in Table 4.1 shows the mean number of redundant movements and the mean time the system requires to enter a stable state. Clearly the optimal number of AMP movements to reach a stable state would occur if each AMP moved a maximum of once, directly to the location it would occupy in the stable state. The column shows that as the number of locations and AMPs increases the mean number of movements per AMP also increases.

Experiment 2 (E2): Rebalancing after an AMP termination. The experiment measures the number of movements and time required for a system to rebalance after an AMP termination. In this experiment only large AMPs are used, and not large and small as in the real and previous simulation experiments (Chapter 3). This is done because an insufficient difference in amount of work between large and small AMPs creates restrictions on large AMP movements after termination of small AMPs, and does not allow to estimate the degree of redundant movements. By the time small AMPs terminate the remaining execution time of large AMPs becomes compatible with the communication time. This results in large AMPs entering stable states that are not balanced, but in which no movements occur because of the communication cost. Hence AMPs make fewer movements to rebalance.

Initially the AMPs are distributed among locations of a network so that the system is in

Config.	E1: Initial distribution		E2: Rebalancing after an AMP termination		E3: Large AMP completion time, sec	
	Mean No. of redun. moves	Mean time, sec	Mean No. of redun. moves	Mean time, sec	Mean	Standard deviation
Scenario 1 25 AMPs, 15 Locs	64	60.4	6	22.5	173.8	7.66
Scenario 2 20 AMPs, 10 Locs	43	50.5	11	28.2	182.1	11.5
Scenario 3 10 AMPs, 3 Locs	13	26.8	6	14.1	232.6	9.91

Table 4.1: AMP Greedy Effect Experiment Summary

a stable state. After that an AMP from a location is removed (however, each location has information about the balanced state), and start the experiment. Time to rebalance is the interval from the time of AMP termination till the time at which the system enters a balanced state again. Column *Rebalancing after an AMP termination* in Table 4.1 shows the mean number of redundant movements and rebalancing time.

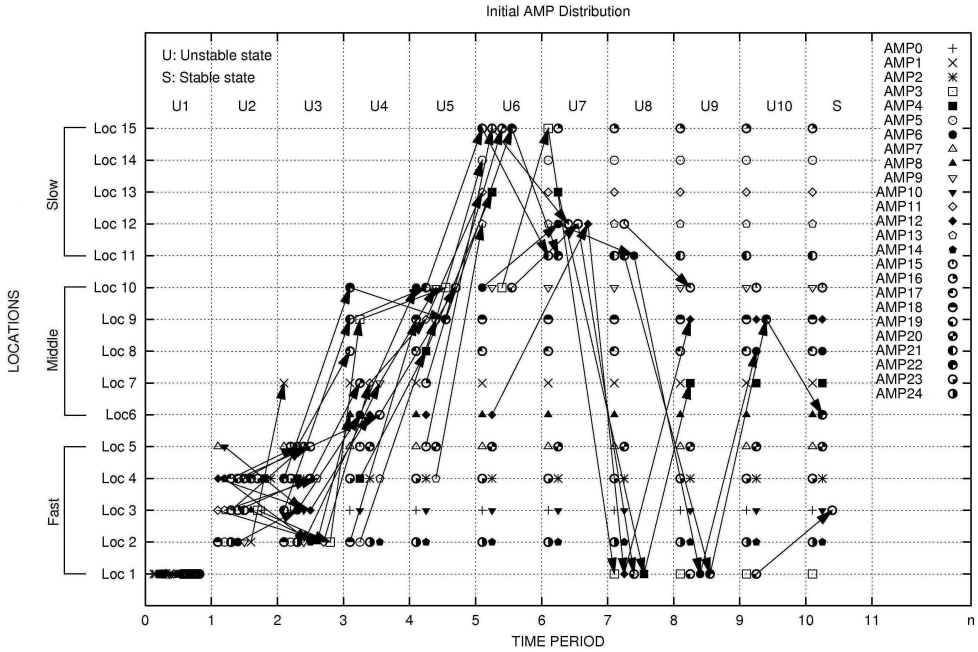
Experiment 3 (E3): Large AMP completion time. This experiment estimates large AMP completion time and measures its variability. The total number of AMPs corresponds to the relevant scenario. All AMPs, two of which are small and the rest are large, start on $Loc\ 1$. The results are presented in column *Large AMP completion time* in Table 4.1.

The simulation experiment results in Table 4.1 show that an increase of the number of AMPs and/or locations causes an increase in the number of redundant movements per AMP.

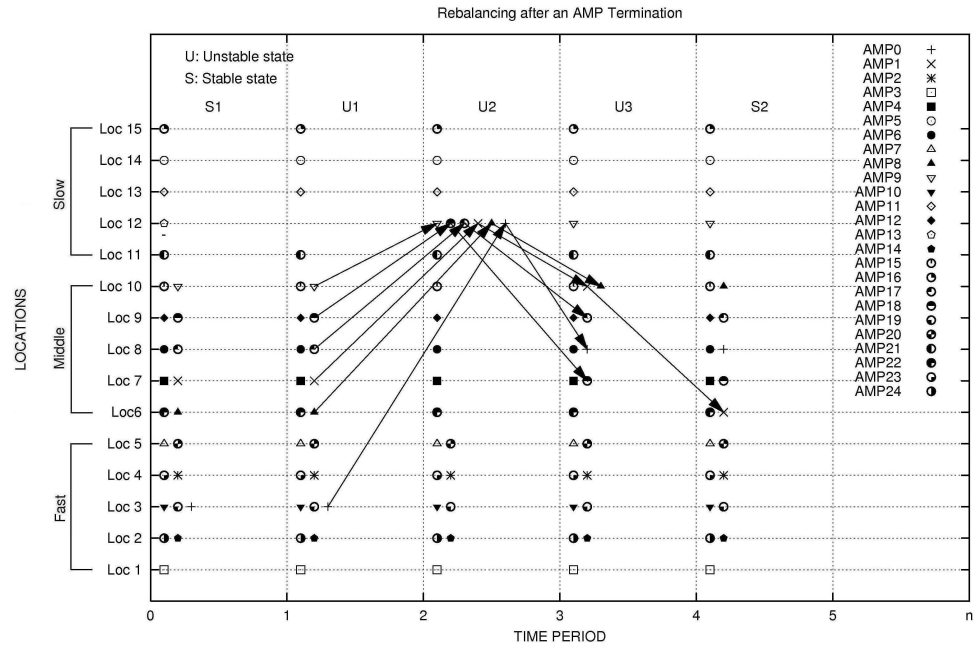
4.3.2 Analysing Greedy Effects

To identify opportunities to reduce AMP redundant movements AMP behaviour is analysed during initial distribution and after an AMP termination. Figure 4.4(a) shows the initial AMP distribution between locations as the system rebalances from initial (unstable) state U1 to balanced state S. As all AMPs move from $Loc1$ in state U1 to $Loc2 - Loc5$ in state U2, the movements are not indicated with lines. There are 88 movements in total, but there would only be 24 if each AMP moved directly to the location it reaches in state S, i.e. the system makes 64 redundant movements. Figure 4.4(b) shows 12 AMP movements which the system makes to enter a new stable state S2 after an AMP termination on $Loc12$ in state S1.

In Figure 4.4(a) consider AMP22 as it moves during the transitions from state U9 to state S. The AMP moves from $Loc1$ in state U9 to $Loc9$ in state U10 which has already two AMPs, and then it moves to $Loc6$ in state S. However, $Loc6$ has only one AMP in state U9 and the same available speed as $Loc9$; AMP22 would complete execution sooner if



(a) Initial Distribution



(b) Rebalancing after AMP Termination

Figure 4.4: AMP Movements (Scenario 1)

it selected $Loc6$ as target. This happens because in the simulation an AMP is associated with a location as soon as it makes a decision to move and not after its actual arrival, i.e. although AMP12 decided to move to $Loc9$ at time 7 $Loc1$ has no information about decisions of AMP12 at time 8. Thus, at time 8 $Loc1$ only has information that there are three locations, i.e. $Loc6$, $Loc8$ and $Loc9$ which have the same available speed and one AMP each. So, AMP22 chooses a location to move (i.e. $Loc9$) at random.

The analysis of AMP movements allows them to be classified into the following main types of redundant movements:

- *Two or more AMPs move from one location to another, and then move again to rebalance.* This type of movement can be observed in Figure 4.4(a) in states U1–U2. At time 0 the load server of $Loc1$ has information that all locations are vacant. As a load server announces that it has an additional AMP only after a full AMP arrival, the load server of $Loc1$ during the whole time of AMP distribution between $Loc2 - Loc5$ (fastest locations) provides information to AMPs that all fastest locations are vacant, and AMPs choose among them at random.

- *Two or more AMPs move from a location, and then some AMPs move back to the location.* In Figure 4.4(a) in state U6 three AMPs move from $Loc10$ to $Loc12$ and $Loc15$. Then in state U9 an AMP from $Loc12$ moves back to $Loc9$. Here, the AMPs are not specified, i.e. which AMPs move from $Loc10$ to $Loc12$ and which AMP moves from $Loc12$ to $Loc10$, because the concern is about the efficiency of the movements.

- *Two or more AMPs move from different locations to one location, and then move again to rebalance.* In Figure 4.4(b) after an AMP termination from $Loc12$ six AMPs from different locations discover a better opportunity for execution simultaneously. They all move to $Loc12$, and then rebalance.

Therefore, the conclusion is that *redundant AMP movements are mainly caused by AMP ignorance of the intentions and actions of other AMPs in the network and hence a lack*

of information to make an efficient decision, i.e. location thrashing.

4.4 Negotiating AMPs and cNAMPs

The section proposes to use an agent-based approach, namely negotiation, to reduce the greedy effects exhibited by AMPs. The main reason for poor AMP movement decisions is a lack of communication between AMPs via the load servers. Possible methods of negotiation between AMPs and/or load servers are discussed in Section 4.4.1. The section introduces a simple negotiation scheme in which AMPs announce their movement intentions and compete for opportunity to transfer. AMPs using this scheme are called cNAMPs. The modifications to AMP algorithm to provide this behaviour are discussed in Section 4.4.2.

As it is discussed in Section 4.1 the greedy effect is not unique for AMPs and is observed in other distributed systems. An important issue here is that the effect appears not only in systems with a scheduler but also in such highly decentralised systems like AMPs. And thus greedy effects are inevitable due to the non-zero nature of communication delays. However, a number of techniques were developed to eliminate redundant movements some of which are presented below.

Schlegel et al. [SBK06] minimize redundant movements by using historical communication information to reduce redundant movements termed the El Farol problem. The original El Farol problem setting is as follows: a hundred people repetitively and independently decide whether to go to the El Farol bar on the specific day of the week or not. The participants do not communicate with each other, and would decide to go to the bar if they expect there less than 60 people. The choice is only based on the number of customers during the previous weeks and is independent of the person previous visits. The approach to reduce the number of redundant movements differs from the one presented in Section 4.4.2 in that the agents make network load prediction to decide where to execute.

Anthony [Ant05] reduces the system state-flapping which is the system constant change between two adaptation states [HMH07] by introducing a delayed-bid mechanism. The principle of the delayed-bid mechanism is in introducing a random delay in a response to a request. Thus, the mechanism spreads the responses in time reducing the number of simultaneous responses in large systems and introduces determinism as it is unknown when a particular node is going to send a response. The current research does not aim to stop flapping because due to the AMP cost model AMPs do not suffer from constant flap between a few states but rather gradually approach the target state. Thus, the target of the current research is to minimise the number of steps and hence time that the system makes to enter a stable state.

Hosseini et al. [HLMV87] reduce thrashing employing damping mechanism. The mechanism introduces a damping factor, D. The nodes that intend to exchange load compare their loads first. The exchange occurs only if the difference between the loads exceeds the damping factor.

Rao et al. [RLS$^+$03] reduces thrashing by implementing an ID based load balancing algorithm that targets structured peer-to-peer (P2P) networks with distributed hash table (DHT) abstraction [RFH$^+$01]. The load balancing is achieved by transferring virtual servers from heavy loaded nodes to light loaded nodes. The algorithm is implemented in three schemes: one-to-one, one-to-many, and many-to-many – where the schemes differ from each other in the number of heavy and light nodes that take part in load balancing procedure. Thus, in the one-to-one scheme a light node randomly generates an ID number, performs a lookup for that ID, and if the node is heavy then the heavy node may transfer some load to the light node. The one-to-many scheme involves a heavy node and a number of light nodes where the virtual servers from the heavy node may be transferred. The many-to-many scheme is implemented in two options: centralised and decentralised. The centralised scheme is implemented via a global pool where all extra load is first transferred to making all nodes light, and then from the global pool the load is redistributed between nodes. The decentralised scheme is implemented similar to the centralised scheme with only difference that the role of pool is implemented by one of

the nodes for a small group of heavy and light nodes. The algorithm has a number of differences from AMPs: first, it targets only P2P DHT abstracted networks; second, each node has a scheduler that decides when and which virtual server should be transferred to another node, whereas in an AMP implementation each AMP decides itself when and where to move.

4.4.1 Negotiating AMPs

The analysis of the greedy effects in the simulated AMP experiments in Section 4.3 shows that the majority of redundant movements occur because an AMP makes a decision on the basis of currently available information and is unaware of impending movements of other AMPs.

In order to reduce the greedy effects AMPs must negotiate with each other, i.e. communicate more information. The most common interpretation of *negotiation* is "interaction among agents based on communication for the purpose of coming to an agreement" [Wei99]. There can be different types of negotiation, such as malicious and honest. A *malicious strategy* would be for a load server to misrepresent the load so that other AMPs were deterred from moving to a location. An *honest strategy* requires AMPs and load servers to share information to reduce wasted movements and is more effective for load balancing. An honest negotiation in an AMP implementation can be designed in a number of ways, some of which are as follows:

- *Competitive.* AMPs compete with each other to move to the target location. Apart from reducing the number of redundant movements this strategy allows us to preserve one of the main AMP advantages, i.e. making an effective movement decision on the basis of small amount of information and simple calculations.
- *Queuing.* Each AMP has a sequence number in a queue. An AMP moves to the new location only if an earlier AMP in the queue decided not to move. The queue

ordering can be global or associated with a location. The method might allow us to minimise AMP activity as AMPs would not take any actions towards reducing their completion time until their turn in the queue has come. The drawback of the method is in the complexity of keeping queue track due to the constant generation and termination of AMPs.

- *Probabilistic.* AMPs and/or load servers collect information about AMPs and locations of a network and also responses to the movement requests. Thus, an AMP makes a decision to move on the basis of calculating the probability of simultaneous AMP movements from other locations. Collected historical information can also be used to identify malicious AMPs and locations. The disadvantage of the method is that it significantly complicates AMP decision making process and enlarges the amount of information that is kept about other AMPs.

- *Relationship.* A network is logically divided into groups, and locations first share information within their group. A location can be a member of more than one group, thus information is spread like a rumour. The approach eliminates the number of AMPs that able to simultaneously react on the opportunity to reduce their completion time. However, the method might be more suitable for large networks rather than small networks like LANs which are considered in the current chapter.

- *Cooperative.* AMPs communicate state information with each other, e.g. remaining work. Comparing remaining work of AMPs that intend to move to the target location AMPs then cooperatively decide which AMP should move to benefit the most from the transfer. Using information about remaining work of AMPs might allow the system to better utilise computer resources, for example, an AMP might move to the target location before an AMP on the target location actually terminates. Thus, the AMP exploits the target location resources immediately after idle resources become available. The drawback of the approach is that the more information it is necessary to collect, the more 'expensive' it becomes to collect and

maintain this information, and hence the more complicated the decision making mechanism becomes.

- *Altruistic.* An AMP recalculates not only its own parameters but also parameters of the fellow AMPs. An AMP moves only if it profits the most from the movement in comparison with other AMPs, otherwise the AMP gives in the movement opportunity for benefit of other AMPs who are in a greater need. The method might allow us to significantly reduce the number of redundant movements. However, the drawback is that the method increases recalculation time and requires AMPs to collect and maintain large amount of information about each other.

- *Musing.* After spotting an opportunity to move to another location an AMP would wait for some random interval and then recalculate its parameters again. If the location is still available then the AMP moves, otherwise it continues execution on the current location allowing another AMP to reduce its completion time by moving to the target location. Like approach in [Ant05] this method also introduces delay between receiving information and reacting on it. The method might be most effective in implementing AMPs in sensor networks.

- *Tit-for-tat.* Load servers would collect information about untruthful AMPs and locations. This information would then be used by AMPs to decide which type of strategy is better to use towards a particular AMP and whether it is worth to move to a particular location. This strategy might be useful in the future research when investigation of AMP behaviour in presence of malicious nodes will be conducted.

4.4.2 The Design of cNAMPs

cNAMPs are negotiating AMPs with an honest competitive scheme which announce their intentions to move and compete with each other for an opportunity to transfer to the new location. In the context of cNAMPs the negotiation is a simple coordination among competitive and self-interested agents [Wei99]. cNAMPs do not negotiate directly with each

other, but only by means of a load server. The reason to choose an honest competitive strategy is due to the fact that it allows us to preserve the main AMP features, such as a truly decentralised algorithm where each AMP decides itself when and where to move, and a 'cheap' decision making mechanism with minimum time and resources required to make an efficient movement decision.

cNAMPs are designed only to reduce location thrashing. Eradication of location thrashing eliminates redundant movements during initial distribution and significantly reduces the number of redundant movements during rebalancing (Table 4.4 in Section 4.5). In addition, reduction of location blindness requires that cNAMPs and load servers possess even more information about locations and cNAMPs of the network. Section 5.2 shows that both the probability of redundant movements and their number are very small.

The section discusses transferring information about cNAMP impending to move, proposes negotiation properties to prevent simultaneous information discovery, and discusses different schemes of information recalculation.

As before state information for cNAMPs is collected by load servers. To compare principles of state information collecting in AMPs and cNAMPs recall the algorithm of collecting state information implemented in AMPs: a load server sends a request to the target location, then the load server on the target location provides its state information, the request returns to the initial location, load server of the initial location renews state information about the target location, then picks another location from the list of location and repeats the procedure (Section 4.2.2). Thus if a network consists of N locations then at any time at most N state messages move in the network, i.e. linear dependency. In cNAMP implementation this linear dependency is preserved but the mechanism of providing information is changed from involuntary type to voluntary type (Section2.5.3). The algorithm is as follows: a load server picks a location from the list of locations and sends its state information to this target location, after receiving the state information, the load server of the target location renews state information about the initial location and deletes the message. In turn the initial node awaits for $min(T_{req})$, picks another

location, and repeats the procedure. Thus, the approach eradicates response messages and at every moment N state messages move in the network. Fraction of location resources allocated to collect information is very small, and even if a LAN consists of 1000 locations only 1000 state messages simultaneously move in the network.

Information about Impending cNAMPs

In contrast to AMPs each cNAMP load server maintains two values for the load:

- the *actual load* that is the number of executing cNAMPs and is used for local cNAMP calculations.
- the *committed load* that represents the actual load of a location together with the cNAMPs that have received confirmation to transfer to the location, and is used by remote load servers.

The rationale is as follows. In the *AMP implementation* the load server increases the number of AMPs only after an AMP has completely arrived and is ready to execute. This means that during the transfer of an AMP other locations will receive information that will shortly be outdated and they may send other AMPs. Locations receive outdated information about the availability of the target location during the time to transfer the first AMP plus the time to renew state information by load servers of all locations, i.e. $T_{send} + T_{renew}$. In the *cNAMP implementation* locations receive outdated information during the time required to transfer information about a new cNAMP arrival plus the time to renew state information at the load servers of all locations, i.e. $T_{res} + T_{renew}$. Thus, the time for which erroneous load information persists is reduced by $T_{send} - T_{res}$ seconds.

Simultaneous Information Discovery

When an AMP decides to move from the initial location to another location it moves immediately without any request or *confirmation* from the target location. Thus, simultaneous information discovery causes the simultaneous AMP movements from different locations in both the simulated and the real experiments.

To solve the problem of simultaneous information discovery cNAMPs send a request. Simultaneously with sending the request the initial location load server locks information about the target location, so that other cNAMPs from the same location do not also send requests to the target location. The information about the target location is locked until a response is received. The cNAMP which sent the request continues execution on the initial location while waiting for the response. The request contains information about the cNAMP's remaining work and remaining execution time on the initial location.

After the request has arrived at the target location, the request recalculates the cNAMP remaining execution time as if the cNAMP was transferred. Then it adds the cNAMP transfer time, and compares the result with the cNAMP remaining execution time on the initial location. If the execution time on the initial location is larger than the remaining execution time on the target location and the communication delay, then the cNAMP informs the target location that the new cNAMP will be transferred, and the target location increases the number of cNAMPs reported by its load server.

After that the response returns to the initial location. If the cNAMP receives:

- a *negative response*, the cNAMP continues execution on the initial location.
- a *positive response*, the load server on the initial location decreases the number of cNAMPs, and the cNAMP moves to the target location.

In both cases, after receiving a response, the local load server renews its information about the target location to prevent other cNAMPs from the current location sending requests to the target location on the basis of out-of-date information.

Configuration	T_{req} (sec)	T_{res} (sec)	T_{renew} (sec)
15 locations	0.25	0.125	3.5
10 locations			2.25
3 locations			0.75

Table 4.2: Parameters of the cNAMP Simulation

To calculate the condition for a cNAMP movement (2.5) communication time, T_{comm}, needs to take into account the time a cNAMP awaits a response from the target location. So, T_{comm} for cNAMPs includes not only the time to transfer a cNAMP, T_{send}, but also the time to receive confirmation to move, T_{req}:

$$T_{comm} = T_{send} + T_{req}. \tag{4.3}$$

The design does not take into account such possibilities as cNAMP termination on the initial location or starting a new cNAMP on the target location which may happen while waiting for a response, $T_{req} \approx 0.25$ second. As time required to recalculate parameters, T_{coord}, is estimated to be equal to 0.011 sec (Section 2.4), and is much smaller than T_{req}; therefore, T_{coord} is included in T_{req} in the simulation. cNAMPs do not recalculate parameters depending on a very low probability of those events during T_{req}. Table 4.2 gives the values of T_{req}, T_{renew}, and T_{res} for the networks of 15, 10 and 3 locations which are used in scenarios 1, 2 and 3 (Section 4.1.1). The values are calculated on the basis of Java Voyager matrix multiplication data discussed in Section 2.4, and equations (4.1) and (4.2).

Schemes of cNAMP Recalculation

After a cNAMP sends a request to the target location the load server on the initial location does not provide information about the target location to the remaining cNAMPs on the initial location. This is done to prevent a simultaneous requests being sent to the target location by cNAMPs from the same location. Hence, while a cNAMP awaits a response

the remaining cNAMPs on the initial location have information only about a subset of the network locations. A cNAMP parameter recalculation can be implemented on the basis of different schemes. Here, only three schemes are analysed, i.e. a cNAMP recalculates parameters only when information about:

- all locations is available.
- at least half locations of the network is available.
- at least one another location is available.

To choose the most effective scheme simulation experiments using the network configuration presented in scenario 1 are conducted, and the distribution of 25, 100 and 500 large cNAMPs is analysed. Figures 4.5, 4.6 and 4.7 show the numbers of terminated cNAMPs in the corresponding intervals in the experiments with 25, 100 and 500 cNAMPs respectively.

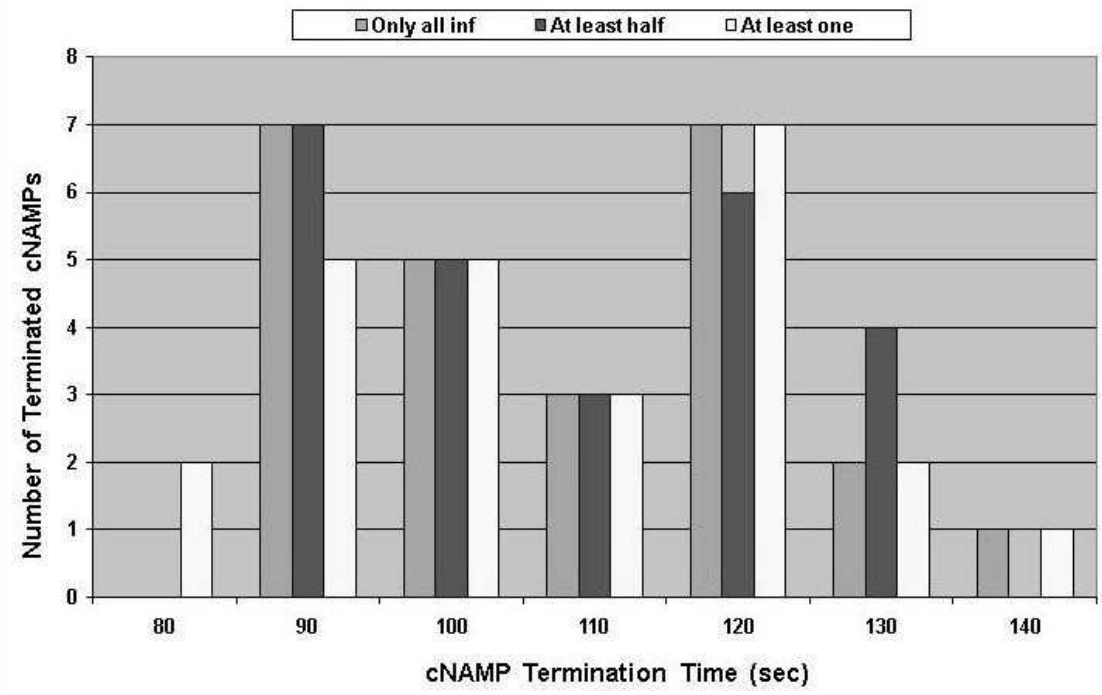


Figure 4.5: Completion Time of 25 cNAMPs

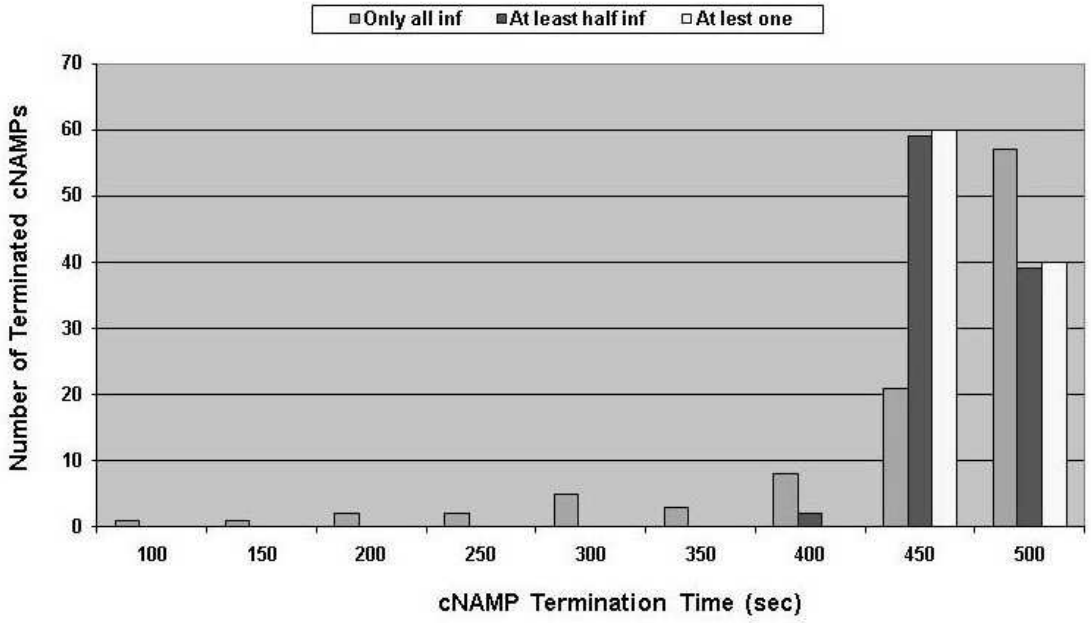


Figure 4.6: Completion Time of 100 cNAMPs

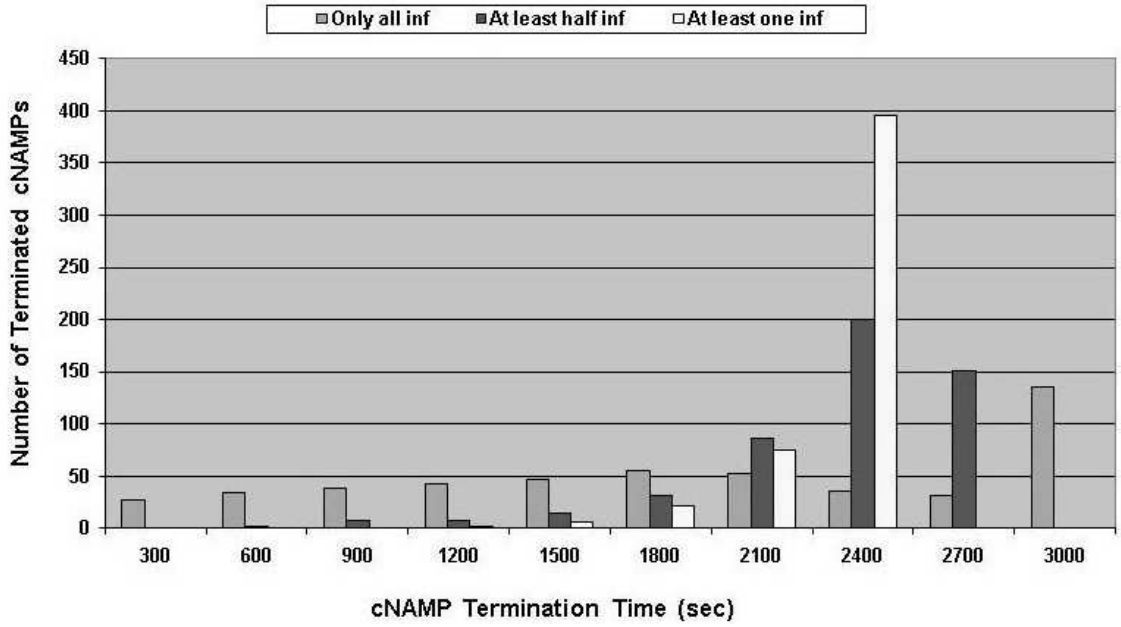


Figure 4.7: Completion Time of 500 cNAMPs

Table 4.3 shows the mean and the median termination time for 25, 100, and 500 cNAMP. For a small number of cNAMPs the recalculation scheme has insignificant impact on the mean and the median values. For a large number of cNAMPs the scheme with

Number of cNAMPs and average values	cNAMP Termination Time (sec)		
	Information about all locations is available	Information about at least half locations is available	Information about at least one location is available
25 cNAMPs			
mean	114	113	113
median	118	117	116
100 cNAMPs			
mean	419	444	444
median	455	440	440
500 cNAMPs			
mean	1804	2073	2144
median	1842	2182	2219

Table 4.3: Mean and Median cNAMP Completion Time

available information about all locations shows the best performance. Therefore, the scheme where cNAMPs can recalculate parameters only when no cNAMP from the same location awaits a response is chosen.

4.4.3 Summary

Negotiating AMPs with competitive scheme (cNAMPs) are proposed to reduce the number of redundant movements caused by location thrashing. In short, the algorithm of cNAMPs is as follows. Before moving a cNAMP sends a request to the target location. The request confirms the movement if remaining execution time on the target location and time to transfer the cNAMP is less than remaining execution time on the initial location. While awaiting the response the cNAMP continues execution. The remaining cNAMPs from the same location do not recalculate their parameters until the cNAMP has received the response. If the movement decision is confirmed the cNAMP moves, otherwise it resumes execution on the current location. Each location has two values of the number of cNAMPs on the current location: actual load and committed load. cNAMPs negotiate via load servers. Pseudocode for a cNAMP and a load server

```
while work remains to execute
   {
    if outstanding request & positive response
       {
        inform local load server about movement
        move to target location
       }
    else if no cNAMP awaits a response on the current location
       { for n from 1 to total number of locations
           find minimum of T_n + T_comm
        if T_h > minimum
           { send request to L_n
            inform local load server about request sent
           }
       }
    continue execution
   }
```

Figure 4.8: cNAMP Pseudocode

```
forever do
case local cNAMP sent a request to location Loc_i:
    lock information about Loc_i
case local cNAMP received response from Loc_i:
   {
    renew and unlock information about Loc_i
    if positive response
       reduce actual and committed loads
   }
case arrival notification from remote cNAMP:
    increase committed load
case cNAMP arrived:
   increase actual load
```

Figure 4.9: Load Server Pseudocode

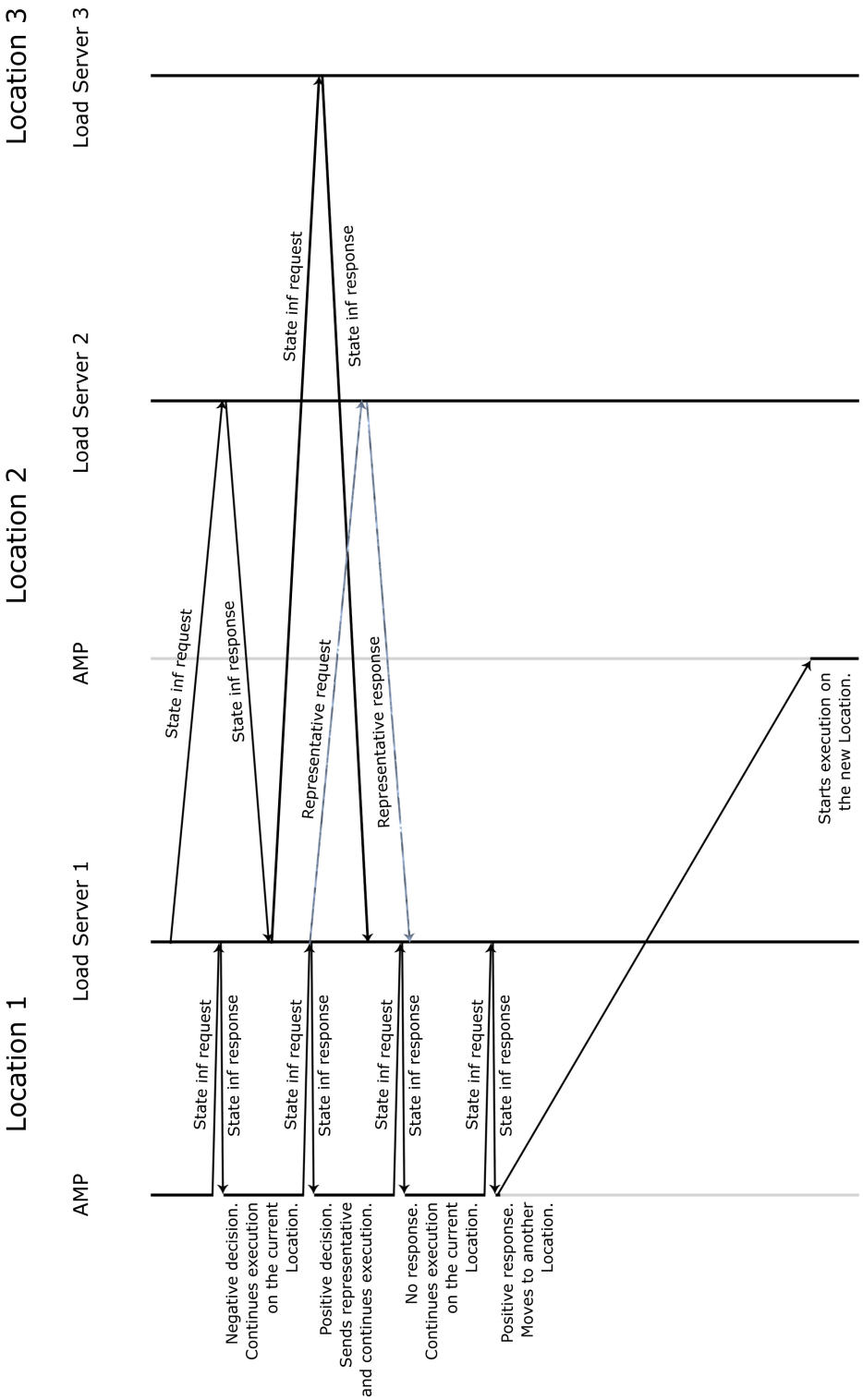


Figure 4.10: cNAMP UML Diagram

is presented in Figures 4.8 and 4.9 respectively. A UML diagram involving locations, load servers, cNAMPs, and auxiliary messages is presented in Figure 4.10.

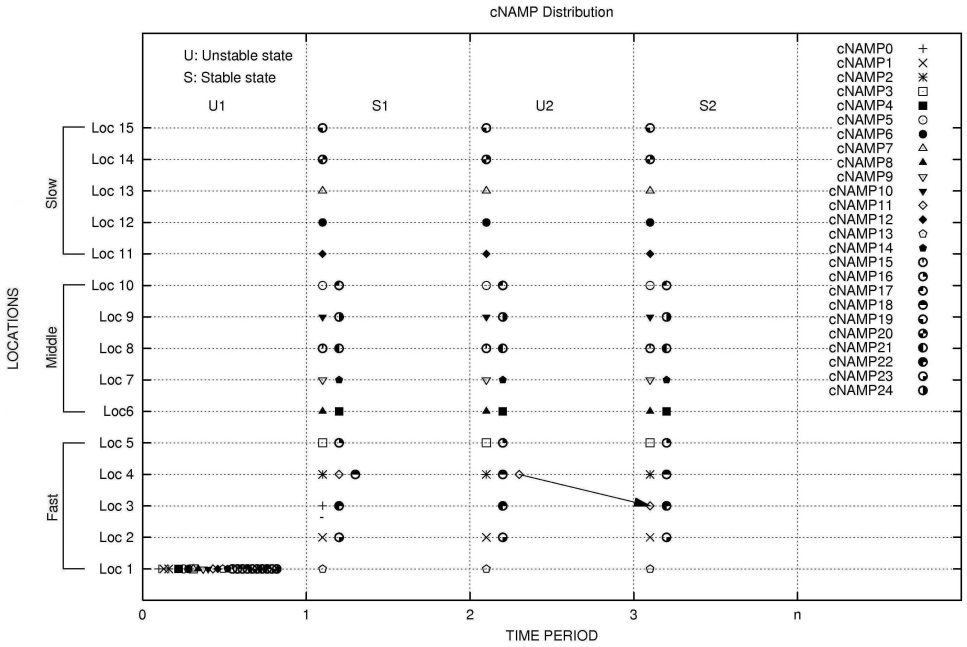
4.5 Comparative cNAMP and AMP Performance

The greedy effects exhibited by AMPs and cNAMPs are compared in Table 4.4 using the experiment design presented in Section 4.3.1. For each scenario the first and the second rows show results of AMP and cNAMP experiments respectively.

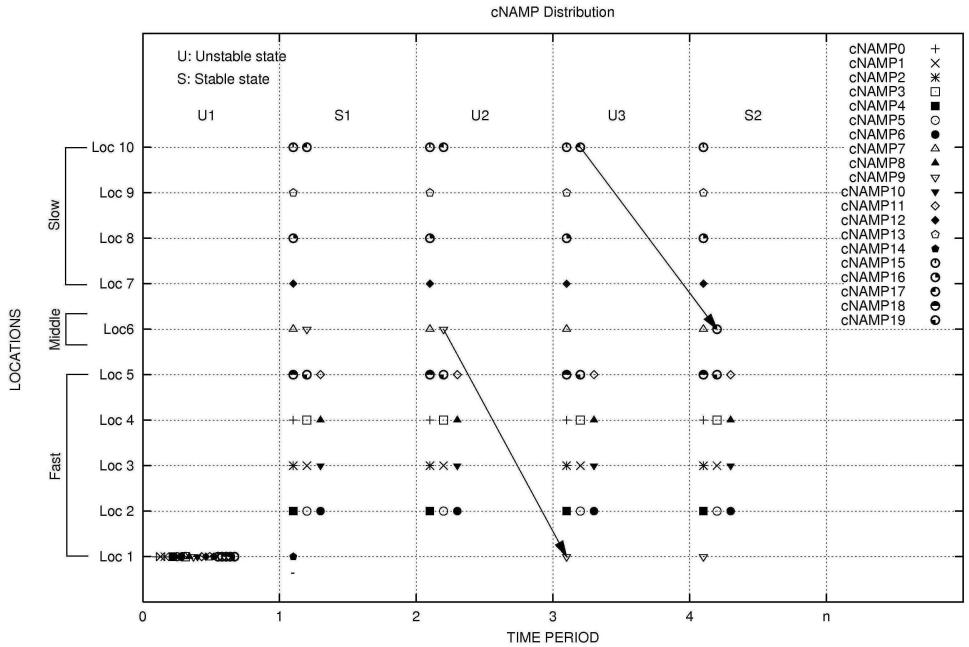
The results show that *even simple negotiation in cNAMPs significantly reduces the number of movements and time to rebalance. cNAMPs do not make redundant movements*

Configuration and type of experiment	Initial distribution		Rebalancing after an AMP/cNAMP termination		Large AMP/cNAMP completion time, sec	
	Time, sec	Mean number of redundant movements	Time, sec	Mean number of redundant movements	Mean	Standard deviation
Scenario 1						
AMPs	60.4	64	22.5	6	173.8	7.66
cNAMPs	14.7	-	5.9	-	104.8	12.9
Reduction	4.11 times	64 moves	3.81 times	6 moves	1.65 times	
Scenario 2						
AMPs	50.5	43	28.2	11	182.1	11.5
cNAMPs	12.4	-	7.8	1	113.6	9.43
Reduction	4.07 times	43 moves	3.62 times	10 moves	1.6 times	
Scenario 3						
AMPs	26.8	13	14.1	6	232.6	9.91
cNAMPs	8.5	-	5.6	-	142.2	4.97
Reduction	3.15 times	13 moves	2.52 times	6 moves	1.64 times	

Table 4.4: Comparative Summary of AMP and cNAMP Greedy Effects



(a) Scenario 1



(b) Scenario 2

Figure 4.11: Initial Distribution and Rebalancing

during initial distribution, and all scenarios show at least three times faster initial balancing in comparison with AMPs, e.g. dropping from 60.4s to 14.7s in scenario 1.

During rebalancing after an AMP/cNAMP termination, cNAMPs make far fewer redundant movements. The vast majority of experiments in scenarios 1 and 3 show that cNAMPs do not make redundant movements to rebalance, and the cNAMP rebalancing takes less than half of the time of AMP rebalancing, e.g. to rebalance 19 AMPs take 28.2s, and 19 cNAMPs take 7.8s in scenario 2.

cNAMPs require less completion time than AMPs. In the analysed worst case, mean cNAMP completion is 0.63 of AMP completion time, e.g. mean completion time of 10 AMPs is 232s, whereas mean completion time of 10 cNAMPs is 142s in scenario 3.

Figures 4.11(a) and 4.11(b) show initial cNAMP distribution and system rebalancing after a cNAMP termination. Arrows show cNAMP movements, and as before the cNAMP movements from state U1 to state S1 are not shown. Figure 4.11(a) should be compared with Figures 4.4(a) and 4.4(b) in Section 4.3. The results show that cNAMPs only display location blindness (Section 4.1) that does not increase cNAMP completion time.

4.5.1 cNAMP Overhead

Table 4.5 presents the overhead of cNAMPs compared with AMPs. Although cNAMPs send more messages (columns 2 and 6), most of them are small request/response mes-

Conf.	AMPs		cNAMPs					
	No. Moves	**Size, Mb**	No. Req/ Resp	No. Moves cNAMPs	**Total No. Msg**	Req/Resp Size, Mb	cNAMP Size, Mb	**Total Size, Mb**
Sc. 1	**106**	**954**	78	35	**113**	36	315	**351**
Sc. 2	**77**	**693**	66	28	**94**	30	252	**282**
Sc. 3	**28**	**252**	26	11	**37**	12	99	**111**

Table 4.5: AMP/cNAMP and Request/Response Messages

sages. Therefore, total communicated data for AMPs is larger than for cNAMPs (columns 3 and 9). As a consequence cNAMPs execute faster than AMPs (column 6 in Table 4.4).

4.6 Discussion

In this chapter two types of greedy effects in AMP systems have been identified: location thrashing causes additional movements *and* increase in AMP completion time; location blindness causes only additional movements, as all transferred AMPs improve their execution environment. Both greedy effects appear in the load balancing literature (Section 4.1).

The chapter has simulated the greedy effects in AMP implementations (Section 4.2), and shown that each AMP makes on average two redundant movements during execution for the scenarios considered. Although greedy effects have limited impact on networks with a small number of AMPs, few locations, or small AMPs, their effects increase as any of these factors scale. The analysis of the redundant movement types and the reasons they occur have shown that redundant movements are mainly caused by location thrashing (Section 4.3).

To reduce location thrashing the concept of negotiating AMPs has been introduced, and AMPs that negotiate with a competitive scheme (cNAMPs) have been described and implemented. Negotiation only implies a coordination among competitive and self-interested agents. The key differences between cNAMPs and AMPs are as follows: before the movement a cNAMP sends a request to the target location; *and* each location has two values for the number of cNAMPs, one that is used by local cNAMPs, and another that is published to other locations (Section 4.4).

cNAMP simulation results show that cNAMPs exhibit only the location blindness. cNAMPs do not make redundant movements during initial distribution, and all scenarios show at least three times faster initial balancing in comparison with AMPs, e.g. dropping from

60.4s to 16.5s in scenario 1. During rebalancing after an AMP/cNAMP termination, cNAMPs make far fewer redundant movements, and the cNAMP rebalancing takes less than half of the time of AMP rebalancing, e.g. to rebalance 19 AMPs take 28.2s, and 19 cNAMPs take 7.8s in scenario 2. cNAMPs require less completion time than AMPs. In the analysed worst case, mean cNAMP completion is 0.63 of AMP completion time, e.g. mean completion times of 10 AMPs and 10 cNAMPs in scenario 3 are 232s and 142s respectively (Section 4.5).

Chapter 5 investigates balanced states of both AMPs and cNAMPs, and provides theoretical analysis of the maximum number of redundant movements in collections of cNAMPs and its probability.

Chapter 5

Theoretical Analysis of Balanced States and Redundant Movements

To estimate the degree of redundant movements in modified AMPs (cNAMPs) discussed in Sections 4.4 and 4.5 this chapter establishes the properties of collections of AMPs in balanced states (Section 5.1) using a set of consistent and rigorous definitions about AMP and cNAMP behaviour (Glossary). Applying the properties of balanced states the chapter then investigates upper bounds on the number of redundant movements and their probabilities after a cNAMP termination in homogeneous and heterogeneous networks (Section 5.2).

5.1 Properties of Balanced Networks

The properties of balanced states are established to analyse the significance of greedy effects. The properties and definitions developed in this section are essential for the proofs in Section 5.2, and are summarised in the Glossary. Note that the properties of balanced states apply to all AMPs, not only cNAMPs. The balanced states are investigated

using a balanced state checker, i.e. a special program to explore the state space the system enters. The balanced state checker uses identical logic to the simulated and to real AMPs. Hence, it is not surprising that the predicted balanced states exactly reproduce simulated results for all scenarios considered. A homogeneous network is analysed as a special case of a heterogeneous network where the root location is taken as a *singleton subnetwork*, i.e. a subnetwork with one location. A *subnetwork* is used to define a set of locations with identical available speeds.

There is a similarity between AMP's balanced states and Nash equilibria [FT91]. Participants in both Nash equilibrium and AMPs aim to get as much *profit* as possible. However, in game theory participants have a choice to play cooperatively or not, whereas AMPs/cNAMPs have no such choice. Though cNAMPs communicate more information than AMPs, cNAMPs are still *selfish*, and they never *care* about others' profit but only about decreasing their own execution time. Another difference is that AMPs/cNAMPs do not predict or estimate other AMP/cNAMP behaviour, i.e. as soon as an AMP/cNAMP finds a location where it can reduce its execution time it moves.

The empirical balanced state checker that is used to investigate balanced states is discussed in Section 5.1.1. The *independent balance property* is presented in Section 5.1.2, and discussion of optimal and near-optimal balanced states follows in Sections 5.1.3 and 5.1.4 respectively. Finally, balanced states for homogeneous networks are characterised in Section 5.1.5 and for heterogeneous networks are characterised in Section 5.1.6.

5.1.1 Balanced State Checker

The properties of balanced states were investigated using a program to explore the states entered as AMPs are added to the system one by one. The program assumes that communication time is negligibly small relative to computation time, i.e. $T_{comm} \ll T_{comp}$. The sequence of stable states that are entered are all balanced.

The algorithm of the balanced state checker is as follows. To allocate a new AMP when a

```
/ all subnetworks are singleton
/ q is the total number of subnetworks
/ S_i is the available speed in subnetwork i
/ x_i is the number of AMPs in subnetwork i, initially all x_i = 0
/ R_i is the AMP relative speed in subnetwork i
/ numAMPs is the total number of AMPs, e.g. 3000
/ list_Rmax is a list of subnetworks where R_i == R_max
/ num_Rmax number of subnetworks that have R_i == R_max

j = 1;
while j <= numAMPs
   { for i from 1 to q by 1
         Increase x_i by one;  Calculate R_i = S_i / x_i
     Find R_max;
     num_Rmax = 0;  list_Rmax = empty
     for i from 1 to q by 1
         if R_i < R_max then Decrease x_i by one
         else remember i in list_Rmax;  num_Rmax++

     if num_Rmax > 1
         { if numAMPs >= j + num_Rmax - 1 then num_floatAMPs = num_Rmax
           else num_floatAMPs = numAMPs - j + 1
           for i from 1 to num_floatAMPs - 1 by 1
               { num_distr = q! / (i! * (num_Rmax - i)!)
                 Print num_distr distributions of (j + i - 1) AMPs }
           if num_floatAMPs == q
               then print distribution of (j + num_Rmax - 1) AMPs
           j = j + num_Rmax }
     else
         { Print distribution of j AMPs
           j++ }
```

Figure 5.1: Pseudocode of the Balanced State Checker

distribution of x AMPs is given ($0 \leqslant x \leqslant \infty$) one AMP is added to each location. Then AMP relative speeds are calculated, and one AMP is removed from each location except those with the highest AMP relative speed. The resulting distribution is the distribution $x + 1$ AMPs where this one AMP can be on one of the locations with the highest AMP relative speed. The rationale of the method is based on the main rule governing AMP movements (2.5), i.e. an AMP moves if execution time on the current location exceeds execution time on the next location and communication delay. As communication time is relatively small, when a new AMP is put on any location, the AMP only checks whether its relative speed on the next location is higher than on the current location. If a few locations have the same highest AMP relative speed then distribution of $x + 1$ AMPs may result in a number of balanced state and this *floating AMP* can be situated on any of the locations with the highest relative speed. The number of distributions is defined by the combination of the number of floating AMPs, $N_{floatAMPs}$, on the number of locations that have the same highest AMP relative speed, N_{Rmax}, i.e.

$$\binom{N_{Rmax}}{N_{floatAMPs}} = \frac{N_{Rmax}!}{N_{floatAMPs}!(N_{Rmax} - N_{floatAMPs})!}. \tag{5.1}$$

Pseudocode of the balanced state checker is presented in Figure 5.1 where x_i is the number of AMPs in subnetwork i when j AMPs are distributed on q subnetworks.

The program was run on a large number of networks and the balanced states were noted. The networks had up to 3000 AMPs split into between 2 and 20 subnetworks. All observations are made on the basis of heterogeneous networks. The program code is provided in Appendix B.

5.1.2 Independent Balance Property

The analysis of balanced states shows the following property.

Property 1 (independent balance)**.** *For a balanced state, the relationship between the number of AMPs x_i and x_j on locations in any two subnetworks i and j is independent*

(a) Scenario A

Available speed of locations	Number of AMPs
S_1	8
S_2	3

(b) Scenario B

Available speed of locations	Number of AMPs
S_1	8
S_2	3
$\vdots$	$\vdots$
S_i	x_i

(c) Scenario C

Available speed of locations	Number of AMPs
S_1	8 $\vdots$ 8
S_2	3 $\vdots$ 3
$\vdots$	$\vdots$
S_i	x_i

Table 5.1: AMP Distribution in a Pair of Subnetworks for a Given Sum of AMPs

of the number of locations in those subnetworks and independent of the presence or absence of other subnetworks, subject only to the sum $x = x_i + x_j$ being constant. The only exception to this rule is the case when distribution of $x' = x_i + x_j + 1$ AMPs results in all x' AMPs having the same relative speed. In this case the partition may have two variants.

The independent balance property holds in all scenarios investigated and can be observed in the following experiments:

Experiment 1.

- Distribute 11 AMPs over a heterogeneous network of two singleton subnetworks. Without loss of generality, make the following assumptions: the distribution places 8 AMPs on the location of subnetwork 1 and 3 AMPs on the location of subnetwork 2 as Table 5.0(a) shows, *and* distribution of 12 AMPs *would not* result in AMPs having the same relative speed.

- Add further singleton subnetworks to the system and also add enough AMPs to the new system so that balanced state is achieved and there are 11 AMPs distrib-

uted between singleton subnetworks 1 and 2. The distribution between singleton subnetworks 1 and 2 will *always* be 8 AMPs in subnetwork 1 and 3 AMPs in subnetwork 2 (Table 5.0(b)). It will *never* be distributed 7 AMPs in subnetwork 1 and 4 AMPs in subnetwork 2, or 9 AMPs in subnetwork 1 and 2 AMPs in subnetwork 2.

- Adding any number of locations to a subnetwork will not change the distribution between a pair of locations from subnetworks 1 and 2 so long as the system is in a balanced state and the sum of the number of AMPs on the pair of locations from subnetworks 1 and 2 is 11 (Table 5.0(c)).

Experiment 2. Assume that distribution of 12 AMPs between subnetworks 1 and 2 *would* result in all AMPs having the same relative speed, and the distribution would be 8 AMPs in subnetwork 1 and 4 AMPs in subnetwork 2. Then distribution of 11 AMPs has two alternatives: 8 AMPs in subnetwork 1 and 3 AMPs in subnetwork 2, *and* 7 AMPs in subnetwork 1 and 4 AMPs in subnetwork 2. The distribution of 11 AMPs may result only in these two partitions independently of presence or absence of other locations and subnetworks.

Let there be two subnetworks, with the available speeds S_1 and S_2, and let each location of subnetworks 1 and 2 have x_1 and x_2 AMPs in a balanced state respectively. Then the following inequalities hold in *any* balanced state:

$$\begin{cases} \dfrac{S_1}{x_1} \geqslant \dfrac{S_2}{x_2+1} \\ \dfrac{S_2}{x_2} \geqslant \dfrac{S_1}{x_1+1}. \end{cases} \tag{5.2}$$

There are two types of balanced states: optimal and near-optimal balanced states. The properties of balanced states and generalisation of some definitions given in [Den07] are presented in Sections 5.1.3 and 5.1.4.

5.1.3 Optimal Balance

As discussed in Section 3.3.1 in an optimal balanced state locations with the same available speed have equal numbers of AMPs. Thus, the total number of AMPs x in an optimally balanced network with q subnetworks is:

$$x = \sum_{i=1}^{q} x_i N_i,$$

where there are N_i locations in subnetwork i and each location has x_i AMPs.

Property 2 (singleton optimality)**.** *All balanced states which a network of singleton subnetworks enters are optimally balanced.*

The *singleton optimal property* is a direct corollary of the optimal balanced state definition. Furthermore, from optimal balanced state definition and independent balance property the following *optimal balance properties* result:

1. finding optimal balance states of arbitrary heterogeneous networks only requires the finding of the optimal balance states for networks with the same number of singleton subnetworks:

$$k = \sum_{i=1}^{q} x_i, \quad k \in [1; +\infty). \tag{5.3}$$

2. any optimally balanced network is a composition of optimally balanced pairs of subnetworks.

Solving inequations (5.2) for a heterogeneous network of two subnetworks with available speeds S_1 and S_2, and $k = x_1 + x_2$, each location has the following number of AMPs:

$$x_i \approx \frac{S_i(k+1)}{S_1 + S_2} - \frac{1}{2}, \qquad i = 1, 2. \tag{5.4}$$

Here, by $\approx$ rounding to the nearest value is meant. If the fraction part of x_i is exactly .5, then either x_i is rounded up and x_j is rounded down, *or* x_i is rounded down and x_j

is rounded up. To indicate the rounding to the nearest value in the thesis double square brackets, $[\![]\!]$ are used. These brackets also imply that both adjacent integer values should be considered if the fractional part of the contents is exactly .5.

This analysis of balanced and optimal balanced states enables testing of an assumption made in [Den07], i.e. it is said that AMPs tend to have equal AMP relative speed in optimal balance. However, the observations show the following results:

Lemma 3. *An AMP relative speed on a faster location tends to be slightly lower than the mean AMP relative speed; and an AMP relative speed on a slower location tends to be slightly higher than the mean AMP relative speed.*

Proof. This can be observed from the following analysis. First, it is important to recall that although a system is equilibrium in an optimal balanced state, AMP relative speeds on locations from different subnetworks need not be the same, because the number of AMPs on a location is integer.

According to assumption in [Den07], i.e. $\frac{S_1}{x_1} \approx \frac{S_2}{x_2}$, and the optimal balance property a location of subnetwork 1 must have $\left[\!\left[\frac{S_1 k}{S_1+S_2}\right]\!\right]$ AMPs. Hence, AMP relative speed on the basis of (2.3) must be

$$R_1' = \frac{S_1}{\left[\!\left[\frac{S_1 k}{S_1+S_2}\right]\!\right]}. \tag{5.5}$$

However, the number of AMPs on a location is given by (5.4), and an AMP relative speed on a location of subnetwork 1 is

$$R_1 = \frac{S_1}{\left[\!\left[\frac{S_1 k}{S_1+S_2} + \frac{S_1-S_2}{2(S_1+S_2)}\right]\!\right]}. \tag{5.6}$$

As $S_1 > S_2 > 0$ and $\frac{S_1-S_2}{S_1+S_2} < 1$, then $0 < \frac{S_1-S_2}{2(S_1+S_2)} < 0.5$. Thus, $\frac{S_1-S_2}{2(S_1+S_2)}$ increases the number of AMPs assumed in [Den07] by zero or one during the rounding. Therefore, an AMP relative speed on a location of subnetwork 1 either coincides with the AMP relative speed assumed in [Den07] or is slightly slower. The same analysis sequence applied to

AMP relative speed in subnetwork 2 shows that it is either coincides with [Den07] or it is slightly faster.

The difference between AMP relative speeds on locations from different subnetworks is less when the difference in the available speeds of locations is small; and it decreases as the total number of AMPs increases. □

5.1.4 Near-Optimal Balance

For any optimal balanced state, the *optimal number of AMPs* for a subnetwork is the number of AMPs on each location in the subnetwork. *The nearest upper (lower) optimal balanced state* is the optimal balanced state which the system enters by adding (removing) the minimum number of AMPs.

A *near-optimal balanced* network is defined to be the network where some networks have near-optimal number of AMPs. The locations of these underloaded subnetworks have either the optimal number of AMPs or one less than the optimal number. The underloaded subnetworks are determined by being the subnetworks with the slowest AMP relative speed in the nearest upper optimal balanced state.

Property 4 (consecutive optimality)**.** *If a system with a total of x AMPs is optimally balanced, and the subnetwork with the highest AMP relative speed is a singleton, then the system with a total of $x + 1$ AMPs is also optimally balanced.*

The consecutive optimal property is derived from the near-optimal balance definition as a corollary.

The independent balance property (Section 5.1.2) allows us to conclude that *a system has no balanced states other than optimal and near-optimal balanced states*. That is, only subnetworks that have the slowest AMP relative speed in the nearest upper optimal

balanced state can be in a near-optimal balanced state. If other networks were simultaneously in a near-optimal balance state, then there would be an immediate transfer of AMPs from the slower to the faster subnetwork (in terms of AMP relative speed).

As the near-optimally balanced subnetworks are determined for each optimally balanced state, a subnetwork which may change its number of AMPs after adding/removing an AMP is identified as follows:

- when an AMP is *added* to an optimally balanced network, the number of AMPs in the subnetwork which would have the highest AMP relative speed if one AMP is added to each location increases by one.
- when an AMP is *removed* from an optimally balanced network, the number of AMPs in the subnetwork with the slowest AMP relative speed decreases by one.

Frequently, this calculation will give a unique successor state. However, if several subnetworks have the same AMP relative speed and it is the highest AMP relative speed *and* AMP is being added, the AMP can be added to any one of those subnetworks. Similarly if an AMP is being removed and several subnetworks have the same AMP relative speed *and* it is the lowest, then the AMP may be removed from any one of those subnetworks.

5.1.5 Characterizing Balanced States in a Homogeneous Network

Let a homogeneous network have N locations and the available speed of non-root locations be S. As only a part of the root location capacity is available for an AMP execution, let the load factor, $0 < f < 1$, define available resources for AMPs on the root location, i.e. available speed of the root location is $f \cdot S$.

In an *optimal balanced state* on the basis of (5.4) the system has the following numbers of AMPs on the root, x_{rt}, and non-root, x_{nrt}, locations:

$$x_{rt} = \left[\left[\frac{2fk + f - 1}{2(f+1)}\right]\right], \tag{5.7}$$

$$x_{nrt} = \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]. \tag{5.8}$$

In a *near-optimal balanced state* the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ AMPs and non-root locations have either $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ AMPs. For further distinguishing non-root locations with a different number of AMPs, those which have an optimal number of AMPs, i.e. $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$, are called *heavy* locations, and those which have one or two AMPs less than a heavy location, i.e. $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ and $\left[\left[\frac{2k-5f-3}{2(f+1)}\right]\right]$, are called *light* and *very light* locations respectively.

5.1.6 Characterizing Balanced States in a Heterogeneous Network

Let $k_{i,j}$ be the sum of AMPs in singleton subnetworks i and j, i.e.

$$k_{i,j} = x_i + x_j, \qquad i,j = 1,2...,q, \quad i < j. \tag{5.9}$$

To calculate the AMP distribution in optimally balanced heterogeneous networks the following algorithm is used.

1. We calculate $k_{i,j}$ for all i and j denoted in (5.9) using the following equation:

$$k_{i,j} = \left[\left[\frac{(2k+q)(S_i+S_j)}{2\sum_{m=1}^{q} S_m} - 1\right]\right]. \tag{5.10}$$

2. Then for each $k_{i,j}$ we find x_i and x_j using (5.4), i.e.

$$\begin{cases} x_i = \left[\left[\dfrac{S_i(k_{i,j}+1)}{S_i+S_j} - \dfrac{1}{2}\right]\right], \\ \\ x_j = \left[\left[\dfrac{S_j(k_{i,j}+1)}{S_i+S_j} - \dfrac{1}{2}\right]\right]. \end{cases} \tag{5.11}$$

3. Using results from (5.11) we make a list of all possible distributions, and delete distributions where $k \neq \sum_{m=1}^{q} x_m$.

4. In the remaining distributions we check inequality (5.2). The inequality is strictly larger for all pairs i and j, i.e.

$$\frac{S_i}{x_i} > \frac{S_j}{x_j + 1}, \quad i, j = 1, 2, ...q. \tag{5.12}$$

The only exception is for pairs where x_i and x_j result in .5. In this case (5.2) is as follows:

$$\begin{cases} \dfrac{S_i}{\lceil x_i \rceil} = \dfrac{S_j}{\lfloor x_j \rfloor + 1}, \\ \\ \dfrac{S_j}{\lceil x_j \rceil} = \dfrac{S_i}{\lfloor x_i \rfloor + 1}. \end{cases} \tag{5.13}$$

The distributions in which condition (5.13) holds are the only distributions of k AMPs on the network of q singleton subnetworks. An example of the calculation is provided in Appendix C.

To estimate the correctness of the above algorithm's distributions of a number of AMPs for different configurations have been calculated using the algorithm and the empirical balanced state checker discussed in Section 5.1.1. First, the AMP distributions x_i and x_i^{sim} have been calculated for up to 50 subnetworks, ranging k from 1 to 10000. The ratio between available speeds of fastest and slowest locations ranges from 1.1 to 14000. The results show that distributions x_i and x_i^{sim} are identical. Therefore, the presented algorithm is a good way of calculating AMP distribution in optimally balanced heterogeneous networks. According to discussion in Section 5.1.4, numbers of AMPs on locations in a *near-optimal balanced state* coincides with the numbers in the nearest upper optimal balanced state, except the subnetworks with the slowest AMP relative speed. Locations of these subnetworks have number of AMPs given in the algorithm or one AMP less.

5.2 cNAMP Greedy Effect Analysis

This section examines the cost of cNAMP location blindness by predicting the maximum number of redundant movements in homogeneous networks (Section 5.2.1) and hetero-

Cost Model Components:

$$T_h = \frac{W_r}{R_h} \tag{5.14}$$

$$T_n = \frac{W_r}{R_n} \tag{5.15}$$

f - the load factor on the root location

R_h - the current location relative speed

R_n - the new location relative speed

T_h - execution time on the current location

T_n - execution time on the new location

W_r - the work remaining

x - the total number of cNAMPs

x_{nrt} - the number of cNAMPs on a non-root location

x_{rt} - the number of cNAMPs on the root location

Figure 5.2: cNAMP Cost Model Components

geneous networks (Section 5.2.2). To estimate the number of redundant movements the conditions under which cNAMPs transfer are analyzed. Figure 5.2 shows the cNAMP cost model. A network with AMPs, and hence with cNAMPs, enters one of two types of balanced state: optimal and near-optimal balance. The analysis presents the maximum number of movements and its probability after a cNAMP termination in homogeneous and heterogeneous networks which are in optimally and near-optimally balanced states.

5.2.1 Homogeneous Network

cNAMP Termination in an Optimally Balanced Network. From (5.7) and (5.8), in an optimal balanced state the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs and every non-root location has $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs.

Theorem 5. *There is no greedy effect when a cNAMP terminates in an optimally bal-*

anced homogeneous network.

Proof. The proof proceeds by case analysis on the location where termination occurs, and where the first movement is initiated.

1) Termination at a Non-root Location. Assume that a cNAMP terminates at a non-root location. After the cNAMP termination, the system has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs on the root location, $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ cNAMPs on the non-root location where termination occurred (the light location), and $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on the remaining non-root locations (i.e. heavy locations). Hence, the system is in the near-optimal balanced state. Possible movements which may occur from the root and heavy locations to the light location are analysed below.

1. *cNAMP movement from the root location.* The main rule on the basis of which cNAMPs make the decision about a movement is presented in (2.5), i.e. $T_h > T_n + T_{comm}$. According to the rule, to make the decision the cNAMP needs to know its execution time on the current location, T_h, and its execution time on the new location after the cNAMP arrival, T_n.

   First, on the basis of (5.14) and (2.3) the cNAMP execution time on the root location before a cNAMP movement is calculated:

   $$T_h = \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk + f - 1}{2(f+1)}\right]\right]. \tag{5.16}$$

   The light location becomes a heavy location after a cNAMP arrival, and has $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs. According to (5.15) and (2.3) execution time is

   $$T_n = \frac{W_r}{S} \cdot \left[\left[\frac{2k - f + 1}{2(f+1)}\right]\right]. \tag{5.17}$$

   Substituting (5.16) and (5.17) in (2.5) gives

   $$\frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk + f - 1}{2(f+1)}\right]\right] > \frac{W_r}{S} \cdot \left[\left[\frac{2k - f + 1}{2(f+1)}\right]\right] + T_{comm}. \tag{5.18}$$

If condition (5.18) holds then the cNAMP moves from the root to the light location. The system enters another optimal balanced state and *cannot have any more movements*.

2. *cNAMP movement from a heavy location.* Another movement is from a heavy to the light location. cNAMP execution time on a heavy location is

$$T_h = \frac{W_r}{S} \cdot \left[\left[\frac{2k - f + 1}{2(f + 1)}\right]\right]. \tag{5.19}$$

After a cNAMP movement to the light location, the light location becomes heavy and execution time there is given by (5.17). Substituting (5.19) and (5.17) in (2.5) gives the following cNAMP movement condition:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k - f + 1}{2(f + 1)}\right]\right] > \frac{W_r}{S} \cdot \left[\left[\frac{2k - f + 1}{2(f + 1)}\right]\right] + T_{comm}$$

or

$$T_{comm} < 0. \tag{5.20}$$

In fact, (5.20) shows that *the number of cNAMPs on locations with the same available speed must differ by at least two before cNAMPs will move between them*. This condition is called the *minimum difference criterion*.

2) cNAMP Termination at the Root Location. Assume that a cNAMP terminates at the root location in an optimal balanced state. Then the root location has $\left[\left[\frac{2fk-f-3}{2(f+1)}\right]\right]$ cNAMPs, and non-root locations have $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs.

Applying the same principle to analyse as before shows that *after a cNAMP termination from the root location in an optimally balanced homogeneous network, only one movement may occur for rebalancing* (Appendix D.1). □

The above analysis leads to the conclusion that *after a cNAMP termination in an optimally balanced homogeneous network, there can be only one cNAMP movement between locations to rebalance and, hence, there is no greedy effect*.

cNAMP Termination in a Near-Optimally Balanced Network. In near-optimal balance a system has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs on the root location, and either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on non-root locations (Section 5.1.5).

Theorem 6. *The greedy effect causes at most one redundant movement, when a cNAMP terminates in a near-optimally balanced homogeneous network.*

Proof of Theorem 6 follows directly from Lemma 7.

Lemma 7. *A redundant movement occurs only in two cases: of a cNAMP termination in near-optimal balance on the root location which is discovered first by a cNAMP from a light location, and of a cNAMP termination in near-optimal balance on a light location which is discovered first by a cNAMP from the root location.*

Recall that a light location has one cNAMP less than the optimal number (Glossary). The proof of Lemma 7 again proceeds by case analysis considering the location where termination occurs, and the location where the first movement is initiated (Appendix D.2).

Probability of the Greedy Effect after a cNAMP termination. In a homogeneous network the greedy effect occurs only in two cases after a cNAMP termination from a near-optimal balanced state:

- a cNAMP terminates at the root location, and then a cNAMP from a light location discovers the opportunity to move first, i.e. the following condition must hold:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right] > \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] + T_{comm}.$$

- a cNAMP terminates at a light location, and then a cNAMP from the root location discovers the opportunity to move first, i.e. the following condition must hold:

$$\frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] > \frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right] + T_{comm}.$$

Thus, the greedy effect probability, P, in a near-optimally balanced homogeneous network is the sum of probabilities of these two events:

$$P = P_1 + P_2. \tag{5.21}$$

The probability P_1 that a cNAMP terminates from the root location, and then a cNAMP from a light location discovers the opportunity to move first is a product of the probability of a cNAMP termination from the root location, P_{termR}, and the probability of the discovery of the better opportunity for execution first by a cNAMP from a light location, P_l:

$$P_1 = P_{termR} \cdot P_l. \tag{5.22}$$

To calculate the probability of a cNAMP termination at the root location, assume that the cNAMP execution time on locations follows an exponential distribution. The mean cNAMP execution time on the root, heavy and light locations is given by:

$$T_{loc} = \frac{W \cdot x_{loc}}{S_{loc}} = \frac{W}{R_{loc}}. \tag{5.23}$$

Hence, the rate of cNAMP terminations at the root, heavy and light locations is:

$$\nu_{loc} = \frac{R_{loc}}{W}.$$

Assume, that there are N_l light locations and N_h heavy locations in the system. Then the probability that a cNAMP terminates at the root location is:

$$P_{termR} = \frac{\nu_{root}}{\nu_{root} + N_h \cdot \nu_h + N_l \cdot \nu_l}$$

or

$$P_{termR} = \frac{R_{root}}{R_{root} + N_h R_h + N_l R_l}. \tag{5.24}$$

cNAMPs from non-root locations have an equal probability of discovering a better opportunity for execution first. The total number of cNAMPs on non-root locations, $x_{nonroot}$, is

$$x_{nonroot} = N_h x_h + N_l x_l.$$

The probability that a cNAMP from a light location is the first to discover a better opportunity for execution is the ratio of the total number of cNAMPs on light locations to the total number of cNAMPs on non-root locations:

$$P_l = \frac{N_l x_l}{N_l x_l + N_h x_h}. \tag{5.25}$$

Thus, substituting (5.24) and (5.25) in (5.22), gives the following probability, P_1, of cNAMP termination on the root location and further discovery of the opportunity to move by a cNAMP from a light location first:

$$P_1 = \frac{R_{root}}{R_{root} + N_h R_h + N_l R_l} \cdot \frac{N_l x_l}{N_l x_l + N_h x_h}. \tag{5.26}$$

Applying the same principle results in the following probability, P_2, of cNAMP termination on a light location and further discovery of the opportunity to move by a cNAMP from the root location first:

$$P_2 = \frac{N_l R_l}{R_{root} + N_h R_h + N_l R_l} \cdot \frac{x_{root}}{x_{root} + N_h x_h}. \tag{5.27}$$

Substituting (5.26) and (5.27) in (5.21) gives the following probability of the greedy effect after a cNAMP termination from a near-optimally balanced homogeneous network:

$$P = \frac{N_l}{R_{root} + N_h R_h + N_l R_l} \cdot \left(\frac{R_{root} x_l}{N_l x_l + N_h x_h} + \frac{R_l x_{root}}{x_{root} + N_h x_h} \right). \tag{5.28}$$

The range of values that P can take is calculated for homogeneous networks of locations (3193 MHz) by changing the total number of locations from 3 to 50, the number of light locations from 1 to $N - 2$, the load factor, $0.05 \leqslant f \leqslant 0.95$, and the number of cNAMPs, k, from 1 to 200. The calculation considers only cases when $N_h \cdot N_l \neq 0$, because a system must have both heavy and light locations in order the greedy effect can occur. The root and light locations must have at least one cNAMP.

In total the probability is calculated for 3,100,872 states. In the experiments the parameters were placed in loops where they were gradually changing in accordance to their

range. The analysis shows that the number of cNAMPs, k, has no significant effect on the probability. The total number of locations, N, the number of light locations, N_l, and the load factor of the root location, f, have a direct impact on the probability. The maximum probability in the conducted calculations is 32%. It occurs in a homogeneous network of 50 locations where 47 locations are light, load factor of the root location $f = 0.95$ and $k = 195$ cNAMPs, i.e. the total number of cNAMPs is 4948. The mean probability is 4%.

Summary. The analysis of cNAMP movements on homogeneous networks have demonstrated dependence between the number of cNAMPs and locations in optimal and near-optimal balanced states. It shows the following results of cNAMP behaviour after a cNAMP termination in an optimal balanced network:

- there is no greedy effect (Theorem 5).
- cNAMPs do not move, when a system is in a near-optimal balanced state (minimum difference criterion).

When a cNAMP terminates in a near-optimal balanced network the greedy effect only occurs in two case and causes only one redundant movement (Theorem 6). The mean probability of this movement in the conducted experiments is 4%.

5.2.2 Heterogeneous Network

The analysis is made for a heterogeneous network of q subnetworks one of which is the singleton subnetwork corresponding to the root location.

Theorem 8. *The number of redundant movements in a heterogeneous network after a cNAMP termination does not exceed* $q - 1$.

Theorem 8 follows from analysis of cNAMP movements after cNAMP termination from optimally in near-optimally balanced networks.

cNAMP Termination in Optimally Balanced Networks.

Lemma 9. *A system makes at most $q-2$ redundant movements after a cNAMP termination from an optimally balanced heterogeneous network.*

Proof. The proof proceeds by case analysis on the location where termination occurs, and where the first movement is initiated. To analyse possible movements after a cNAMP termination from optimal balance, the subnetworks are numbered in the ascending order of cNAMP relative speeds, i.e.

$$R_1 > R_2 > ... > R_q.$$

According to the minimum difference criterion, a movement after a cNAMP termination from optimal balance in a heterogeneous network can be only from a location of another subnetwork. Assume that a cNAMP terminates from a location of subnetwork i where $1 < i < q$. The analysis is conducted by investigation of possible movements from locations of subnetworks $i-1$ and $i+1$, i.e. $R_{i-1} > R_i > R_{i+1}$.

cNAMP movement from a location of subnetwork $i-1$. cNAMP execution time on a location of subnetwork $i-1$ before the movement is

$$T_h = \frac{W_r}{R_{i-1}}, \tag{5.29}$$

and cNAMP execution time after the movement becomes

$$T_n = \frac{W_r}{R_i}. \tag{5.30}$$

Substituting (5.29) and (5.30) in (2.5) gives the following condition of a cNAMP movement:

$$\frac{W_r}{R_{i-1}} > \frac{W_r}{R_i} + T_{comm}$$

or

$$W_r(\frac{1}{R_{i-1}} - \frac{1}{R_i}) > T_{comm}. \tag{5.31}$$

To satisfy condition (5.31) at least $\frac{1}{R_{i-1}} - \frac{1}{R_i}$ must be positive. However, according to initial condition $R_{i-1} > R_i$, hence $\frac{1}{R_{i-1}} - \frac{1}{R_i} < 0$. Thus, *after a cNAMP termination from optimal balance a cNAMP never moves from a location which in optimal balance has higher cNAMP relative speed to a location which in optimal balance has lower cNAMP relative speed.*

cNAMP movement from a location of subnetwork $i+1$. cNAMP execution time on a location of subnetwork $i+1$ before the movement is

$$T_h = \frac{W_r}{R_{i+1}}, \tag{5.32}$$

and cNAMP execution time on a location of subnetwork i after the movement becomes (5.30).

Thus, condition of the cNAMP movement (2.5) on the basis of (5.32) and (5.30) is

$$\frac{W_r}{R_{i+1}} > \frac{W_r}{R_i} + T_{comm}. \tag{5.33}$$

Hence, the maximum number of cNAMP movements occurs when the following conditions hold:

1. a cNAMP terminates from a location with the highest cNAMP relative speed, i.e. R_1;
2. then cNAMPs from locations of subnetworks i (where $1 < i \leqslant q$) discover better opportunities for execution on locations of subnetworks $i+1$ first, i.e. in the descending order of cNAMP relative speeds;

Therefore, *after a cNAMP termination in an optimally balanced heterogeneous network of q subnetworks there can be at most $q-2$ redundant movements.* □

cNAMP Termination in Near-Optimally Balanced Networks.

Lemma 10. *A system makes at most $q-1$ redundant movements after a cNAMP termination from a near-optimally balanced heterogeneous network.*

The proofs of Lemma 10 again proceed by case analysis on the location where termination occurs, and the location where the first movement is initiated (Appendix D.3).

Probability of the Greedy Effect after a cNAMP Termination. The calculations presented in Appendix D.4 show that *the median probability of $q-2$ redundant movements after a cNAMP termination from optimally balanced heterogeneous network does not exceed 1%.*

As it is difficult to estimate mean and maximum values of the probability for a near-optimally balanced subnetwork, the results of conducted experiments show that the probability is less than 30%, and it rapidly decreases as the number of subnetworks increases (Appendix D.5).

Summary. The analysis of cNAMP movements after a cNAMP termination from *optimally balanced* heterogeneous networks shows that:

- cNAMPs never move from locations which in optimal balance have higher cNAMP relative speed to locations which in optimal balance have lower cNAMP relative speed.
- a system makes at most $q-2$ redundant movements to rebalance (Lemma 9).
- the probability median value of maximum number of redundant movements does not exceed 1%.

Results of analysis after a cNAMP termination from *near-optimally balanced* heterogeneous networks are as follows:

- a system makes at most $q - 1$ redundant movement to rebalances (Theorem 8).
- in the experiments the probability of the maximum number of movements does not exceed 30% and rapidly decreases as the number of subnetworks increase.

5.3 Discussion

The chapter has established the properties of balanced states to further analyse the significance of the greedy effect. Here, AMPs have been analysed as the general case. The chapter has described three properties of balanced states including *independent balance*, *singleton optimality*, *consecutive optimality*, and characterised optimal and near-optimal balanced states for homogeneous and heterogeneous networks (Section 5.1).

The load flattening observed in the balanced states may seem to contradict the discussion in Section 2.8 where it is stated that AMPs do not aim to even load but only to reduce their execution time. It is important to note that this flattening is not an aim but a possible result of the AMP attempts to reduce their completion time. In case AMPs enter a stable state which is not balanced the flattening does not occur but AMPs still reduce their completion time. Whereas in the systems where the goal is to even out the load the distribution is always attempted to be even but this does not necessarily results in a reduction of task or program completion time.

The significance of the greedy effect has been established by predicting the worst case (maximum number) of redundant movements after a cNAMP termination from a network of q subnetworks. The results show that a difference in the number of cNAMPs needs to be at least two before a movement will occur between locations of a same subnetwork. A system with q subnetworks makes at most $q-2$ redundant movements after a cNAMP termination from optimally balanced network with median probability less than 1% (Lemma 9). The number of movements after a cNAMP termination in a network of q subnetworks does not exceed $q - 1$ (Section 5.2).

Chapters 4 and 5 provide the first substantial investigation of threshing, or greedy effects, in distributed collections of autonomous mobile agents. Chapter 6 adapts the cNAMP cost model and investigates alternatives of cNAMP design in multilevel networks.

Chapter 6

Multilevel Network Design

Although AMPs were initially proposed as a load management tool for large and dynamic networks, so far they have been investigated only on Local Area Networks (LANs). This chapter investigates cNAMP design options in large networks and proposes a cNAMP design for large multilevel networks. For instance, many LANs connected into a Campus Area Network (CAN), in turn many CANs connected into a Metropolitan Area Network (MAN). Thus, locations, LANs, CANs, MANs form network levels. A simulation allows us to emulate large networks with an arbitrary number of levels. It also allows us to abstract from both network topology and computer configuration, e.g. single processor computers or multiprocessor computers.

The chapter starts with investigation of alternative network topologies (Section 6.1) and calculation of transfer delays (Section 6.2). Then alternative multilevel network architectures are considered and the chosen design justified. The design decisions aim to support an arbitrary number of network levels (Section 6.3). The chapter concludes with the justification of simulation parameters (Section 6.4) and summary (Section 6.5).

6.1 Topology

The first key design decision is how to model the network's topology. This section investigates possible schemes for simulating large networks. A number of simulation approaches can be used depending on the properties and parameters that the simulated network aims to investigate. Some of the approaches are as follows:

1. *Pure random* strategy is based on a probability p that two nodes are connected [ZCB96]. This strategy is typically used to study networking problems where the structure of the internetwork is not crucial.

2. *Hierarchical tree structure* [KK80]. Network nodes are clustered following hierarchical tree structure. Nodes are gathered in subnetworks which in turn are connected to each other by means of gateways. The *gateways* (or routers) are nodes that are connected to two or more networks; they may also be used to execute programs and tasks and to collect state information [MP95].

3. *The Waxman algorithm* creates a network on the basis of a fully connected graph, and then decides whether a link should exist using a probability function [Wax91]. The probability depends on the number of parameters, such as a distance between two nodes, the maximum distance between nodes, the total expected number of edges, ratio of long and short links. One of the drawback of Waxman algorithm is that it does not guarantee a connected network. Some of the algorithm alternative implementations are suboptimal resilient trees [DL93] and group Takahashi and Matsuyama algorithm [LW00].

4. *Explicitly hierarchical modelling approach* [CDZ97]. A network is constructed on the basis of a three level hierarchy: transit domains, stub domains, and LANs. The approach uses two sets of parameters to control properties of a generated network. The first set controls relative sizes of hierarchy levels, and the second set controls connectivity. The variants of the approach include Transit-Stub model [ZCB96] and Tiers [Doa96].

5. *Power-law strategy* is based on three power-laws of Internet topology discussed in [FFF99]. The authors use an analogy with communication networks where power-laws are used to describe traffic. The power-low strategy is used in such topologies generators as a random graph model [ACL00] and an incremental topology generator [BT02].

The current research implements a multilevel hierarchical network with a tree structure (Figure 6.1). This approach is chosen because the research aims to investigate cNAMP properties and distribution, and not a particular network or step-by-step data transfer between nodes. Following the discussion in Section 2.1.2 the children of a parent node are interconnected and division into levels is logical, i.e. a link between two nodes only indicates the possibility to move data from one node to another. To allow scalability and to be consistent with the research presented in the previous chapters, the locations form $Level\ 0$. Therefore, the number of a level is determined by children nodes taking into account sibling nodes (i.e. using the longest path from locations up to the root). Figure 6.2 shows an example of a three-level network, and Figure 6.3 shows its simulated prototype. A simulated network can have directly connected nodes which in the real network are connected via other nodes. These possible intermediate nodes are taken into account in calculation of a transfer delay (Section 6.2). Thus, a real network can be either a balanced tree or an unbalanced tree whereas its simulated prototype is always a balanced tree.

A choice between a fully and partially connected representation of networks is always a trade-off between simulating a general case or a particular network. Both networks in some way will carry properties that are more typical for one type of network than another. A fully connected network representation is avoided when a research investigate one of the following topics: workload of communication lines, complex network topologies with multiple network links between any two resources [Cas01], routing algorithms [PDR94], and topology sensitive algorithms e.g. information exchange occurs only between directly connected devices [NLJU02]. However, as this thesis does not

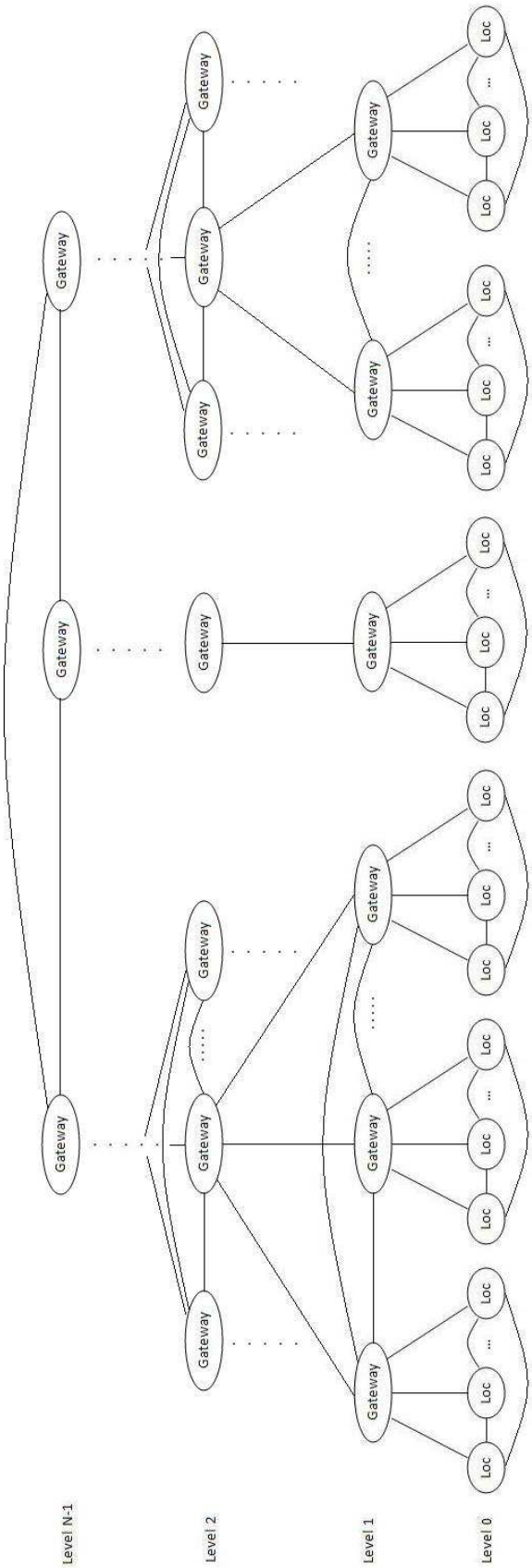


Figure 6.1: Hierarchical Tree Architecture

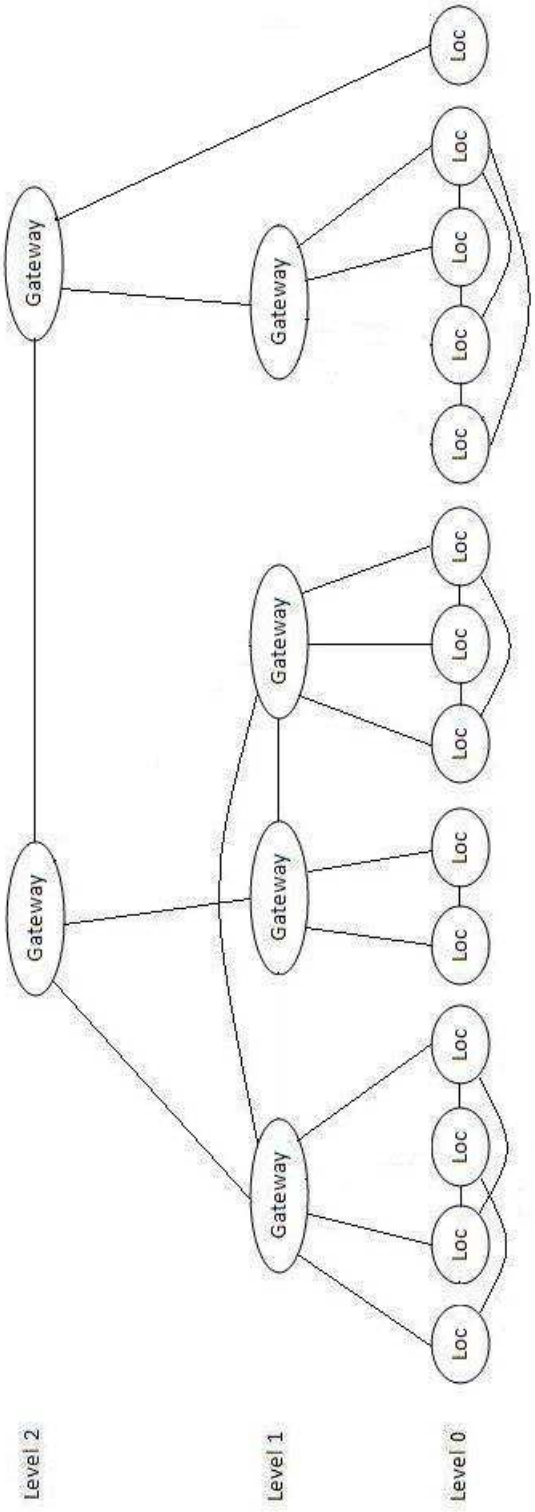


Figure 6.2: A Specific Hierarchical Tree Architecture (HA1)

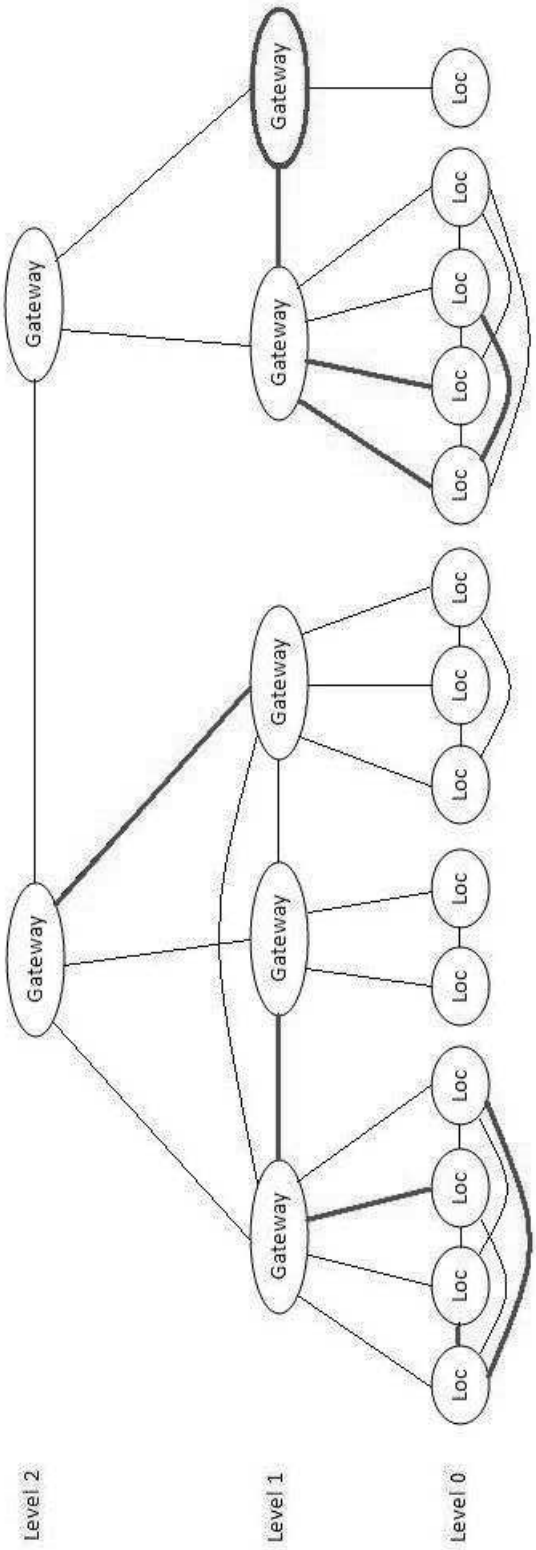


Figure 6.3: A Simulated HA1 Architecture

aim to investigate properties of a particular network, and none of the above research topics is in the scope of the current research a fully connected network representation is considered to be the most suitable choice.

In general a multilevel hierarchical architecture allows the simulation of an arbitrary number of levels. The analysis presented in Chapter 7 examines networks with up to five level. The zero level is formed by sets of locations, and the other levels are formed by gateways. The gateway functions in the real systems can be implemented by one of the locations. To analyse gateway features the functions of gateways and locations are separated. Thus, the total transfer delay, T_{tr}, from one location to another is calculated as follows:

$$T_{tr} = \sum_{i=0}^{N_{PG-1}} (2T_{tr,i}) - T_{tr,N_{PG}-1} \tag{6.1}$$

where N_{PG} is the number of the level of the nearest common parental gateway for both the initial and the target locations. Calculation of transfer delay depending on level i, $T_{tr,i}$, is discussed in Section 6.2.

6.2 Transfer Delay

A transfer delay can be simulated in a number of ways, depending on the parameters that are considered to have the most significant impact. For example, [AD96] discusses traffic simulation on a packet level, [TMR97] analyses traffic characteristics in terms of packet sizes, flow duration, [PV03] investigates relationship between round-trip-time and the geographical distance, [ZCBD04] examines delays across autonomous system links using empirical evaluation, [PZMH07] analyses the impact of network topologies and routing changes on delay-related metrics.

For the current research the widely cited work presented in [BMH+02, HM01] is found to be the most suitable. The main advantages of the chosen algorithm for the current research are as follows: a relative simplicity of the calculation, a small number of required

parameters that can be relatively easy be obtained for the simulation, and possibility to calculate a transfer delay of an arbitrary size program. The delay calculation is based on two parts: deterministic [BMH+02] and stochastic [HM01]. The calculations were made on the basis of measurements conducted in 1998–2001 by RIPE Network Coordination Centre [GGK+01, rip11]. To simplify the presentation the calculations provided below combine both deterministic and stochastic parts.

The following notions are used to support the discussion. A *hop* is a pausing in each node until sufficient resources are available for the further data transfer [oxf08]. A *distance* is a physical distance between two nodes (i.e. locations, gateways) [HM01]. A *router* is a node that only forwards messages from one network to another, and a *gateway* is a node that in addition to acting as a router also collects state information. The measurements presented further in this section are taken from [BMH+02, HM01] if otherwise is not stated.

The time to transfer a cNAMP to a target location, $T_{tr,i}$, is calculated as the sum of one packet transfer delay, D_p, the transmission delay to push all packets into the wire, D_T, the delay between dispatching the last and the first bits of two sequential packets, T_{gap}, and the increase in T_{gap} after a packet propagation over a number of hops, T_{sk}. Thus, cNAMP transfer time, $T_{tr,i}$, is

$$T_{tr,i} = D_p + D_T + T_{gap} + T_{sk}. \tag{6.2}$$

The measurements and calculations in [BMH+02] are made for 100 byte packets. However, measurements in [FML+03] show that 1500 byte packets are one of the most common default data transmission units. Therefore, to calculate a deterministic delay of a 1500 byte packet, D_p, the deterministic delay of 100 byte packet from [BMH+02] is multiplied by 15, i.e.

$$D_p = 15(D_l + D_{tt} + D_{ew}). \tag{6.3}$$

The meaning and the values of the variables in (6.3) are as follows:

- D_l is the total processing- and transmission delay in the intermediate routers that depends on the number of hops, h, and per-router latency, i.e. $D_l = 224h \cdot 10^{-6}$ sec.

- D_{tt} is the processing delay and transmission delay at the end-points. The *processing delay* is the time required to process a packet at a node and prepare it for transmission. The *transmission delay* is the time required to transmit a whole packet from the first to the last bit over a communication link. The *end-point* is a destination node. The value of D_{tt} was determined from direct link measurements in [BMH+02] and is equal to $155 \cdot 10^{-6}$ sec.

- D_{ew} is the propagation delay, i.e. the time required to transmit a bit through a communication link. It is directly proportional to a distance in kilometres, L, that data pass to get to the destination, i.e. $D_{ew} = 5L \cdot 10^{-6}$ sec. The measurements and discussion in [BMH+02] show that although distance has much smaller effect on delay than number of hops it still should be taken into account.

Therefore, (6.3) can be written as follows:

$$D_p = (224h + 5L + 155) \cdot 15 \cdot 10^{-6}. \tag{6.4}$$

The analysis of the distance, L, and the number of routers, h, in [BMH+02, Table 4] does not show explicit dependence between the parameters. Therefore, to calculate D_p the distance ranges are correlated with the hop ranges in Table 6.1. Column *Total Distance between Locations* shows the total possible distance between locations when there are gateways of the corresponding level between the two.

The time required to push all packets into the wire, D_T, is defined as follows [BMH+02]:

$$D_T = \frac{X \cdot N_p}{R_{tr}} \tag{6.5}$$

where X is a size of a packet (i.e. 1500 bytes), R_{tr} is a transmission rate, N_p is the number of packets in a program of size Z bytes, i.e.

$$N_p = \frac{Z}{X}. \tag{6.6}$$

Level	Distance between nodes, L (km)	Number of Routers h	Total Distance between locations (km)
0	1	1	1
1	1 – 15	1 – 2	3 – 17
2	10 – 65	1 – 2	14 – 96
3	55 – 200	1 – 2	79 - 362
4	160 – 440	1 – 2	294 - 1002

Table 6.1: Number of Levels vs. Distance and Hops

Substituting (6.6) in (6.5) gives the following:

$$D_T = \frac{Z}{R_{tr}}. \tag{6.7}$$

In the experiments presented in Chapter 7 two types of transmission rate are used: 12.5 MBps is assigned to the links of $Level\ 0$, and 4MBps is assigned to the links of $Level\ 1-Level\ 4$. These values agree with the measurements presented in [CK06].

The total delay between dispatching the last and the first bits of two sequential packets, T_{gap}, is calculated by multiplying the delay between dispatching the last and the first bits of two sequential packets, $T_{gap,1}$, by the number of packets, N_p, i.e.

$$T_{gap} = T_{gap,1} \cdot N_p. \tag{6.8}$$

Substituting (6.6) in (6.8) gives:

$$T_{gap} = \frac{T_{gap,1} \cdot Z}{X}. \tag{6.9}$$

The measurements and discussions presented in a number of sources, such as [NSS+04, SBK04], lead to the conclusion that the following approximation can be used: $T_{gap,1} \approx R_{tr} \cdot 10^{-13}$ sec.

Stochastic delay is the sum of queuing delays on the intermediate and end nodes [LGY09]. To calculate stochastic delay, T_{sk}, for the complete program the stochastic delay of one

packet presented in [HM01] is multiplied by the number of packets N_p, i.e.

$$T_{sk} = N_p \cdot (\sum_{j=1}^{h} D_{rp;\,j} + D_T) \tag{6.10}$$

that can be written as

$$T_{sk} = \frac{Z \cdot (\sum_{j=1}^{h} D_{rp;\,j} + D_T)}{X} \tag{6.11}$$

where the meaning and the values of $D_{rp;j}$ and D_T are as follows:

- $D_{rp;j}$ is a router processing delay approximated by Gaussian density with the mean given by $530 \cdot 10^{-6}$ sec and standard deviation given by $78 \cdot 10^{-6}$ sec;
- D_T is approximated by exponential distribution with mean given by $139 \cdot 10^{-6}$ sec.

Substituting (6.4), (6.7), (6.9) and (6.11) in (6.2) gives the following program transfer delay:

$$T_{tr,i} = (224h + 5L + 155) \cdot 15 \cdot 10^{-6} + Z \left(\frac{1}{R_{tr}} + \frac{R_{tr}}{X \cdot 10^{13}} + \frac{\sum_{j=1}^{h} D_{rp;\,j} + D_T}{X} \right). \tag{6.12}$$

Before the simulation starts, each node defines the distance and the number of hops to all neighbour nodes using uniform distribution. Nodes of one level treat a gateway to the upper level as another node of the same level.

6.3 cNAMP Design on Multilevel Networks

The section considers possible cNAMP designs on multilevel networks. First, the alternative implementations of multilevel networks, gateways, load servers, cNAMPs, and auxiliary messages are presented in Section 6.3.1. The provided alternatives are based on cNAMPs discussed in Chapter 4. Then the functioning of each device according to the chosen design is discussed in Section 6.3.2.

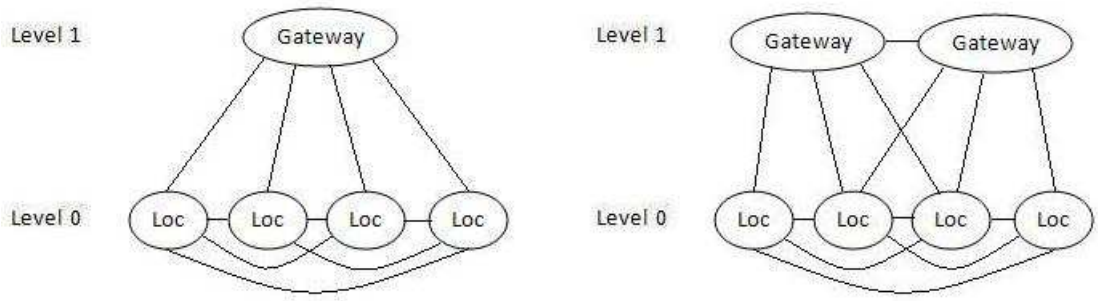
6.3.1 Design Alternatives

The design issues and alternatives of cNAMP implementation on multilevel networks are presented in Table 6.2. To clarify the design alternatives the questions are divided into the following three groups:

- *Multilevel Network* group investigates design decisions concerning a multilevel network architecture in general. Thus, to the nearest upper level nodes can have either a single parental gateway like in Totem multilevel communication system [MMSA+96] or multiple parental gateways like in Fremont multilevel system [WCS93]. The examples of single and multiple parental gateways are presented in Figure 6.4. Another design decision is a choice between weak and strong mobility. The type of mobility for one-level networks was discussed in Section 2.2, and is considered again for multilevel networks in Section 6.4.

- *Gateway and Location* group analyses alternatives concerning gateways and locations, i.e. gateway functions, types of information and rules the nodes follow to provide and collect this information. Recall that $Level\ 0$ is formed by the locations, whereas $Level\ 1$ and above are formed by gateways.

- *cNAMP and Request/Response Message* group examines alternatives of cNAMP and auxiliary message functioning. Auxiliary messages include state informa-



(a) Single Parental Gateway (b) Multiple Parental Gateways

Figure 6.4: Number of Parental Gateways

No.	Parameters	Alternative 1	Alternative 2	Alternative 3
Multilevel Network				
1	Number of parental gateways	One	Multiple	
2	Type of mobility	Strong	Weak	
Gateways and Locations				
2	Gateway functions	Collecting information	Executing cNAMPs and collecting information	
3	Type of information a location provides to the gateway	Available speed, committed load, latency of a state message	Expected relative speed, latency of a state message	Total relative speed, total load, latency of a state message
4	Type of information a gateway provides to other gateways	Maximum available speed, committed load, latency of a state message	Maximum expected relative speed, latency of a state message	Total relative speed, total load, latency of a state message
5	A location collects information about	all sibling locations and the parental gateway	all sibling locations and all parental locations up to the root	all sibling locations, all gateways of $Level$ 1, and the parental gateway of $Level$ 2
6	A gateway collects information about	child nodes of the nearest lower level, sibling gateways of the same level, and the parental gateway	child nodes of the nearest lower level, sibling gateways of the same level, and all parental gateways up to the root	
7	A gateway provides information about	one node	multiple nodes	
8	A gateway chooses information to pass on the basis of	maximum expected relative speed	maximum relative speed	minimum number of cNAMPs

Table 6.2: Multilevel AMP Architecture Design Alternatives

No.	Parameters	Alternative 1	Alternative 2	Alternative 3
cNAMPs and Request/Response Messages				
9	A cNAMP checks possibility to move to a LAN	every time the cNAMP recalculates parameters	only if there is no opportunity to reduce completion time locally	according to a timer, i.e. only after a certain period
10	If a cNAMP awaits a response from another LAN then other cNAMPs from the same location	may consider movements to other locations	may NOT consider movements to other locations	may recalculate parameters if the number of requests is less than a certain value
11	A request moves between levels	In any direction	According to some rule	
12	A request makes	any number of movements	a limited number of movements	a limited number of interlevel movements
13	A request actions after refusal	The response goes back to the initial location	The request goes one level back and tries again	

Table 6.2: Multilevel AMP Architecture Design Alternatives (continued)

tion and request/response messages. The cNAMP design alternatives cover the conditions of when cNAMPs are allowed to recalculate parameters to search an opportunity to move to another location. Request/response design alternatives include restrictions on maximum number of request movements, allowance of request movements between levels, and response actions after making a negative decision.

6.3.2 Implemented Design

The chosen cNAMP design on multilevel networks corresponds to alternative 1 in Table 6.2. The scheme is a so called *fusion scheme* due to the procedure the nodes use to collect and pass information [Hal92, Kle93], i.e. the nodes collect information from one group of

neighbour nodes, summarise and modify this information, and pass it to another group of nodes. To simplify the model the nodes in the simulated networks may have only one parental network. cNAMPs on multilevel networks are designed for strong mobility because in weak mobility the procedure of constant data transfer from the root location to the migrated cNAMPs is time consuming.

The components of cNAMP implementation on multilevel networks discussed in this section are as follows: cNAMPs, load servers, gateways, auxiliary messages. The choice of alternatives from Table 6.2 is largely influenced by cNAMP design on one-level networks to allow easy network scalability over unlimited number of levels. Therefore, the components have either the same or extended functions in comparison to one-level networks. The minor differences are due to the presence of gateways, and are discussed below. In general, the particular options were chosen due to the following three considerations: a) *scalability*, i.e. independently of the level gateways have the same functions which are very similar to a location load server, b) *elimination of the greedy effect* by taking into account investigation conducted in Chapter 4, and c) *simplicity* of the initial implementation to estimate system performance.

cNAMPs and Load Servers. The implementation of load servers and cNAMPs in multilevel networks is almost the same as in one-level networks with only minor additions. Pseudocode for cNAMPs and load servers is presented in Figures 6.5 and 6.6 respectively. Figure 6.5 should be compared with Figure 4.8, and Figure 6.6 should be compared with Figure 4.9. In multilevel networks cNAMPs can recalculate parameters using data from the gateway at any time. This is done to allow cNAMPs to distribute more quickly between remote locations. A load server does not lock gateway information when a cNAMP from the same location sends a request to the gateway. However, to prevent excessive requests the load server increases the committed load at the gateway. In terms of providing information a location treats the gateway as another sibling location and provides the gateway the same information as it provides to other locations, i.e. available speed, committed load, and time to transfer a state message.

Gateways. The gateway structure and functions are very similar to load servers. In general a gateway can combine functions of collecting information and executing cNAMPs. In the chosen alternative a gateway only collects information. In the future the gateway functionality can be extended by allowing gateways to execute cNAMPs. Gateways collect information using two lists of nodes:

- *Same level list*, contains information about sibling gateways and the parental gateway.

```
while work remains to execute
   {
    if outstanding request & positive response
       {
        inform local load server about movement
        move to target location or gateway
       }
    else if no outstanding request
       {
        find T_remote of T_gateway + T_comm_gateway
        if no cNAMP awaits a local response on the current location
            for n from 1 to total number of locations
               find minimum of T_n + T_comm
        if T_h > minimum | T_h > T_remote
           {
            if minimum > T_remote
                send request to Gateway
            else
                send request to Location_n
            inform local load server about request sent
           }
       }
    continue execution
    }
```

Figure 6.5: Multilevel Network cNAMP Algorithm

```
forever do
case local cNAMP sent a request:
    if to local Location_i
        lock information about Location_i
    else
        increase Gateway committed load
case local cNAMP received response:
   {
    renew information
    if from Location_i
        unlock information about Location_i
    if positive response
       reduce actual and committed loads
   }
case arrival notification from remote cNAMP:
    increase committed load
case cNAMP arrived:
   increase actual load
```

Figure 6.6: Multilevel Network Load Server Algorithm

- *Lower level list* contains information about nodes at the lower level.

The process of collecting and distributing information is as follows: a gateway collects information from nodes of one list, chooses a node with the highest relative speed, and passes the node information to the nodes of the other list. The information contains location available speed, committed load, and state message latency. The only fragment of state information that requires modification after arriving at another node is the state message latency. The latency accumulates the transfer delay, and is used to calculate cNAMP transfer delay. A gateway provides information about one location that has the maximum expected relative speed. Gateway pseudocode is presented in Figure 6.7.

Auxiliary Messages. cNAMPs are supported by two types of auxiliary messages: request/response and state information. In the designed version all auxiliary messages

```
forever do
case request arrived:
    renew information about visited and target nodes
case cNAMP arrived:
   renew information about visited node
```

Figure 6.7: Multilevel Network Gateway Algorithm

also transfer state information from node to node. This is done to faster renew state information and to better utilise transferring messages.

Request/response messages in multilevel networks have the same functions as in one-level networks, i.e. representing a cNAMP while the cNAMP continues execution on its location. As gateways provide only summary of state information, requests move from node to node recalculating state information each time until the request either denies or confirms the cNAMP transfer. The transfer can be denied at any node, whereas acceptance can be made only at a location. In the current implementation a request is not restricted in either the movement directions or the number of movements.

A request contains information about the remaining work of the cNAMP and the relative speed of the initial location. On the way to a better location a cNAMP accumulates the return request transfer delay, and the one-way cNAMP transfer delay. On the way back to the initial location the response message carries the transferring decision and state information that is renewed at every node. After arrival at the initial location the response message informs the load server about the cNAMP transferring decision, provides state information of the last visited node, and terminates.

State messages are generated on each node using two lists: the same level list and the lower level list. For the lower level list, the information is accumulated from the nodes of the same level list, and for the same level list the information is accumulated from the nodes of the lower level list. The information to the nodes of these two lists is sent independently from each other at an interval. The interval between state message generation is chosen to be equal to the mean time required to transfer state information to a node

from the list. The state information contains maximum available speed, committed load, and state message latency of a location that has the maximum expected relative speed in the second list. After arrival to the target location a state message increases information about state message latency by the time of the message transferring, then it provides state information, and terminates.

6.4 Simulation Parameters

The section presents simulation parameters and their justifications. The simulation does not aim to validate or reproduce any particular real network. However, the parameters used in the simulation are selected to reflect real networks. For example, interconnected LANs may form a Campus Area Network (CAN), interconnected CANs within Scotland may form a Scotland Area Network (SAN).

Previously simulated AMPs and cNAMPs were implemented using measurements made for AMPs with weak mobility. However, weak mobility does not preserve execution states, and programs must restart execution after a movement, whereas strong mobility allows program execution to continue on a remote location from the point of interruption (Section 2.2). Although many programs can be implemented using weak mobility, strong mobility covers a larger spectrum of programs. Therefore, the simulation in Chapter 7 is conducted for strong mobility.

From the discussion in Section 3.1 the main measurements in [Den07] were made for weak mobility using Java Voyager. One of the Java Voyager features is that the language does not extend Java. This feature was taken into account in choosing JavaGoX [SSY00] among mobile languages that support strong mobility.

For the current simulation the change from weak to strong mobility is mainly associated with executing speed and size of transferring data. As Java Voyager does not save execution state it is assumed that Java Voyager has the same execution speed and program

size as Java. Following [IKKW02] JavaGoX has approximately an 100% increase of bytecode and approximately 50% execution time overhead in comparison with Java due to necessity to save execution state.

Recall that in the current simulation execution speed is calculated in units per second. The functionality of a unit depends on the type of program. For example, one unit in matrix multiplication programs consists of three operations, i.e. one operation of multiplication, one operation of addition and one operation of assigning (Section 2.4.1). Using measurements presented in [Den07] for weak and strong mobility and discussion in [IKKW02] it is assumed that the considered programs spend more time on saving current state using strong mobility than on data transfer using weak mobility. Therefore, to convert execution speed from weak to strong mobility the weak mobility execution speed is decreased by 33%.

As [Den07] does not provide direct dependence between program size, Z, and program dimension, d, the calculations presented below are used. Size of program Z in matrix multiplication AMPs, for example, includes matrix multiplication program code and two matrices B and C where $A \times B = C$. Thus, rows of matrix A are periodically transferred from the initial location whereas matrix B and resulting matrix C always move with the program code. The calculations are made for each program separately, i.e. matrix multiplication, coin counting, and ray tracing. Here, only the calculation of the matrix multiplication parameters are presented. Parameters for coin counting and ray tracing programs are calculated using the same mechanism.

The calculation is made on the assumption that doubling the data size doubles the transfer time on a LAN. Thus, equating (6.12) and double (3.1) gives

$$(224h + 5L + 155) \cdot 15 \cdot 10^{-6} + Z\left(\frac{1}{R} + \frac{R}{X \cdot 10^{13}} + \frac{\sum_{j=1}^{h} D_{rp;\,j} + D_T}{X}\right) = \\ = 2 \cdot (0.029 + 5.07 \cdot 10^{-6} \cdot d^2). \tag{6.13}$$

Substituting values from Section 6.2 for a LAN and taking $h = 1$, $L = 1$ gives the

Type of Programs	T_{com} on LAN, sec	W, units	Z, bytes	Speed	
				S_W, units per sec	S_{CPU}, MHz
Coin Counting	0.074	d	129529	363	3193
Ray Tracing	$0.035 + 3.97 \cdot 10^{-5} \cdot d^2$	d^2	$55501 + 75.35d^2$	53.5	
Matrix Multiplication	$0.029 + 5.07 \cdot 10^{-6} \cdot d^2$	d^3	$99158 + 19.24d^2$	14243401	

Table 6.3: Simulation Parameters

following dependence between program size, Z, and matrix dimension, d:

$$Z = 99158 + 19.24d^2. \tag{6.14}$$

The summary of simulation parameters for programs of coin counting, ray tracing, and matrix multiplication are presented in Table 6.3. For each type of program the table provides the following information using data from [Den07]: formulas on calculation communication time on LANs, the total program work, and correlation of execution and CPU speeds. Coin counting and ray tracing program sizes in bytes depending on dimension, d, are calculated using the same approach as for matrix multiplication program in (6.14).

As strong mobility eliminates the necessity to constantly send data from the root location to the transferred cNAMPs, all locations are treated equally, i.e. load factor of the root location $f = 1$.

6.5 Discussion

The chapter has examined network topologies and identified the reasons to use abstract multilevel networks to investigate cNAMP behaviour on large networks. The main reason multilevel networks are used to analyse cNAMPs on large networks is that this abstracts from both network topology and architecture. Thus, although child nodes of the same parental node are fully connected, this does not necessarily mean that a real

connection between the nodes exists. The connection only indicates the possibility of transferring data from one node to another; probable intermediate nodes and distance variability are taken into account in transfer delay (Sections 6.1 and 6.2).

The chapter provides design alternatives for cNAMPs on multilevel networks that consider such parameters as number of parental gateways, type of mobility, gateway functionality, type of node exchange information, conditions of movements to remote locations, and request movements. The discussions of cNAMP components together with pseudocode and justification of simulated parameters are provided for the implemented design (Section 6.3–6.4).

The effectiveness and redundant movements of the proposed design depending on different parameters are investigated in Chapter 7.

Chapter 7

Evaluation of Multilevel cNAMP Architecture

The chapter evaluates the cNAMP fusion scheme on multilevel networks presented in Chapter 6. The experiments analyse up to five level networks varying the number of locations from 5 to 336, and the number of cNAMPs from 8 to 3360. The majority of experiments is conducted on three-level networks of locations with the same available speed (3193 MHz). The experiments use matrix multiplication programs of size 5000×5000 that start on one location and then distribute themselves across the network. These parameters are used unless others are stated. To have a link between different types of experiments a number of them include the following scenario: a symmetric three-level network has 5 nodes of $Level\ 0$, four nodes of $Level\ 1$, and three nodes of $Level\ 2$ where each node of $Level\ 0$ has 3193 MHz available speed and 240 cNAMPs in total. To simplify the indication of the scenario it is marked with $(*)$ To simplify the notation a set of locations that have the same parental node of $Level\ i$ is called the $Level\ i$ group. cNAMP effectiveness is investigated in Section 7.1 and redundant movements are analysed in Section 7.2.

7.1 Effectiveness

The effectiveness of the proposed cNAMP scheme is investigated by conducting eight experiments where minimum, mean, and maximum completion time is analysed. Some experiments compare simulated completion time with the hypothetical completion time, T_{hyp}, which assumes fair round-robin scheduling [DKS89] without taking into account communication and coordination costs. The hypothetical time is calculated as a ratio of the total cNAMP work to the total speed of locations:

$$T_{hyp} = \frac{\sum_{i=1}^{N_{cNAMP}} W_i}{\sum_{j=1}^{N_{Loc}} S_j}.$$

Here, N_{cNAMP} is the total number of cNAMPs, N_{Loc} is the total number of locations, W_i is the total work of cNAMP i, S_j is available speed of location j. All cNAMPs simultaneously start on one location, distribute themselves in the network with zero communication delay, and therefore terminate also simultaneously. This allows us to compare the performance of the implemented cNAMP against an 'ideal' execution.

The experiments analyse cNAMP completion time depending on the number of levels (Section 7.1.1), the type of network topology (Section 7.1.2), the number of locations (Section 7.1.3), the number of cNAMPs (Section 7.1.4), the work of cNAMPs (Section 7.1.5), the type of cNAMPs (Section 7.1.6), the speed of locations (Section 7.1.7), and the type of rebalancing, i.e. initial distribution, rebalancing after adding and termination cNAMPs (Section 7.1.8).

7.1.1 Experiment A1: Number of Levels

This experiment investigates the effectiveness of cNAMP balancing depending on the number of network levels. The number of levels ranges from one to five. The total number of locations is chosen so that each location has four cNAMPs in an optimal balanced state. The experiment scenarios are presented in Table 7.1. The topologies are

Scenario No.	No. of Levels	Number of nodes on					Total number of	
		$Level\ 4$	$Level\ 3$	$Level\ 2$	$Level\ 1$	$Level\ 0$	Locations	cNAMPs
1.1	1	-	-	-	-	5	5	20
1.2	2	-	-	-	4	5	20	80
1.3 $(*)$	3	-	-	3	4	5	60	240
1.4	4	-	2	3	4	5	120	480
1.5	5	3	2	3	4	5	360	1440

Table 7.1: Experiment A1 Topologies

homogeneous in having the same order (number of nodes) at every level. The order of each level is different to avoid unexpected patterns emerging. The impact of topologies is discussed in Section 7.1.2.

The results in Figure 7.1 show that minimum, mean and maximum completion time for all scenarios is close to the hypothetical value. The reason the mean and the maximum values increase with more levels is because the cNAMPs have to move larger distances (Table 6.1 in Section 6.2), whereas the hypothetical values are calculated without considering any communication costs. Although a cNAMP can decrease its completion time by moving either a short or a long distance, the advantage gained by moving a large distance is smaller. Therefore, the larger distance a cNAMP transfers to the larger difference between experimental and hypothetical times. Table 7.2 shows the changes of minimum, mean, and maximum completion time as the number of levels increases. The table shows that the mean value increases by 1%–3% with every additional level, and the maximum completion time increases by 2%–5%.

The decrease in the minimum value as the number of levels increases is due to a difference in the recalculation method at $Level\ 0$. cNAMPs are always allowed to recalculate their parameters using information from the gateway but data from the local locations can only be used when no cNAMP on the same location awaits a response from the local locations. Thus, if there is a cNAMP which never moves from its initial location this cNAMP has no communication costs *and* has more time *for its own* execution which allows it to terminate earlier than the rest of cNAMPs.

Type of Time	Level change			
	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$	$4 \rightarrow 5$
Maximum	3%	4%	2%	5%
Mean	2%	1%	2%	3%
Minimum	-0.5%	-2%	-2%	1%

Table 7.2: Changes of Completion Time Depending on the Number of Levels (A1)

Additional experiments can be conducted in the future to investigate in details the dependence and the factors that impact on increase of completion time with the increase of the number of levels. For example, to clarify whether the difference increases with the scale of number of levels. However, as the mean completion time increases by just a few percent with each new level we conclude that *the multilevel architecture scales effectively*.

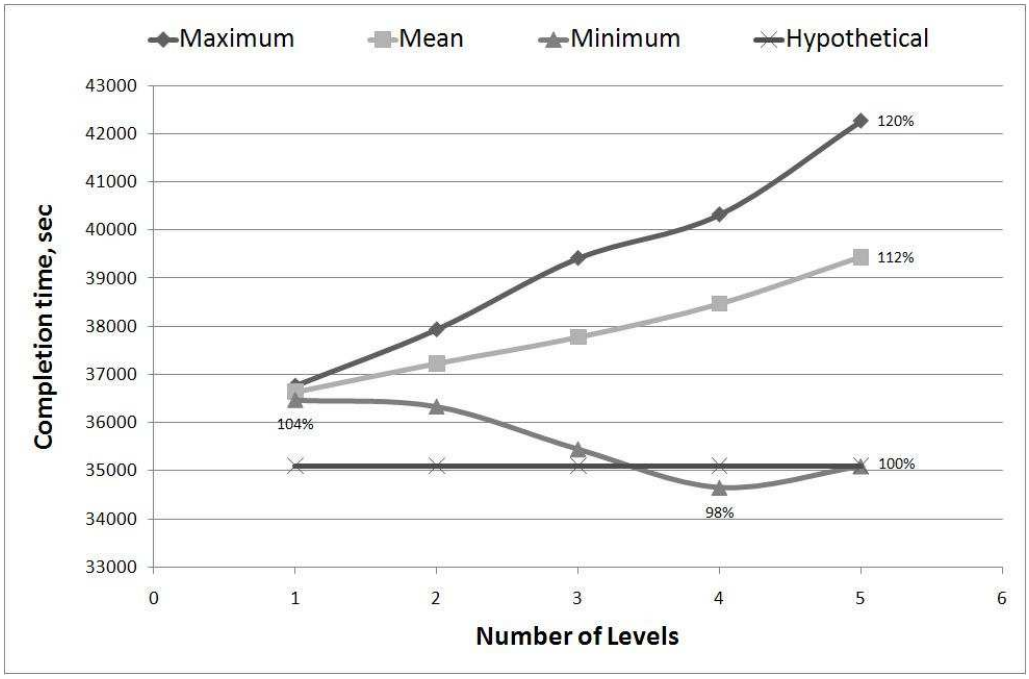


Figure 7.1: cNAMP Completion Time vs. Number of Levels

7.1.2 Experiment A2: Network Topology

This experiment analyses cNAMP effectiveness depending on the network topology, i.e. distribution and grouping of nodes. The experiment is conducted on three-level networks where the number of nodes of $Level\ 2$ ranges from two to five, the number of nodes of $Level\ 1$ ranges from two to eight, and the number of nodes of $Level\ 0$ ranges from two to sixteen. The total number of locations in each experiment is 60. The total number of cNAMPs is 240. The cNAMPs are programs of matrix multiplication of size 3000. Scenarios are presented in Table 7.3. Topology 2.2 is homogeneous (i.e. it has the same order at every level) and the remaining topologies are heterogeneous. The table shows the number of nodes for each level and the total number of locations. To optimally present data in the table the number of nodes in each $Level\ 1$ group of nodes presented in a row in column $Level\ 0$.

The results from Figure 7.2 lead to the conclusion that *the type of topology has no significant impact on cNAMP completion time* as long as the number of levels and the number

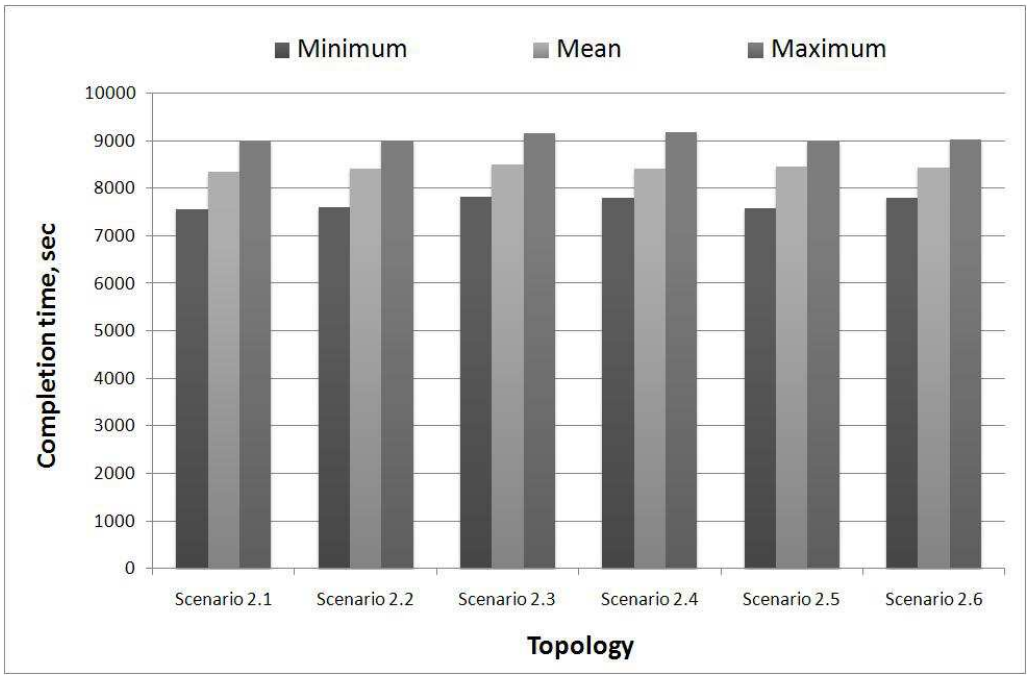


Figure 7.2: cNAMP Completion Time vs. Topology

No. of Scenario	Number of nodes on			Total No. of Locations
	$Level\ 2$	$Level\ 1$	$Level\ 0$	
2.1	2	8	8 – 3 – 5 – 5 – 2 – 4 – 7 – 6	40
		4	3–2–9–6	20
2.2 $(*)$	3	4	5 – 5 – 5 – 5	20
		4	5 – 5 – 5 – 5	20
		4	5 – 5 – 5 – 5	20
2.3	3	2	5 – 4	9
		6	7 – 9 – 2 – 5 – 3 – 2	28
		3	6 – 9 – 8	23
2.4	4	4	7 – 16 – 2 – 5	30
		2	7 – 2	9
		3	2 – 3 – 4	9
		3	4 – 5 – 3	12
2.5	4	3	4 – 7 – 3	14
		3	9 – 5 – 3	17
		2	3 – 6	9
		2	3 – 7	10
2.6	5	3	4 – 4 – 5	13
		2	3 – 7	10
		4	6 – 8 – 9 – 3	26
		2	4 – 3	7
		2	2 – 2	4

Table 7.3: Experiment A2 Topologies

of locations are constant, e.g. mean completion time varies by less than 2% across all topologies. The reason the cNAMP completion time is almost the same for different topologies is mainly due to the particular design of cNAMPs on multilevel networks (Section 6.3.2), i.e. cNAMPs that move within their $Level\ 1$ group tend to make more redundant movements than cNAMPs that move to remote locations. Analysis in Section 7.2.1 shows that a cNAMP that moves to a location of another $Level\ 3$ group tends to arrive directly to the location on which it will stay when the system enters a balanced state. Whereas, a cNAMP that moves within $Level\ 1$ group tends to make a number of

redundant movements before the system enters a balanced state. Thus, cNAMPs that move to remote locations spend approximately the same time as cNAMPs that relocate locally.

7.1.3 Experiment A3: Number of Locations

This experiment analyses cNAMP effectiveness depending on the total number of locations. The number of locations ranges from 8 to 336. The experiment is conducted on three-level networks. The total number of cNAMPs is chosen to allow each location to have four cNAMPs in optimal balanced state. The experiment scenarios are presented in Table 7.4.

Results in Figure 7.3 show that minimum, mean and maximum values of cNAMP completion time are very close to the hypothetical value, i.e. within 3%, 9%, and 13% respectively. This leads to the conclusion that *completion time shows no significant dependence on the number of locations*.

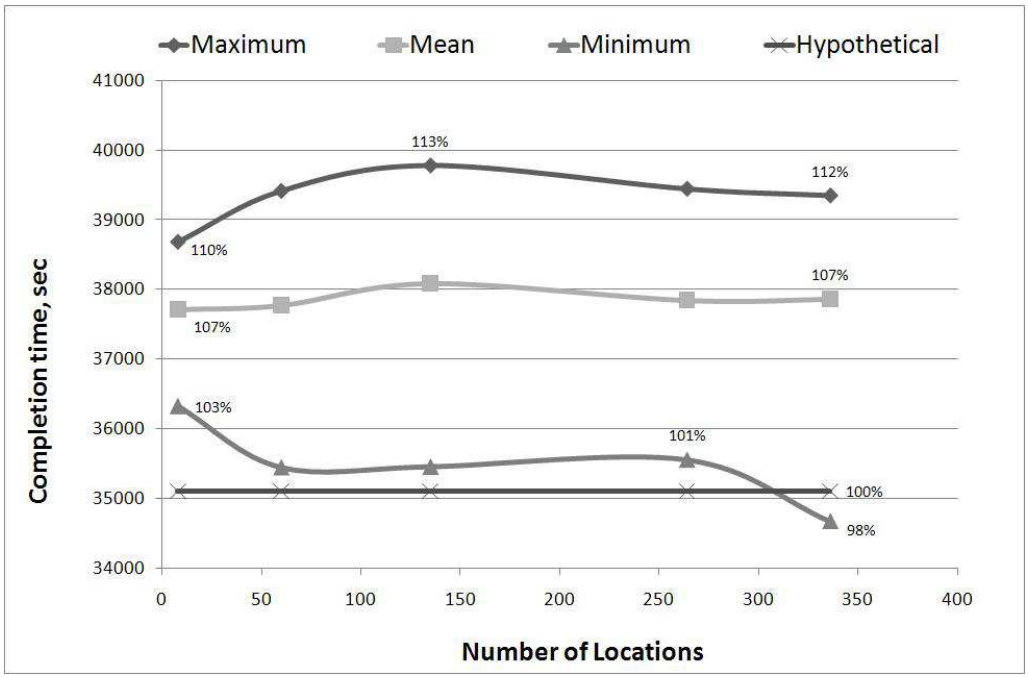


Figure 7.3: cNAMP Completion Time vs. Number of Locations

No. of Scenario	Total No. of Locs.	Total No. of cNAMPs	Number of Nodes		
			$Level\ 2$	$Level\ 1$	$Level\ 0$
3.1	8	32	2	2	2
3.2 $(*)$	60	240	3	4	5
3.3	135	540	5	3	9
3.4	264	1056	3	8	11
3.5	336	1344	4	12	7

Table 7.4: Experiment A3 Topology

7.1.4 Experiment A4: Number of cNAMPs

This experiment estimates cNAMP effectiveness depending on the total number of cNAMPs. The completion time of collection of 31, 60, 61, 200, 240, 420, 460, and 600 cNAMPs is analysed. The experiment is conducted on a three-level network. The number of nodes in the network is presented in Table 7.5.

Results in Figure 7.4 show that minimum, mean, and maximum values are very close to the hypothetical value. The measurements are presented in Table 7.6. The horizontal part between 31 and 60 cNAMPs corresponds to the integer design of cNAMPs, i.e. a cNAMP cannot be executed by more than one node. The near vertical growth in maximum completion time between 60 and 61 cNAMPs is also due to the integer design of cNAMPs, as in the experiment with 61 cNAMPs all locations have one cNAMP except one location that has two cNAMPs. These two cNAMPs have half of execution speed of other locations and therefore the maximum completion time increases.

From this we conclude that *cNAMP completion time directly relates to the number of*

Scenario	Number of nodes on			Total No. of Locations
	$Level\ 2$	$Level\ 1$	$Level\ 0$	
4.1 $(*)$	3	4	5 – 5 – 5 – 5	20
		4	5 – 5 – 5 – 5	20
		4	5 – 5 – 5 – 5	20

Table 7.5: Experiment A4 Topology

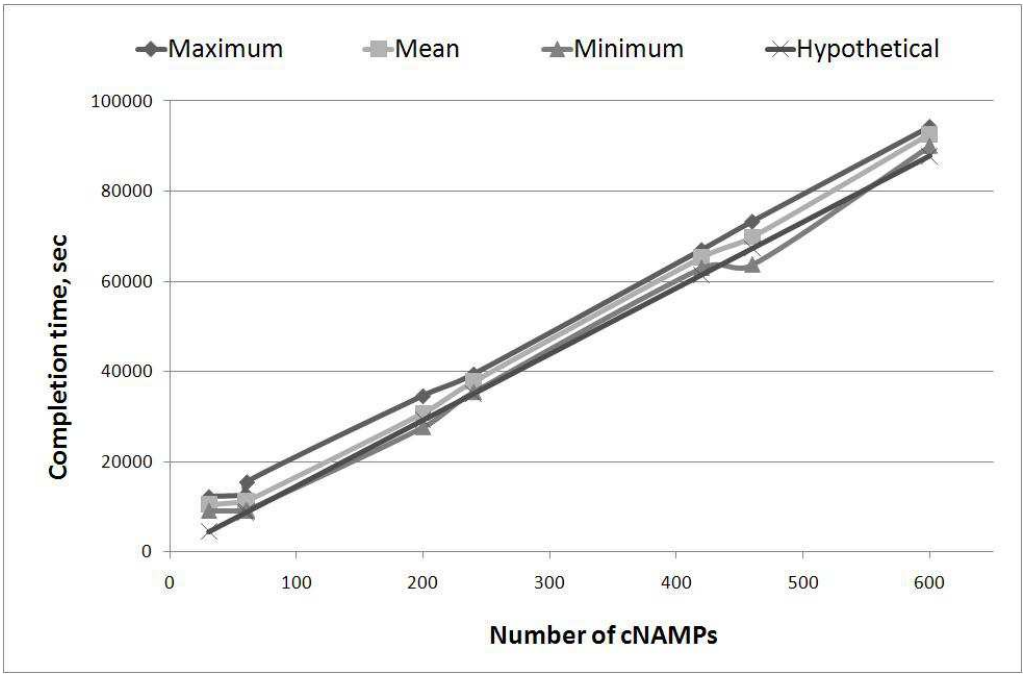


Figure 7.4: cNAMP Completion Time vs. Number of cNAMPs

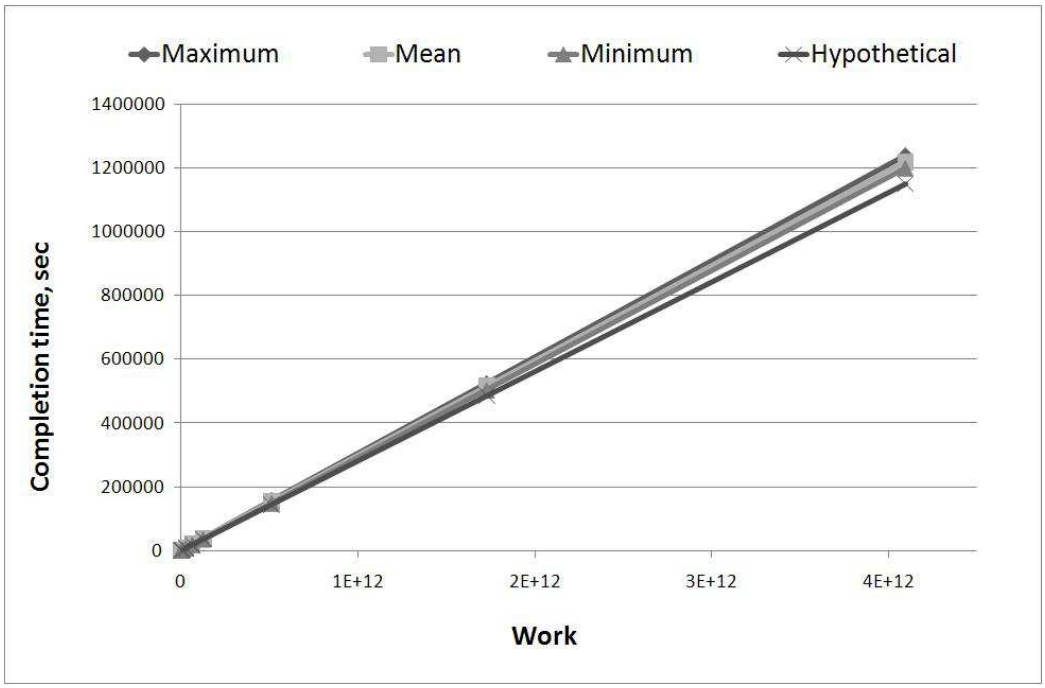
cNAMPs when the number of cNAMPs exceeds the number of locations.

7.1.5 *Experiment A5: Work of cNAMPs*

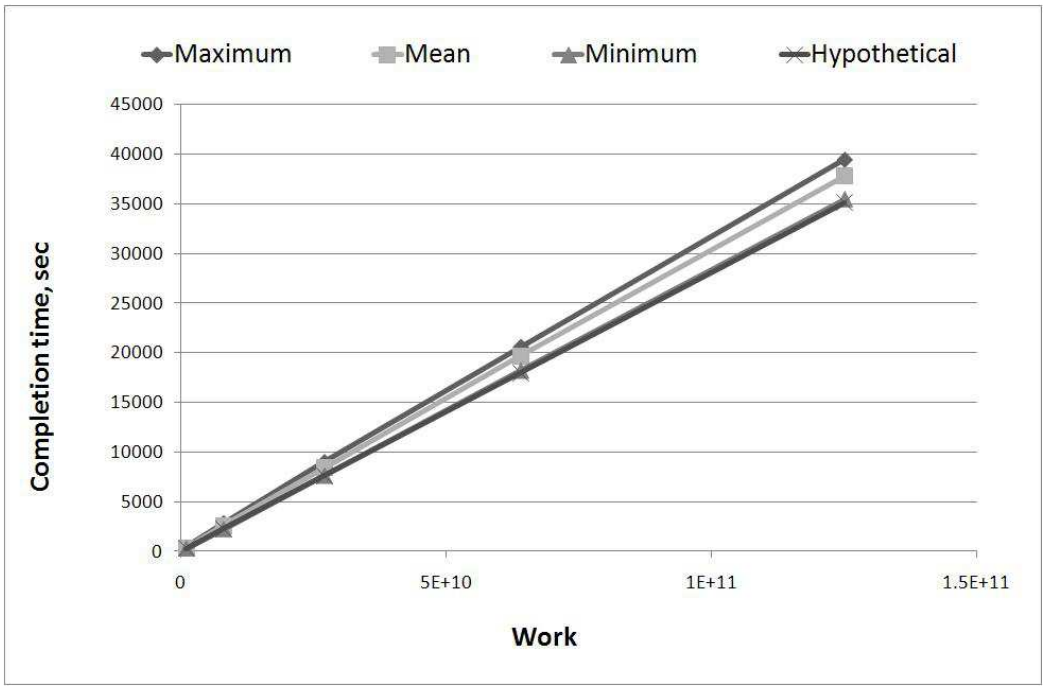
This experiment analyses cNAMP effectiveness depending on the work that cNAMPs need to execute. The experiment is conducted using 240 cNAMPs. The completion time of cNAMPs using matrices of the following dimensions are analysed: 1000, 2000, 3000, 5000, 8000, 12000, and 16000. Recall that total work, W, for matrix multiplication programs is $W \propto d^3$ where d is dimension of a square matrix (Section 2.4). The

Completion time	Number of cNAMPs							
(sec)	31	60	61	200	240	420	460	600
Minimum	9130	9131	9206	27669	35446	62972	63720	90010
Mean	10547	11196	11304	30814	37774	65256	69847	92623
Maximum	12195	12733	15487	34595	39412	67033	73325	94353

Table 7.6: cNAMP Completion Time in Experiment A4



(a) Full Results



(b) Scaled Results

Figure 7.5: cNAMP Completion Time vs. cNAMP Work

Completion time, sec	Total cNAMP Work							
	10^9	$8 \cdot 10^9$	$27 \cdot 10^9$	$64 \cdot 10^9$	$125 \cdot 10^9$	$512 \cdot 10^9$	$1728 \cdot 10^9$	$4096 \cdot 10^9$
Minimum	256	2206	7589	18179	35446	147898	503117	1200208
Mean	358	2570	8415	19678	37774	154003	515823	1219223
Maximum	429	2851	9009	20592	39412	158406	524674	1239061
Hypothetical	294	2354	7944	18830	36488	150660	508478	1205280

Table 7.7: cNAMP Completion Time in Experiment A5

experiments are conducted on a three-level network using topology from Table 7.5.

Figure 7.5 and experimental data in Table 7.7 show that minimum, mean, and maximum values are very close to the hypothetical value. The absolute difference between hypothetical and experimental values increases with the increase of the work. However, the relative difference presented in Table 7.8 decreases, e.g. discrepancy between hypothetical and measured values is within 10% when the total work exceeds $125 \cdot 10^9$ units. From the results we conclude that *cNAMP completion time is directly related to the cNAMP work.*

7.1.6 Experiment A6: Type of cNAMPs

This experiment analyses cNAMP effectiveness depending on the type of programs. The analysis is conducted on a three-level network using such programs as coin counting, ray tracing, and matrix multiplication. The main reason of choosing these programs is

Completion time, sec	Total cNAMP Work							
	10^9	$8 \cdot 10^9$	$27 \cdot 10^9$	$64 \cdot 10^9$	$125 \cdot 10^9$	$512 \cdot 10^9$	$1728 \cdot 10^9$	$4096 \cdot 10^9$
Minimum	-13%	-6%	-4%	-3%	-3%	-2%	-1%	-0.5%
Mean	21%	9%	6%	5%	4%	2%	1%	1%
Maximum	50%	21%	13%	9%	8%	5%	3%	3%

Table 7.8: Relative Difference of Experimental Completion Time from Hypothetical

No. of Scenario	Number of cNAMPs			Total No. of cNAMPs
	Coin Counting	Ray Tracing	Matrix Multiplication	
6.1	240	-	-	240
6.2	-	240	-	
6.3 $(*)$	-	-	240	
6.4	120	120	-	
6.5	120	-	120	
6.6	-	120	120	
6.7	80	80	80	

Table 7.9: Number and Types of cNAMPs in Experiment A6

to have diversity in communication time (Table 6.3). The network is configured on the basis of Table 7.5. The experiment analyses 240 cNAMPs. The type of programs in each scenario is presented in Table 7.9. The work for each type of program was chosen to allow approximately the same maximum cNAMP completion time in experiments 6.1-6.3. The rationale is to investigate discrepancy between minimum, mean, and maximum values of completion time for different types of cNAMPs.

The results in Figure 7.6 show that the smaller the cNAMP communication time the less the completion time discrepancy between maximum, mean, and minimum values, e.g. discrepancy between minimum and mean values are 0.2%, 4%, and 10% in scenarios 6.1, 6.2, and 6.3 respectively. At the same time the larger the variety of communication time the larger the difference between the minimum and the mean completion time. This is due to the hidden capacity phenomenon, i.e. while cNAMPs with long communication time transfer to the target location the cNAMPs which are already present on the location use idle resources. This phenomenon also reduces the mean completion time.

Type of Program	Size, d	Work, W	Mean communication latency within a LAN, sec
Coin Counting	$3 \cdot 10^6$	d	0.15
Ray Tracing	10^3	d^2	79
Matrix Multiplication	3200	d^3	104

Table 7.10: Program Types in Experiment A6

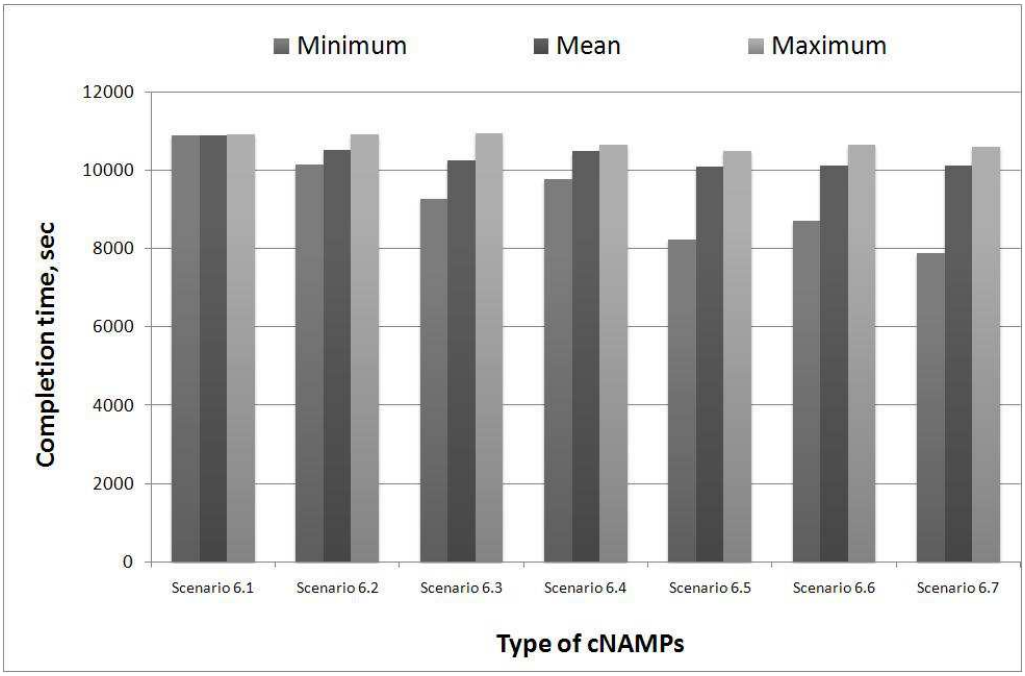


Figure 7.6: cNAMP Completion Time vs. Type of cNAMPs

The results lead to the conclusion that *the discrepancy between maximum, mean, and minimum values of completion time is directly proportional to the values and variety of communication time.*

7.1.7 Experiment A7: Speed of Locations

This experiment analyses cNAMP efficiency depending on speeds of locations. The experiments employ locations of three different available speeds: 3193 MHz, 2168MHz, and 1793 MHz. The analysis is conducted on three level networks where scenarios 2.2 and 2.3 from Table 7.3 are used to configure networks for scenarios 7.1–7.3 and 7.4–7.6 respectively. Thus, scenarios 7.1–7.3 have homogeneous topology, and scenarios 7.4–7.6 have heterogeneous topology. The total number of cNAMPs is chosen to allow cNAMPs to enter optimal balance after initial distribution. Following Section 5.1.6 the total number of cNAMPs is chosen to allow cNAMPs to enter the following optimally balanced states: four cNAMPs on locations with available speed 3193 MHz, three

No. of Scenario	Speed of Locations[1]	Total No. of Locations		
		Fast	Middle	Slow
7.1	$a^5 - a^5 - a^5 - a^5$	20	20	20
	$b^5 - b^5 - b^5 - b^5$			
	$c^5 - c^5 - c^5 - c^5$			
7.2	$a^5 - b^5 - b^5 - c^5$	15	30	15
	$a^5 - b^5 - b^5 - c^5$			
	$a^5 - b^5 - b^5 - c^5$			
7.3	$aabbc - aabbc - aabbc - aabbc$	24	24	12
	$aabbc - aabbc - aabbc - aabbc$			
	$aabbc - aabbc - aabbc - aabbc$			
7.4	$a^5 - a^4$	9	28	23
	$b^7 - b^9 - b^2 - b^5 - b^3 - b^2$			
	$c^6 - c^9 - c^8$			
7.5	$a^5 - b^4$	23	25	12
	$a^7 - b^9 - c^2 - a^5 - b^3 - c^2$			
	$a^6 - b^9 - c^8$			
7.6	$abcab - abca$	23	21	16
	$abcabca - abcabcabc - ab - abcab - abc - ab$			
	$abcabc - abcabcabc - abcabcab$			

[1] a – 3193 MHz, b – 2168 MHz, c – 1793 MHz.

Table 7.11: Speeds of Locations in Experiment A7

cNAMPs on locations with 2168 MHz, and one cNAMP on locations with 1793 MHz. Thus, after initial distribution cNAMPs have 785 MHz, 723 MHz, and 897 MHz relative speeds respectively. The cNAMPs are programs of matrix multiplication of size 3000. The total number of locations is sixty.

The comparative analysis of cNAMP experimental and hypothetical completion time in Figure 7.7 shows that cNAMPs effectively distribute themselves independently of location speeds and location settings. The mean completion times differ by at most 3%. Thus, we conclude that *mixture of speeds of locations has no sufficient impact on effectiveness of cNAMP distribution*.

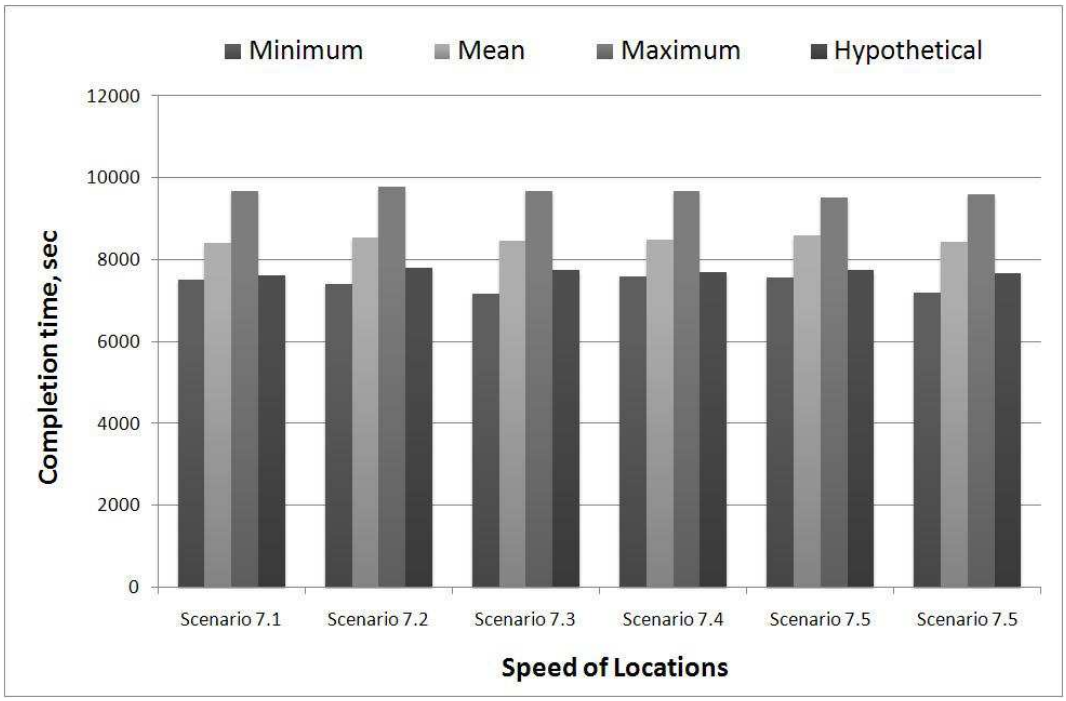


Figure 7.7: cNAMP Completion Time vs. Speed of Locations

7.1.8 Experiment A8: Rebalancing

This experiment analyses cNAMP efficiency depending on the type of rebalancing: initial distribution, rebalancing after adding more cNAMPs and rebalancing after cNAMPs terminate. The experiment is again conducted on a three-level network. Scenario classes 8.x.1 and 8.x.2 use topology presented in scenario 2.2 from Table 7.3 (homogeneous topology), scenario classes 8.x.3 and 8.x.4 use topology presented in scenario 2.3 from Table 7.3 (heterogeneous topology). The second digit in the number of scenario class defines the type of experiment, i.e. 1 is initial distribution, 2 is rebalancing after adding cNAMPs, and 3 is rebalancing after termination cNAMPs. The total number of locations is 60. The total number of cNAMPs is 240 which are programs of matrix multiplication of size 3000. Speeds of locations in scenario classes 8.x.1–8.x.4 are presented in Table 7.12.

Scenario class No.	Speed of Locations[1]	Total No. of Locations		
		Fast	Middle	Slow
8.x.1	$a^5 - a^5 - a^5 - a^5$	60	-	-
	$a^5 - a^5 - a^5 - a^5$			
	$a^5 - a^5 - a^5 - a^5$			
8.x.2	$aabbc - aabbc - aabbc - aabbc$	24	24	12
	$aabbc - aabbc - aabbc - aabbc$			
	$aabbc - aabbc - aabbc - aabbc$			
8.x.3	$a^5 - a^4$	60	-	-
	$a^7 - a^9 - a^2 - a^5 - a^3 - a^2$			
	$a^6 - a^9 - a^8$			
8.x.4	$abcab - abca$	23	21	16
	$abcabca - abcabcabc - ab - abcab - abc - ab$			
	$abcabc - abcabcabc - abcabcab$			

[1] a – 3193 MHz, b – 2168 MHz, c – 1793 MHz.

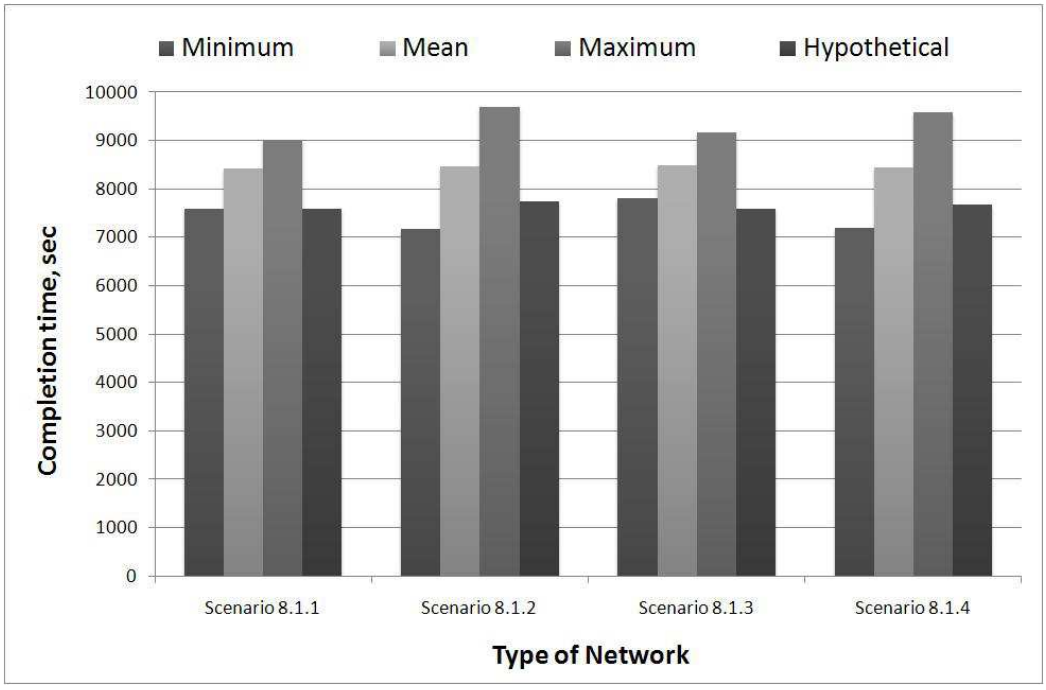
Table 7.12: Speed of Locations in Experiment A8

Initial Distribution. The effectiveness of initial distribution is analysed on optimal and near-optimal balancing. The total number of cNAMPs in each scenario class is presented in Table 7.13. The results in Figures 7.8(a) and 7.8(b) show that minimum, mean, and maximum values of completion time are very close to the hypothetical value, e.g. the mean value exceed hypothetical value by at most 12%.

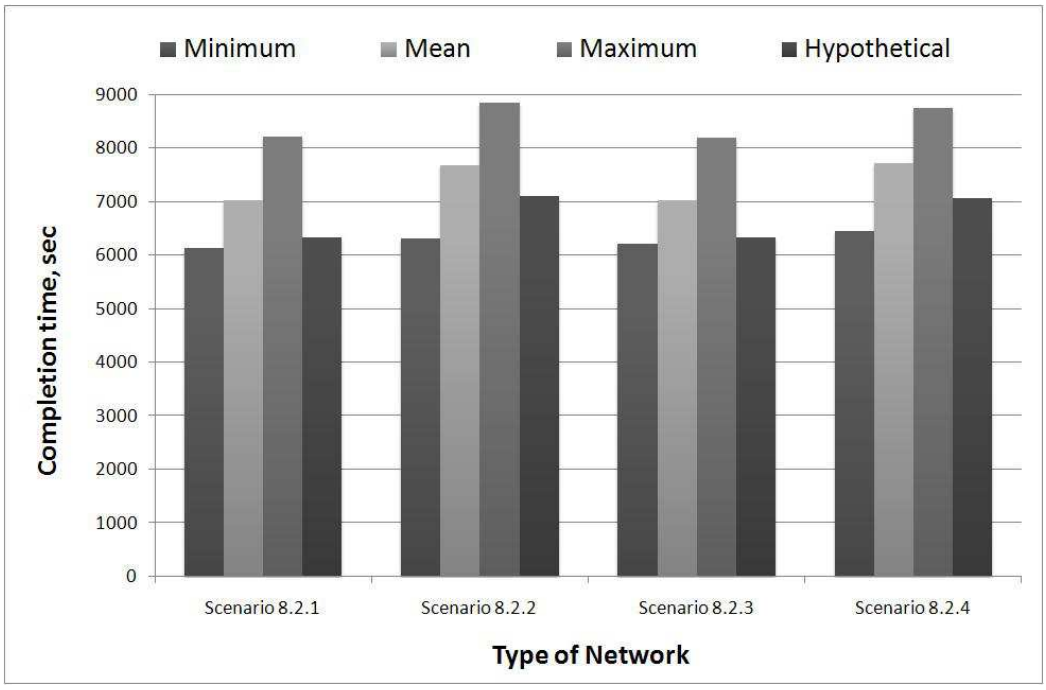
We conclude that *cNAMPs perform equally efficiently with both optimal and near-optimal number of cNAMPs. Initial distribution shows no significant dependence on either the type of topology or speeds of locations.*

Type of Balancing	Optimal Balance				Near-Optimal Balance			
Scenario class	8.1.1 $(*)$	8.1.2	8.1.3	8.1.4	8.2.1	8.2.2	8.2.3	8.2.4
Number of cNAMPs	240	192	240	187	200	176	200	172

Table 7.13: Scenario classes for the Initial Distribution Experiment



(a) Optimal Balance



(b) Near-Optimal Balance

Figure 7.8: Initial distribution

Scenario class	8.3.1	8.3.2	8.3.3	8.3.4
Number of cNAMPs	$240 + 33$	$192 + 25$	$240 + 33$	$187 + 25$

Table 7.14: Scenario classes for the *Adding More cNAMPs* Experiment

Adding More cNAMPs. This experiment analyses cNAMP effectiveness after adding more cNAMPs to a system that is in optimal or near-optimal balance. The experiment scenario classes are presented in Table 7.14. In the *Number of cNAMPs* row the first number indicates the optimal number of initially distributed cNAMPs, and the second number indicates the number of cNAMPs that additionally start on the root location.

The results in Figure 7.9 show that the mean value is very close to the hypothetical value, e.g. the mean value differs from the hypothetical value by at most 2%. The effect is due to the hidden capacity phenomenon that emerges from non-zero communication. When cNAMPs are being transferred to new location they are not executing anywhere. The total processing power is being shared amongst the cNAMPs that are *not* being transferred, so they receive more processing power and will terminate earlier. The effect

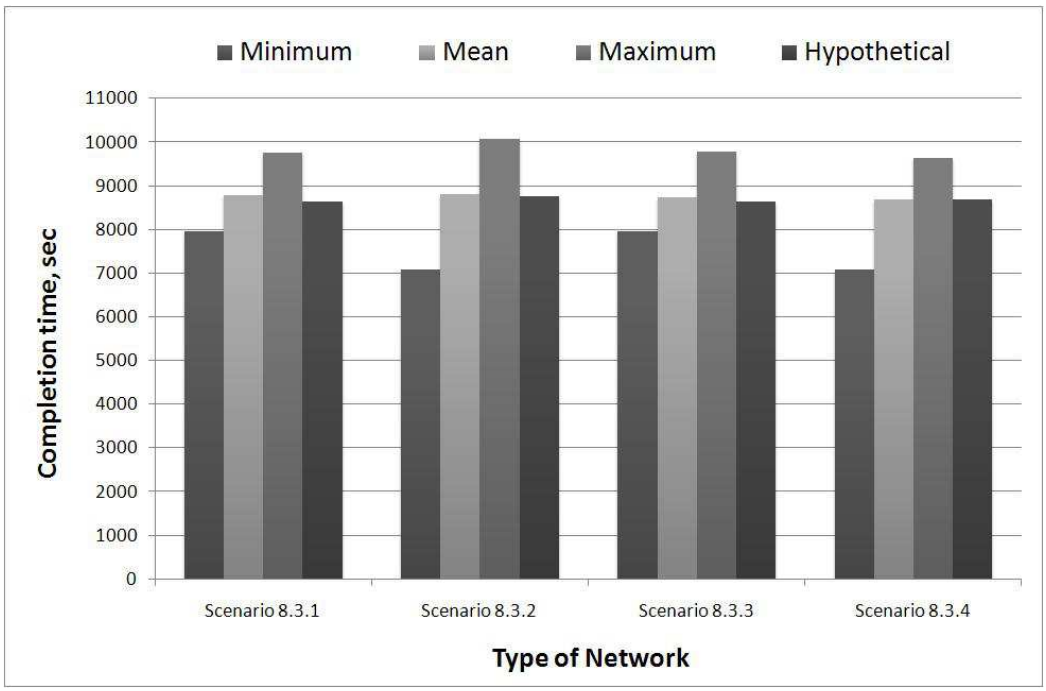


Figure 7.9: Adding More cNAMPs

is more pronounced early in distribution of cNAMPs. Thus, the mean value can even be smaller than the hypothetical value. The effect can only be observed when locations are not idle. The phenomenon impact on cNAMP completion time is also discussed in Section 7.1.6. The results lead to the conclusion that *additional cNAMPs effectively distribute themselves across the networks*.

Removing cNAMPs. This experiment examines cNAMP effectiveness after termination of some cNAMPs from optimal and near-optimal balance. The scenario classes are presented in Table 7.15. For each scenario class the table shows optimal and terminated number of cNAMPs, and cNAMP distribution at the beginning of the experiment, i.e. after termination.

The results are presented in Figure 7.10. As in the previous experiments simulated cNAMP completion time is very close to the hypothetical values, e.g. the difference between mean and hypothetical values of cNAMP completion time is within 2%. This leads to conclusion that *cNAMPs perform effective rebalancing after termination of some cNAMPs*.

Scenario	Number of cNAMPs	cNAMP Distribution at Initial Time
8.4.1	$240 - 20$	$0.0.0.0.0 - 4.4.4.4.4 - 4.4.4.4.4 - 4.4.4.4.4$
		$4.4.4.4.4 - 4.4.4.4.4 - 4.4.4.4.4 - 4.4.4.4.4$
		$4.4.4.4.4 - 4.4.4.4.4 - 4.4.4.4.4 - 4.4.4.4.4$
8.4.2	$192 - 16$	$0.0.0.0.0 - 4.4.3.3.2 - 4.4.3.3.2 - 4.4.3.3.2$
		$4.4.3.3.2 - 4.4.3.3.2 - 4.4.3.3.2 - 4.4.3.3.2$
		$4.4.3.3.2 - 4.4.3.3.2 - 4.4.3.3.2 - 4.4.3.3.2$
8.4.3	$240 - 20$	$0.0.0.0.0 - 4.4.4.4$
		$4.4.4.4.4.4.4 - 4.4.4.4.4.4.4.4.4 - 4.4 - 4.4.4.4.4 - 4.4.4 - 4.4$
		$4.4.4.4.4.4 - 4.4.4.4.4.4.4.4.4 - 4.4.4.4.4.4.4.4$
8.4.4	$187 - 16$	$0.0.0.0.0 - 4.3.2.4$
		$4.3.2.4.3.2.4 - 4.3.2.4.3.2.4.3.2 - 4.3 - 4.3.2.4.3 - 4.3.2 - 4.3$
		$4.3.2.4.3.2 - 4.3.2.4.3.2.4.3.2 - 4.3.2.4.3.2.4.3$

Table 7.15: Scenarios of *Removing cNAMPs* Experiment

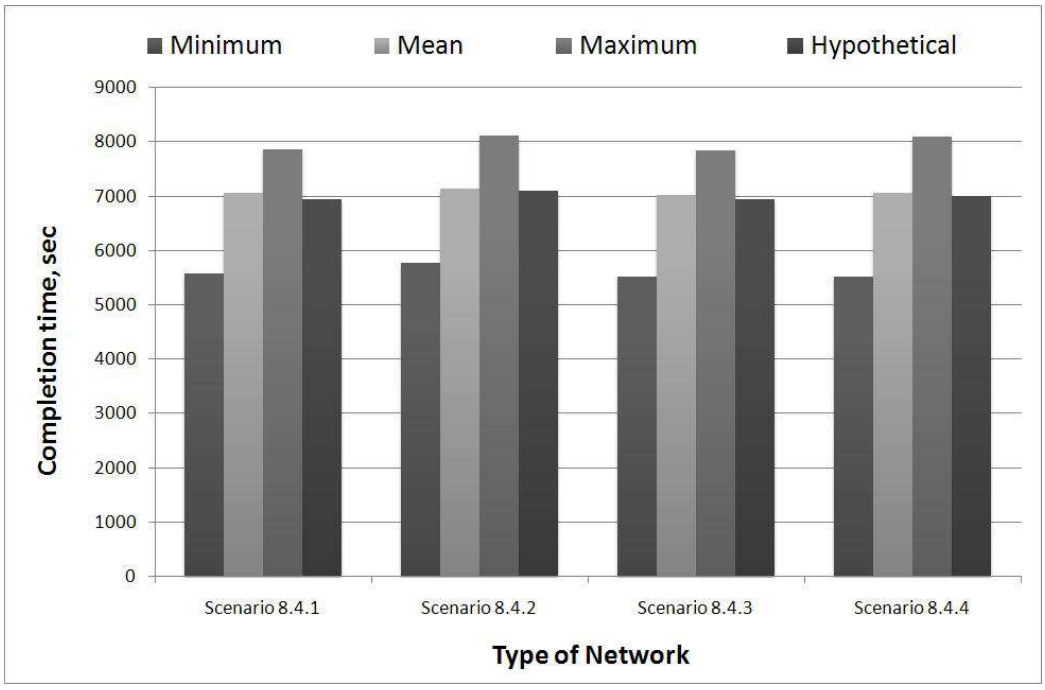


Figure 7.10: Removing cNAMPs

7.2 Redundant Movements

This section analyses the number and the type of redundant movements, and the impact of redundant movements on cNAMP completion time. The majority of experiments are conducted on the basis of the same scenarios as the experiments in Section 7.1. Therefore, the experiments only provide brief scenario descriptions and references to the corresponding scenarios. The investigation of redundant movements considers the number of levels and the type of rebalancing (Section 7.2.1), the number of locations and the number of cNAMPs (Section 7.2.2), the size of cNAMPs (Section 7.2.3), the type of cNAMPs (Section 7.2.4).

7.2.1 Experiment B1: Number of Levels

This experiment analyses the number of redundant movements depending on the number of levels. The experiment is conducted for up to five-level networks. The number of

No. of Levels	Type of Moves	Total No. of Moves	Redundant Moves		Number of Moves				
			Total	Outside	$Lev0$	$Lev1$	$Lev2$	$Lev3$	$Lev4$
Scenario class 9.1.1									
2	Optimal	76	-	-	16	60			
	Simulated	92	16 (17%)	-	32	60			
3	Optimal	236	-	-	16	60	160		
	Simulated	340	104 (31%)	-	69	111	160		
4	Optimal	476	-	-	16	60	160	240	
	Simulated	755	279 (37%)	-	147	208	160	240	
5	Optimal	1276	-	-	16	60	160	240	960
	Simulated	2496	1220 (49%)	-	356	607	333	240	960
Scenario class 9.1.2									
2	Optimal	64	-	-	64	0			
	Simulated	65	1 (2%)	13 (20%)	52	13			
3	Optimal	192	-	-	192	0	0		
	Simulated	196	4 (2%)	46 (23%)	150	31	15		
4	Optimal	384	-	-	384	0	0	0	
	Simulated	541	157 (29%)	361 (68%)	180	321	30	10	
5	Optimal	1152	-	-	1152	0	0	0	0
	Simulated	1590	438 (28%)	1081 (68%)	509	906	121	35	19

Table 7.16: cNAMP Movements during Initial Distribution in Experiment B1

cNAMPs is chosen to allow each location to have four cNAMPs in optimal balance. The topology is discussed in Section 7.1.1, and scenario classes are presented in Table 7.1. The experiment examines redundant movements in two types of rebalancing: initial distribution and rebalancing after cNAMP termination.

Initial distribution experiment analyses redundant movements between cNAMPs start the execution and the system enters a balanced state. Here, two initial scenario classes are considered:

- *Scenario class 9.1.1*: all cNAMPs start on one location.

- *Scenario class 9.1.2*: cNAMPs start on one of locations in each $Level\ 1$ group.

Table 7.16 shows the total number of movements, the number of redundant movements, and the number of movements within the levels. The optimal number of movements and the number of movements that actually occurred in the experiments are presented for each level.

Results of scenario class 9.1.1 show that redundant rebalancing occurs on the lower levels where communication cost is cheaper. In scenario class 9.1.2 the optimal rebalancing is concentrated within $Level\ 0$. However, the implemented fusion scheme is designed to allow quick distribution across the network (Section 6.3.2). Therefore, a part of cNAMPs move outside of $Level\ 0$. Column *Outside Redundant Moves* shows the number of movements outside of $Level\ 0$. The column also shows the ratio of the outside movements to the total number of movements. The positive feature here is that the vast majority of the exchange movements occurs on the lower levels.

Rebalancing after termination experiment analyses redundant movements in the interval between cNAMP termination and the system entering a balanced state. The following two scenario classes are considered:

- *Scenario class 9.2.1*: all cNAMPs terminate from one location of each group $Level\ 1$.
- *Scenario class 9.2.2*: all cNAMPs terminate from all locations of $Level\ 1$ group in each group of $Level\ 2$.

Scenario class	Number of Levels			
	2	3	4	5
9.2.1	$80-16$	$240-48$	$480-96$	$1440-288$
9.2.2	$80-20$	$240-60$	$480-120$	$1440-360$

Table 7.17: Total and Terminated Numbers of cNAMPs in Experiment B1

No. of Levels	Type of Movements	Total No. of Moves	Redundant Moves		Number of Moves				
			Total	Outside	$Lev0$	$Lev1$	$Lev2$	$Lev3$	$Lev3$
Scenario class 9.2.1									
2	Optimal	12	-	-	12	0			
	Simulated	13	1 (8%)	-	13	0			
3	Optimal	36	-	-	36	0	0		
	Simulated	36	0 (0%)	-	36	0	0		
4	Optimal	72	-	-	72	0	0	0	
	Simulated	73	1 (1%)	-	73	0	0	0	
5	Optimal	216	-	-	216	0	0	0	0
	Simulated	221	5 (2%)	-	221	0	0	0	0
Scenario class 9.2.2									
2	Optimal	15	-	-	0	15			
	Simulated	19	4 (21%)	3 (16%)	3	16			
3	Optimal	45	-	-	0	45	0		
	Simulated	52	7 (14%)	8 (15%)	2	44	6		
4	Optimal	90	-	-	0	90	0	0	
	Simulated	107	17 (16%)	21 (20%)	8	86	11	2	
5	Optimal	270	-	-	0	270	0	0	0
	Simulated	327	57 (17%)	77 (24%)	13	250	37	18	9

Table 7.18: cNAMP Movements after cNAMP Termination in Experiment B1

The total number of cNAMPs and the number of terminated cNAMPs in each experiment are presented in Table 7.17.

The results of the experiments are presented in Table 7.18. In scenario classes 9.2.1 and 9.2.2 the optimal rebalancing in concentrated between nodes of $Level\ 0$ and $Level\ 1$ respectively. In scenario classes 9.2.1 the redundant number of movements exceeds the optimal number of movements by up to 8%. The results of scenario class 9.2.2 show that cNAMPs make up to 21% of redundant movements, whereas the number of movements outside $Level\ 1$ does not exceed 24%.

We conclude that although *cNAMPs may have a large number of redundant movements*, e.g. 49% and 21% of redundant movements in scenario classes 9.1.1 and 9.2.2 respect-

ively, *these movements do not effect completion time dramatically* (Section 7.1.1).

7.2.2 Experiment B2: Number of Locations

This experiment analyses the number of redundant movements depending on the number of locations. The scenarios are conducted on three-level networks with the topologies specified in Table 7.4. Each scenario has three variants. The number of cNAMPs for each variant is chosen in such a way that each scenario would analyse redundant movements for a different number of cNAMPs, i.e. each location would have either one, four, or ten cNAMPs in optimal balanced states. The redundant movements during initial distribution are measured.

The experiment results are presented in Table 7.19. The table shows optimal and redundant number of movements and the number of movements on each level for different numbers of cNAMPs and locations. As for the previous experiments the redundant rebalancing occurs mainly on lower levels, i.e. $Level\ 0$ and $Level\ 1$, which have cheaper communication cost than the upper level, i.e. $Level\ 2$. The percentage of redundant movements is calculated as a ratio of the number of redundant movements to the total number of movements.

Figure 7.11 shows the number of redundant movements depending on the number of locations for different numbers of cNAMPs. The number of redundant movements has some dependence on the number of locations in small networks that becomes less significant in networks of more than 135 locations. Recall, the current section only investigates redundant movements and not their impact on cNAMP completion time. Table 7.19 shows cNAMP movement overhead whereas completion time overhead is discussed in Section 7.1.3. Analysis of Table 7.19 and Figure 7.11 shows that although overhead of cNAMP movements is can be high, e.g. 40% in the experiment with 540 cNAMPs on 135 locations (Table 7.19), mean cNAMP completion time overhead for the same experiment is only 8% (Figure 7.11). This allows us to conclude that *the number of*

No. of Locs	Type of Moves	Number of Movements														
		Total	Redun.	$L.0$	$L.1$	$L.2$	Total	Redun.	$L.0$	$L.1$	$L.2$	Total	Redun.	$L.0$	$L.1$	$L.2$
		One cNAMP per Location					Four cNAMPs per Location					Ten cNAMP per Location				
8	Opt.	7	-	1	2	4	28	-	4	8	16	70	-	10	20	40
	Simul.	10	3 (3%)	4	2	4	34	6 (18%)	9	9	16	81	11 (14%)	20	21	40
60	Opt.	59	-	4	15	40	236	-	16	60	160	590	-	40	150	400
	Simul.	92	33 (36%)	24	28	40	340	104 (31%)	69	111	160	849	259 (31%)	184	265	400
135	Opt.	134	-	8	18	108	536	-	32	72	432	1340	-	80	180	1080
	Simul.	224	90 (40%)	83	33	108	891	**355 (40%)**	278	181	432	2201	861 (39%)	607	514	1080
192	Opt.	191	-	5	18	168	764	-	20	72	672	1910	-	50	180	1680
	Simul.	313	122 (39%)	66	79	168	1223	459 (38%)	219	332	672	3119	1209 (39%)	509	930	1680
264	Opt.	263	-	10	77	176	1052	-	40	308	704	2630	-	100	770	1760
	Simul.	430	167 (39%)	106	148	176	1686	634 (38%)	409	573	704	4142	1512 (37%)	904	1478	1760
336	Opt.	335	-	6	77	252	1340	-	24	308	1008	3350	-	60	770	2520
	Simul.	567	232 (41%)	100	215	252	2182	842 (39%)	396	778	1008	5320	1970 (37%)	930	1870	2520

Table 7.19: cNAMP Movements during Initial Distribution in Experiment B2

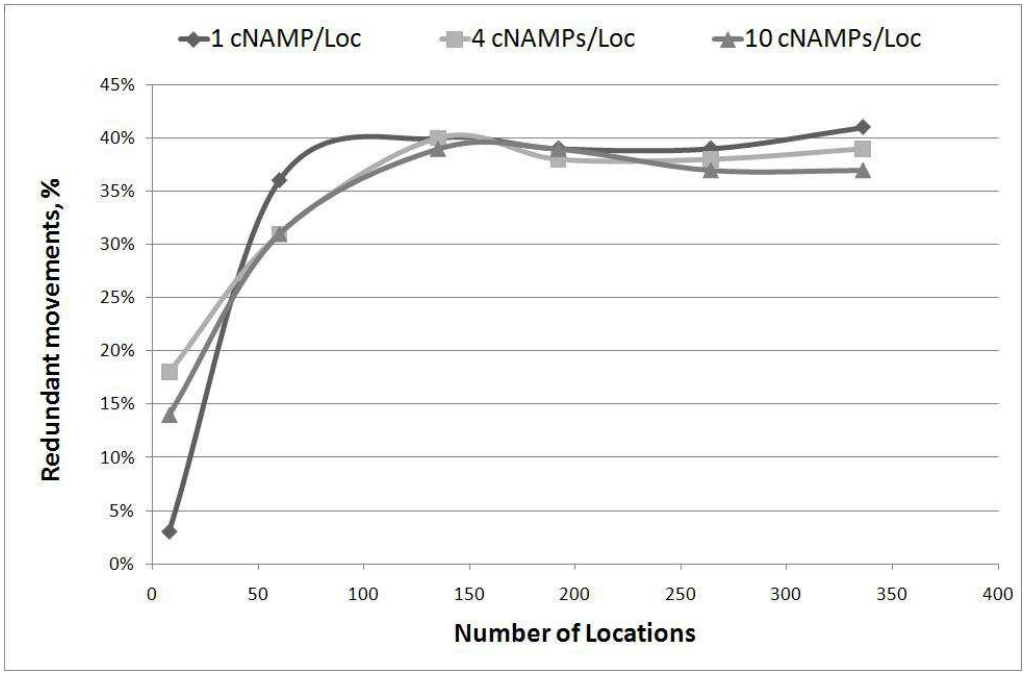


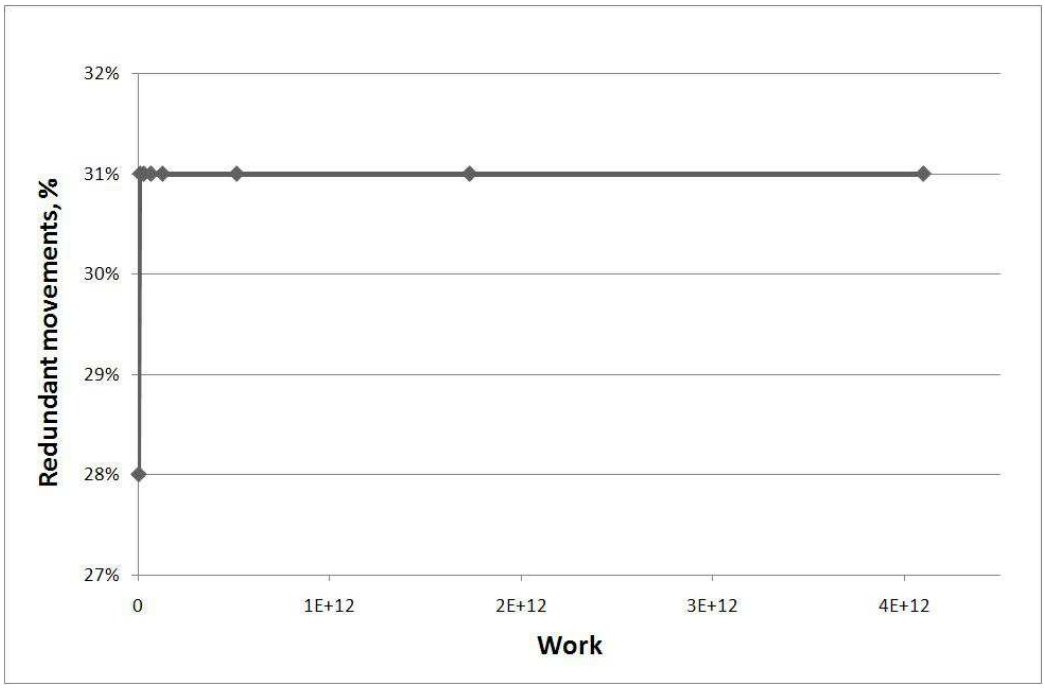
Figure 7.11: Redundant Movements vs. Number of Locations

redundant movements depends on the number of cNAMPs that simultaneously start on one location and the size of networks. This dependency is observed in small networks of up to 135 locations. In the larger networks the number of redundant movements does not indicate significant dependency on the parameters.

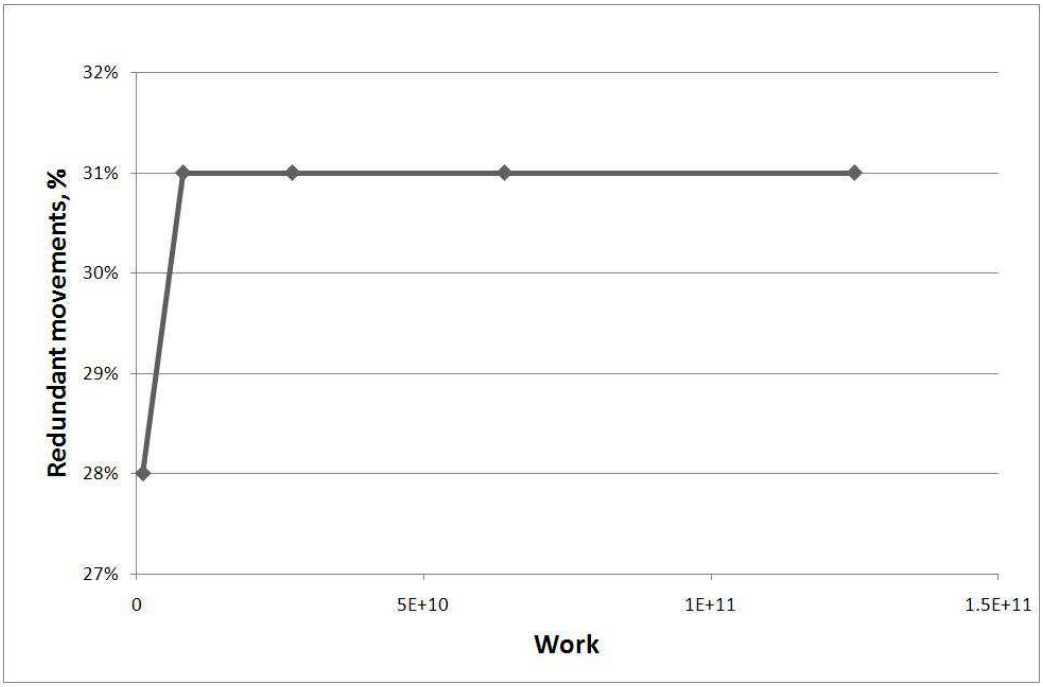
7.2.3 Experiment B3: Work of cNAMPs

This experiment analyses redundant movements depending on the size of cNAMPs. The topology is discussed in Section 7.1.5. The experiment is conducted using square matrix multiplication programs of dimensions: 1000, 2000, 3000, 5000, 8000, 12000, and 16000.

The results presented in Figure 7.12 show that as soon as cNAMPs have enough work to enter an optimal balanced state and the remaining execution time becomes much larger than communication time *the ratio of redundant movements to the total number of movements stays the same and does not depend on the size of cNAMPs*. The experiment



(a) Full Results



(b) Scaled Results

Figure 7.12: Redundant Movements vs. Work of cNAMPs (B3)

State	Type of Movements	Number of Movements				
		Total	Redundant	*Level* 0	*Level* 1	*Level* 2
10 cNAMP per Location						
Stable	Optimal	236	-	16	60	160
	Simulated	326	90 (28%)	77	104	145
Optimally Balanced	Optimal	236	-	16	60	160
	Simulated	340	104 (31%)	69	111	160

Table 7.20: cNAMP Movements in Experiment B3

results are as expected, because independently of program size cNAMPs go through the same steps of gradual rebalancing.

7.2.4 Experiment B4: Type of cNAMPs

This experiment analyses redundant movements depending on the type of cNAMPs, i.e. programs of coin counting, ray tracing, matrix multiplication and mixture of the three. The redundant movements of 240 cNAMPs are examined on a three-level network during initial distribution. The topology is the same as discussed in Section 7.1.6.

The results presented in Table 7.21 and Figure 7.13 show that *type of programs has no significant affect on redundant movements*. Experiments with coin counting show 5% larger number of redundant movements in comparison with the rest of the experiments. This is due to the small communication time of coin counting programs that allows

Type of Movements	Type of cNAMPs	Number of Movements				
		Total	Redundant	*Level* 0	*Level* 1	*Level* 2
Optimal	-	236	-	16	60	160
Simulated	Coin Counting	367	131 (36%)	77	130	160
	Ray Tracing	341	105 (31%)	70	111	160
	Matrix Multiplication	341	105 (31%)	70	111	160
	Mixture	341	105 (31%)	73	108	160

Table 7.21: cNAMP Movements during Initial Distribution in Experiment B4

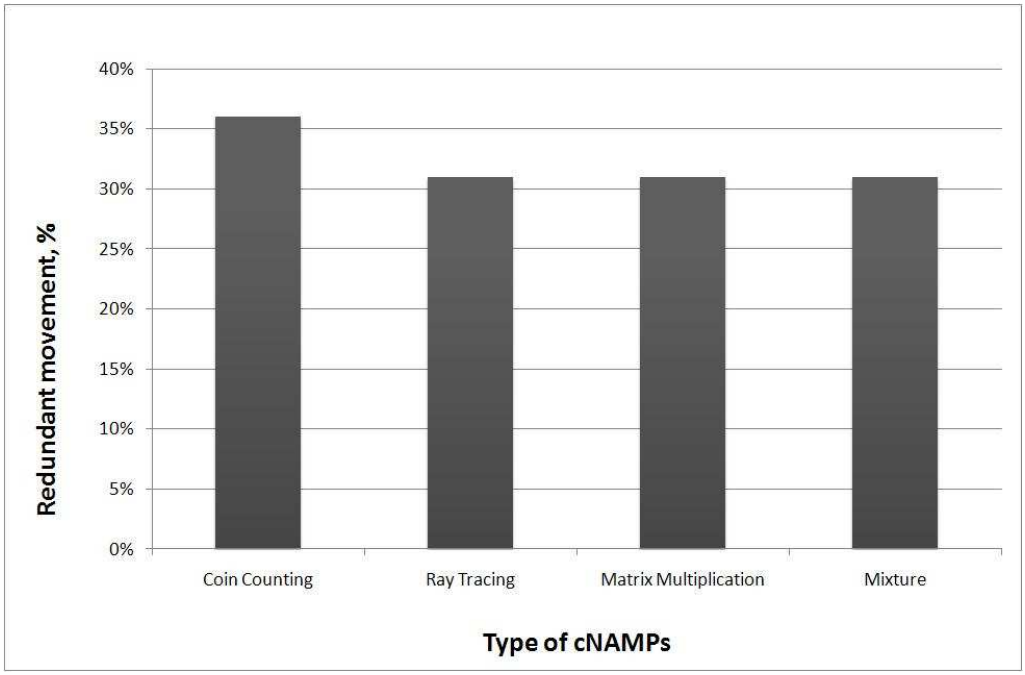


Figure 7.13: Redundant Movements vs. Type of cNAMPs (B4)

cheaper cNAMP relocation (Section 7.1.6).

7.3 Discussion

The analysis of cNAMP fusion scheme on multilevel networks has been conducted on the basis of twelve experiments where eight experiments examine cNAMP effectiveness (Section 7.1), and four experiments examine redundant movements (Section 7.2). The effectiveness and redundant movements are estimated depending on network parameters (i.e. number of levels, topologies, number of locations, speed of locations), cNAMP parameters (i.e. number of cNAMPs, work of cNAMPs, type of cNAMPs) and type of rebalancing (i.e. initial distribution, rebalancing after adding cNAMPs and termination cNAMPs). The results validate the multilevel architecture by showing that cNAMPs distribute themselves across the networks, have small variance of completion time, and the results are very close to the hypothetical values.

The type of topology, the number of locations, the speed of locations, and the type of

rebalancing show *no significant effect* on cNAMP completion time (Experiments A2, A3, A7, A8). In contrast the number of levels, the number of cNAMPs, and the amount of work has a *direct effect* on cNAMP completion time (Experiments A1, A4, A5). Thus, the experiment on number of levels where up to five levels had been examined shows that each additional level increases the mean cNAMP completion time by 1%–3%, and the mean completion time in a five level network differs from the hypothetical value by 12%. In the experiments on work of cNAMPs with the increase of work the relative difference between experimental and hypothetical completion time decreases.

The experiment with different types of cNAMPs shows that completion time discrepancy depends on relative difference between cNAMP communication time and cNAMP completion time. The larger the difference the less the discrepancy. The results also show that cNAMPs with different communication time use computer resources more effectively as they utilise hidden resources, i.e. resources that are booked but are not used during cNAMP transferring (Experiment A6).

The analysis of redundant movements shows that cNAMPs gradually distribute themselves from the closest to the remote locations. Such parameters as the work of cNAMPs and the type of cNAMPs demonstrate *no significant effect* on the number of redundant movements (Experiments B3, B4). The type of distribution and the number of levels have the largest effect on number of redundant movements (Experiment B1). The number of locations and the number of cNAMPs have direct impact up to the certain point (Experiment B2). The implementation of different design alternatives from Section 6.3.1 may help to more effectively balance the speed of cNAMP distribution between nearest and remote locations. However, even in the current design the redundant movements show no significant impact on cNAMP completion time.

Chapter 8

Conclusion and Future Work

8.1 Summary

Chapter 2 surveys the key concepts addressed in the thesis, i.e. load management, mobility, autonomous systems, AMPs, network scale, topology, and simulators. The chapter also situates AMPs among other load balancers using taxonomies, and describes how collections of AMPs differ from other load management systems.

Chapter 3 analyses AMPs on homogeneous and heterogeneous LANs, and validates simulation against the real experiment results conducted on Java Voyager in [Den07]. In total fourteen experiments are conducted: twelve for homogeneous networks and two for heterogeneous networks. On homogeneous networks the experiments are as follows: seven experiments for optimal balance, two experiments for near-optimal balance, two experiments for adding AMPs, and one experiment for removing AMPs. The results show that simulated and real AMPs enter the same balanced states except for a few minor and explainable deviations. For example, the mismatch in *optimal balance* and *adding AMP* experiments is due to the using the communication workload in the simulation experiments equal to 50%, whereas in the real experiments the workload varies between

48% and 51%. In *removing AMP* experiments 18% of the simulated experiments enter all of the states entered in the real experiments; the other states AMPs enter are also stable (Section 3.3). The conclusion from the analysis is that the simulation is a suitable tool for investigation networks of AMPs.

Chapter 4 identifies two types of redundant movements or greedy effects: location thrashing causes additional movements *and* increase in AMP completion time; location blindness causes only additional movements, as all transferred AMPs improve their execution environment. The simulation model discussed in Chapter 3 is adapted to simulate the greedy effects. The results show that in the three experiments considered the mean number of redundant movements per AMP is two (Table 4.1). The number of redundant movements is proportional to the number of AMPs, number of locations, and the work of AMPs. These redundant movements are mainly caused by location thrashing.

To eradicate location thrashing negotiating AMPs with competitive scheme (cNAMPs) are described and implemented. The two key differences of cNAMPs from AMPs are that a cNAMP sends a representative to confirm the movement, *and* each location has two values of its load: actual load is used by local cNAMPs and committed load is published to other locations (Section 4.4.2). The analysis of cNAMP simulation results shows that cNAMPs exhibit only location blindness. cNAMPs do not make redundant movements during initial distribution, and all three scenarios show at least three times faster initial balancing in comparison with AMPs. During rebalancing after an AMP/cNAMP termination, cNAMPs make far fewer redundant movements, and the cNAMP rebalancing takes less than half of the time of AMP rebalancing. In consequence cNAMPs require less completion time than AMPs (Table 4.4).

Chapter 5 provides the first substantial investigation of redundant movements in distributed collections of autonomous mobile agents. The chapter establishes balanced state properties to estimate the cNAMP greedy effect. AMPs are investigated as a general case because modifications of cNAMPs do not affect balanced states, and both AMPs

and cNAMPs enter the same balanced states. The chapter defines the following three properties: *independent balance*, *singleton optimality*, *consecutive optimality*. Optimal and near-optimal balanced states for homogeneous and heterogeneous networks are also characterised (Section 5.1). A set of consistent and rigorous definitions are assembled into the Glossary and are essential for reasoning about AMP/cNAMP behaviour.

The significance of greedy effects is established by predicting the worst case (maximum number) of redundant movements after a cNAMP termination from a network of q subnetworks. The results show that a difference in the number of cNAMPs needs to be at least two before a movement will occur between locations of a same subnetwork (*minimum difference* criterion). The analysis is summarised in three theorems and three lemmas. For example, a system with q subnetworks makes at most $q - 2$ redundant movements after a cNAMP termination from an optimally balanced network (Lemma 9 on page 102). The calculation of probability shows that the median probability of these $q - 2$ redundant movements to occur is less than 1% (Appendix D.4).

Chapter 6 develops a fusion-based architecture to extend cNAMPs beyond LANs to multilevel networks like WANs. It starts by discussion of means and approaches to simulate large networks, and justifies the reasons to use multilevel network approach. The proposed multilevel network abstracts from both network topology and location architecture. The chapter provides design alternatives for cNAMPs on multilevel networks considering such parameters as number of parental gateways, type of mobility, gateway functionality, type of node exchange information, conditions of movements to remote locations, and conditions of request movements. The discussions of cNAMP components together with pseudocode and justification of simulated parameters are provided for the implemented fusion scheme.

Chapter 7 evaluates the multilevel fusion-based architecture by showing that cNAMPs distribute themselves across the networks, have small variance of completion time, and the results are very close to the hypothetical values. The analysis is conducted on the

basis of twelve experiments where eight experiments examine cNAMP effectiveness (Section 7.1), and four experiments examine redundant movements (Section 7.2). The effectiveness and redundant movements are estimated depending on such parameters as network parameters (i.e. number of levels, topologies, number of locations, speed of locations), cNAMP parameters (i.e. number of cNAMPs, work of cNAMPs, type of cNAMPs) and type of rebalancing (i.e. initial distribution, rebalancing after adding cNAMPs and termination cNAMPs).

The topology, the number of locations, the speed of locations, and the type of rebalancing show *no significant effect* on cNAMP completion time. In contrast the number of levels, the number of cNAMPs, and the amount of work has a *direct effect* on cNAMP completion time. The experiment with different types of cNAMPs shows that cNAMPs with different communication time use computer resources more effectively as they utilise resources that are booked but are not used during cNAMP transferring (Section 7.1). The analysis of redundant movements shows that cNAMPs gradually distribute themselves from the closest to the remote locations. Such parameters as the work of cNAMPs and the type of cNAMPs demonstrate *no significant effect* on the number of redundant movements. The type of distribution and the number of levels have the largest impact on the number of redundant movements (Section 7.2). Overall, the redundant movements show no significant impact on cNAMP completion time.

8.2 Limitations

The limitations of the research presented in this thesis are outlined below. The limitations apply to both AMPs and cNAMPs.

- *System reliability*. The design of AMPs assumes that locations do not disappear from the network. Location software and hardware, and AMPs themselves are also supposed to work correctly. Thus, in case of one of the above component

failure the AMPs associated with the location require to be restarted. The current research did not aim to investigate all conditions of AMP successful execution, and hence the system reliability was taken for granted. One possible solution to this issue might be adoption of either replication or check pointing schemes in AMPs [PBKY02], e.g. a periodic sending of a back-up copy to either the root location or a trusted gateway.

- *Equal sharing of the CPU power*. Currently, it is assumed that AMP input/output time is negligibly small and the majority of the AMP execution time is destined for computation. Thus, to calculate relative speed the location available speed is equally divided between the AMPs, i.e. round robin job scheduling [DKS89]. To apply different partition of CPU power such uniprocessor scheduling approaches like task aware scheduling and priority scheduling can be considered [SSDNB95, Pin08]

- *Sufficiency of resources*. The AMP cost model takes into account only available speed and communication time but does not take into account other resources such as available RAM, and space on a hard disc. Thus, in case of insufficient resources either the AMP execution speed will be lower than the declared or the AMP will be prevented from transfer to the target location. The solution might be in multi-objective optimisation where AMPs choose the best location not only on the basis of time but taking into account other parameters such as available RAM, space on a hard disc, cost of transferring and execution.

8.3 Future Work

Possible future development of the work might include the following directions:

- *Design AMPs for Clouds*. The Cloud is a rapidly expanding technology that provides computational and storage services on a pay-as-you-go basis. The invest-

igation and developing a self-organising and distributing system for cloud computing on the basis of AMPs and brokers is a promising technology. An AMP-broker based cloud might execute programs either faster for the given money or cheaper during the given period in comparison with currently exploited static schemes. In these static schemes a program after being submitted to a cloud has no opportunity to move to another cloud (provider) automatically without user interference. AMPs might move between clouds choosing *the best* cloud in terms of time-cost trade-off within the given limits of either time or funds using multi-objective optimisation. The AMPs target Infrastructure-as-a-Service (IaaS) based Clouds, such as Amazon EC2 [ama11], Eucalyptus based clouds [NWG+09] like StACC [sta11].

Initially, the AMP movements might be defined by two parameters: time and cost [Gar11], i.e.

$$\begin{cases} T_h > T_n + T_{comm}, \\ C_h > C_n + C_{comm}. \end{cases}$$

Here, C_h is the execution cost on the current node, C_n is the execution cost on the new node, and C_{comm} is the transfer cost. Before the execution starts the user might define either the maximum price that is allowed to be spent on the AMP execution or the maximum time during which the AMP must be executed. Later the number of parameters that are taken into account might be expanded.

- *Implementation cNAMPs on WANs*. Another prospective research direction is validating cNAMP design by constructing cNAMPs from Section 6.3.2 using mobile languages like JavaGoX [SSY00] and Java Voyager [Rec10]. The implementation can be done either independently (e.g. for different types of programs) or in a collaboration with another research (e.g. focus on a particular type of programs, for example, programs that use genetic algorithms). By a collaboration is meant that collaborators will work on specific goals of the encapsulated programs, and AMP developers will aim to accelerate the program completion time by moving them within a network. In both cases the goal of AMP developers will be to quickly and effectively encapsulate the programs and find the most suitable prediction mech-

anism. Here, not only minimising completion time can be targeted but also other parameters, such as reliability, security, location configuration, and program priority.

- *Negotiating AMP alternatives*. Negotiating AMPs (NAMPs) can be implemented in a number of ways. The examples of the alternative designs are discussed in Section 4.4.1. The current thesis considers negotiation as a simple coordination among competitive and self-interested agents [Wei99]. Other forms of more complicated negotiation between AMPs and/or load servers are also of interest. One of the possible direction is investigation of NAMPs using game theory.

- *Multilevel cNAMP alternatives*. Section 6.3.1 discusses design alternatives of cNAMPs on multilevel networks. The proposed fusion scheme shows that completion time is very close to the hypothetical value (Section 7.1) in spite of a large number of redundant movements in selected experiments (Section 7.2). The investigation of alternative schemes may lead to reduction of redundant movements and overall cNAMP completion time. The further investigation may also lead to defining cNAMP properties on multilevel networks. The current research analyses cNAMPs on up to five level networks and up to 336 locations. The increase of the scale will also help to examine trends and regularities.

Glossary

Table 8.1 assembles the main notions defined in the paper. In column *Source* 1 denotes a definition from [Den07], 2 a generalisation of a definition from [Den07], and 3 a more precise definition than in [Den07]. To clarify the difference and similarity of the concepts column *Related Concept* provides lists of related concepts. Column *Section* gives the numbers of the sections where the notions are introduced in the paper.

Table 8.1: Glossary

Term & Definition	**Source**	**Related Concept**	**Section**
The *actual load* is the number of executing cNAMPs on a location. It is used for local cNAMP calculations.		committed load	4.4.2
An *AMP relative speed*[1] R, is an available speed, S, equally divided between the AMPs at a location, x_{loc}, i.e. $R = \frac{S}{x_{loc}}$.	3	available speed	2.4
Autonomous mobile programs (AMPs) are mobile agents that improve execution efficiency by managing load; AMPs are aware of their resource needs, sensitive to the execution environment and periodically seek a better location to reduce execution time.	1	cNAMP, load server	1.1
The *available speed*[2] of a location is the execution speed of a single AMP on that location, i.e. $S = (CPU speed) \cdot (1 - non\ AMP\ load)$.	3	AMP relative speed	2.4
In a *balanced state* no AMP can gain a greater AMP relative speed by moving.	1	near-opt. balance, optimal balance, stable state	3.3

[1] Termed *average relative speed* in [Den07].

[2] Termed *relative speed* in [Den07].

cNAMPs are negotiating AMPs with an honest competitive scheme which announce their intentions to move and compete with each other for an opportunity to transfer to the new location.		AMP, load server	4.4.2
The *committed load* represents the actual load of a location together with the cNAMPs that have received confirmation to transfer to the location. It is used by remote load servers.		actual load	4.4.2
A *communication cost* is the number of AMP movements during a rebalancing.		greedy effects, opt. rebalancing	4.1.2
Consecutive optimal property. If a system with a total of x AMPs is optimally balanced, *and* the subnetwork with the highest AMP relative speed is a singleton, then the system with a total of $x + 1$ AMPs is also optimally balanced.		optimal balance, singleton subnetwork	5.1.4
Greedy effects are the result of a non-optimal AMP rebalancing which differs from the optimal rebalancing in having additional redundant movements, and is a result of the AMP making a locally optimal choice.		location blindness, location thrashing, optimal rebalancing	4.1
A *heavy location* is a location with the optimal number of AMPs.		light location, optimal number of AMPs, root location	5.1.5
A *heterogeneous network* is a set of locations with different available speeds.	2	homogeneous network, subnetwork	3.4
A *homogeneous network* is a set of locations with the same available speed, except for the root location which may have reduced speed, because of the communication with the remote processes that have migrated away from the root location.	2	heterogeneous network, subnetwork	3.3

Independent balance property. For a balanced state, the relationship between the number of AMPs x_i and x_j on locations in any two subnetworks i and j is independent of the number of locations in those subnetworks and independent of the presence or absence of other subnetworks, subject only to the sum $x = x_i + x_j$ being constant. The only exception to this rule is the case when distribution of $x' = x_i + x_j + 1$ AMPs results in all x' AMPs having the same relative speed. In this case the partition may have two variants.		balanced state, subnetwork	5.1.2
A *light location* is a location which has one AMP less than the optimal number of AMPs in a heavy location.		heavy location, very light location	5.1.5
A *load server* on a location collects network state information to reduce AMP coordination time.	1	AMP, cNAMP	2.4
Location blindness is the greedy effect resulting from an AMP's lack of information about the remaining execution time of other AMPs.		greedy effects, location thrashing, opt. rebalancing	4.1.3
Location thrashing is the greedy effect resulting from an AMP's lack of information about other AMPs intending to move to the same location.		greedy effects, location blindness, opt. rebalancing	4.1.2
Minimum difference criterion. The number of AMPs on locations with the same available speed must differ by at least two before AMPs will move between them.		balanced state, stable state	5.2.1
A *near-optimal balanced* network is defined be the network where some networks have near-optimal number of AMPs. The locations of these underloaded subnetworks have either the optimal number of AMPs or one less than the optimal number. The underloaded subnetworks are determined by being the subnetworks with the slowest AMP relative speed in the nearest upper optimal balanced state.	3	balanced state, nearest upper (lower) optimal balanced state, optimal balance, subnetwork	5.1.4

Nearest upper (lower) optimal balanced state is the optimal balanced state which the system enters by adding (removing) the minimum number of AMPs.		near-opt. balance, optimal balance	5.1.4
In *optimal balance* locations with the same available speed have equal numbers of AMPs.	2	balanced state, near-opt. balance	3.3.1
For any optimal balanced state, the *optimal number of AMPs* for a subnetwork is the number of AMPs on each location in the subnetwork.		heavy location, optimal balance, subnetwork	5.1.4
An *optimal rebalancing* is a sequence of AMP movements that is the minimum number of AMP movements needed to enter a stable state.		greedy effects	4.1
The *root location*[3] is the location where all AMPs start.		heavy location, light location	2.4
Singleton optimal property. All balanced states which a network of singleton subnetworks enters are optimally balanced.		balanced state, optimal balance, singleton subnetwork	5.1.3
A *singleton subnetwork* is a subnetwork with one location.		subnetwork	5.1
In a *stable state* no AMP can reduce its execution time by moving.		balanced state	3.4
A *subnetwork* is a set of locations with identical available speeds.		heterogen. network, homogen. network,	5.1
A *very light location* is a location which has two AMPs less than the corresponding heavy location.		heavy location, light location	5.1.5

[3] It is either called *initiating location* or *first location* in [Den07]

Chapter A

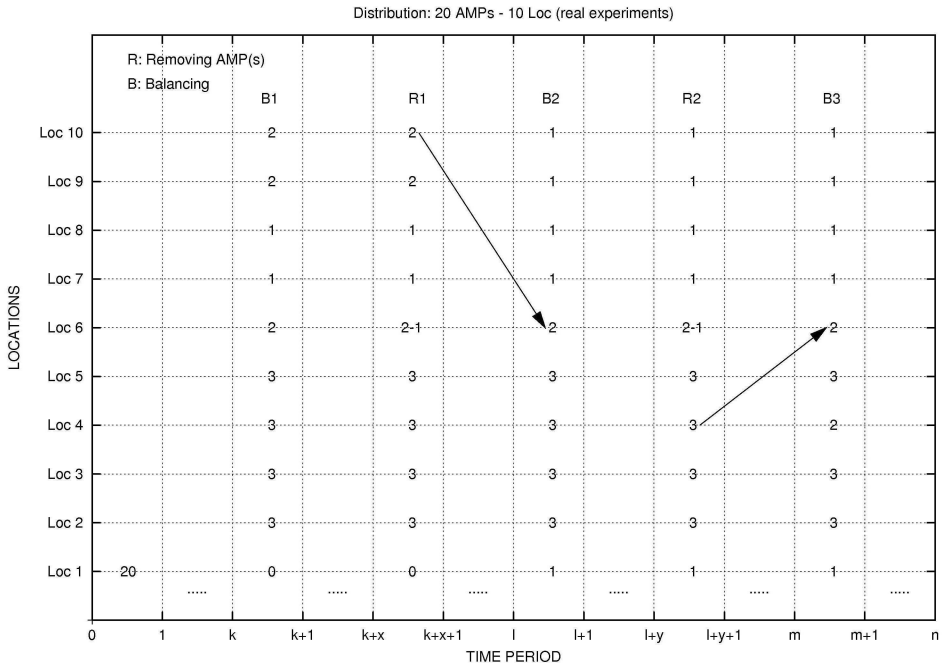
Distribution of 20 AMPs on 10 Locations

The experiment examines the distribution of 20 AMPs on 10 locations. The locations have the following CPU speeds: 3193 MHz ($Loc1 - Loc5$), 2168 MHz ($Loc6$), 1793 MHz ($Loc7 - Loc10$). Again the locations are divided into slow ($Loc1 - Loc5$), middle ($Loc6$) and fast ($Loc7 - Loc10$) speed locations depending on their available speeds. 10 large and 10 small AMPs start on $Loc1$. Figures A.1(a) and A.1(b) show AMP distribution in the real and simulated experiments respectively.

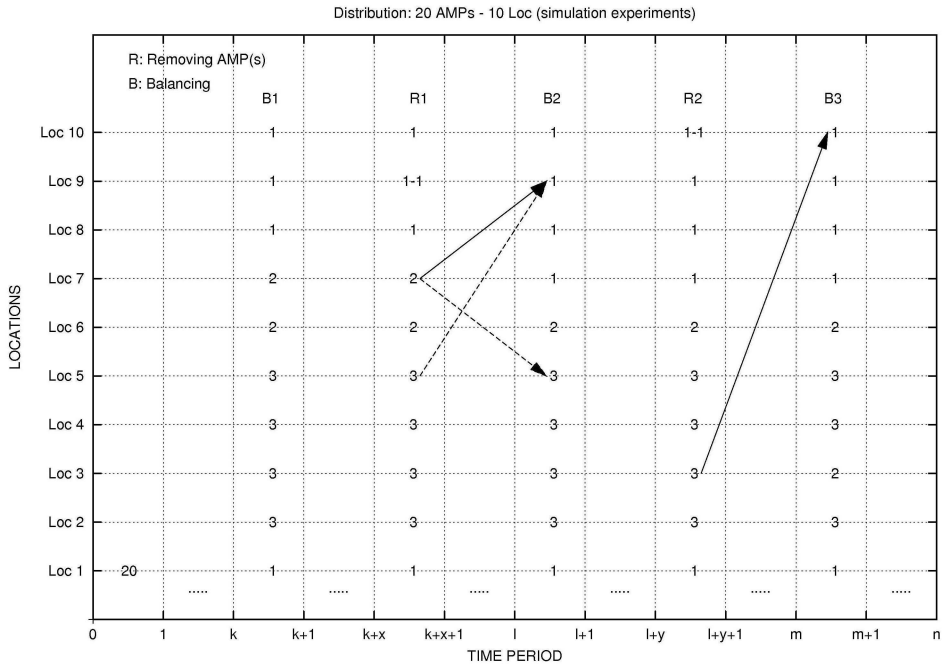
The types of AMP distribution after termination of the first and the second AMPs are presented in Table A.1. The first column shows the maximum number of AMPs on middle and slow speed locations. Here, the first digit is the number of AMPs on a middle speed location and the second digit is the maximum number of AMPs on a slow speed location. An example of the real experiment presented in Figure A.1 enters distribu-

Type of Distribution	After the 1^{st} AMP Termination	After the the 2^{st} AMP Termination
.../2/2...	18%	11%
.../2/1...	82%	58%
.../1/2...	-	6%
.../1/1...	-	25%

Table A.1: States after AMP Termination



(a) Real



(b) Simulated

Figure A.1: Distribution of 20 AMPs on 10 Locations

No. of movements	After the 1^{st} AMP Termination	After the 2^{nd} AMP Termination
0	-	29%
1	17%	71%
2	83%	-

Table A.2: Number of Movement after AMP Termination

tion '.../2/1...'. This distribution is also the most common in the simulated experiments, i.e. 82% and 58% of simulated experiments have this distribution after the first and the second AMP termination respectively. Distribution '.../1/1...' after termination of the second AMP is also balanced and depends on the type of location from which an AMP discovers a better opportunity to move first. Uncommon distributions '.../2/2...' and '.../1/2...' are the result of comparatively small computations performed by AMPs and high CPU speeds of locations, i.e. after AMP termination the remaining execution time of remaining AMPs on the slower locations is less than the movement and execution on

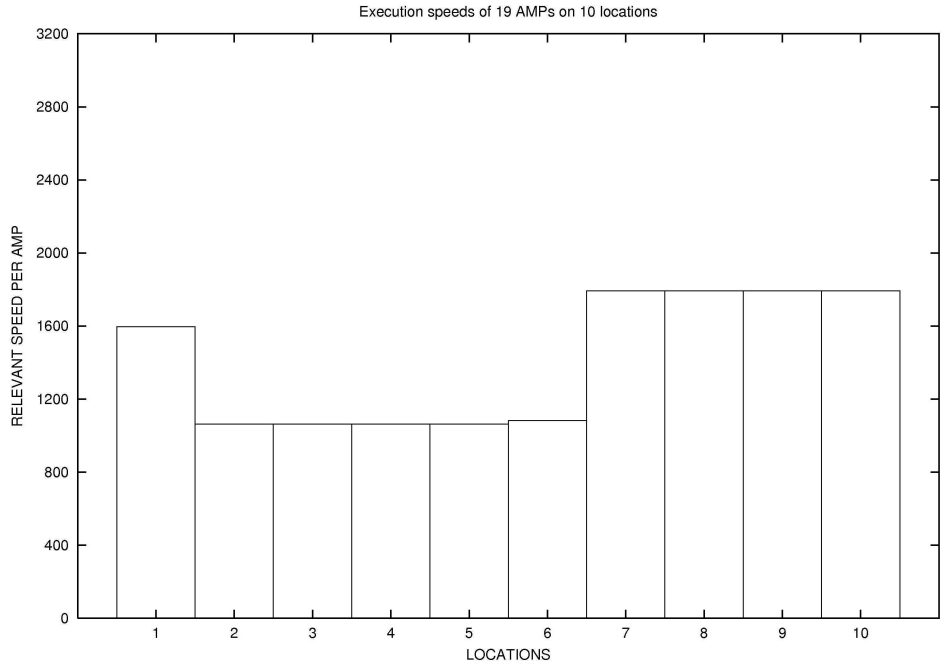


Figure A.2: Relative CPU Speeds of 19 AMPs on 10 Locations in Distribution '.../2/1...'

a faster location; therefore, AMPs choose not to move.

Figure A.2 shows AMP relative speeds in balanced distribution '.../2/1...' with 19 AMPs. From the figure we conclude that AMP relative speeds are assigned uniformly in all balanced states. Table A.2 shows the number of AMP movements in the simulated experiments after AMP termination. The reason the additional movements occur is the same as discussed in Section 3.4. *The results of 20 AMP distribution on 10 locations agree with the result of 25 AMP distribution on 15 locations.*

Chapter B

C++ Code of Balanced State Checker

```
#include<iostream>
using namespace std;

int fl(int x);

int main ()
{
 int i, j, k, m;
 int q, numAMPs, num_Rmax, num_distr, rep, num_floatAMPs;
 int d[1000], x[1000], list_Rmax[1000];
 double R_max, S[1000], R[1000];

 //Defining the values
 q = 5;      // q is the number of subnetworks
 S[1] = 5; S[2] = 7; S[3] = 10; S[4] = 12; S[5] = 15; // Available speed
 // x[i] is the number of AMPs in subnetwork i
 // R[i] is an expected AMP relative speed

 met1: cout << "Number of AMPs: ";
 cin >> numAMPs;

 // Zeroing the number of AMPs in all subnetworks
```

```
for (i = 1; i <= q; i++)
    x[i] = 0;

// Calculation of the balanced states
j = 1;
while (j <= numAMPs)
    {
     // Calculation of the maximum AMP relative speed
     R_max = 0;
     for (i = 1; i <= q; i++)
         {
          x[i]++;
          R[i] = S[i] / x[i];
          if (R[i] > R_max)
              R_max = R[i];
         }

     // Defining the subnetworks with the maximum AMP relative speed
     num_Rmax = 0;
     for (i = 1; i <= q; i++)
         {
          if (R[i] < R_max)
              x[i]--;
          else
              {
               list_Rmax[num_Rmax] = i;
               num_Rmax++;
              }
         }

     if (num_Rmax > 1)
         {
          // Defining the maximum number of floating AMPs
          if (numAMPs >= j + num_Rmax - 1)
              num_floatAMPs = num_Rmax;
          else
```

```
            num_floatAMPs = numAMPs - j + 1;

    for (m = 1; m <= num_floatAMPs; m++)
        {
         // Calculation of the number of distributions
         num_distr = fl(num_Rmax) / (fl(m) * fl(num_Rmax - m));
         if (num_distr > 1)
             {
              // Marking the subnetworks with the highest relative speed
              for (i = 1; i <= q; i++)
                  {
                   d[i] = 0;
                   for (k = 0; k < num_Rmax; k++)
                       if (i == list_Rmax[k])
                           d[i] = 1;
                  }

              // Printing the distribution
              cout << j + m - 1 << " AMPs have " << num_distr;
              cout << " distributions, i.e. combination of " << m;
              cout << " AMPs between " << num_Rmax << " subnetworks "<< endl;
              cout << "        ";
              for (i = 1; i <= q; i++)
                  if (d[i] == 0)
                       cout << "(" << x[i] << ")  ";
                  else
                       cout << "(" << x[i] - 1 << " or " << x[i] << ")  ";
              cout << "\n\n";
             }
         else
             {
              // Printing the distribution
              cout << j + num_Rmax - 1 << " AMPs:  ";
              for (i = 1; i <= q; i++)
                  cout << "(" << x[i] << ")  ";
              cout << "\n\n";
```

```
                }
            }
       j = j + num_Rmax;
      }
else
      {
       // Printing the distribution
       cout << j << " AMPs:  ";
       for (i = 1; i <= q; i++)
           cout << "(" << x[i] << ")  ";
       cout << "\n\n";
       j++;
      }
}

cout << "\n\nRepeat? (1 - Yes) ";
cin >> rep;
cout << "\n";
if (rep == 1)
     goto met1;

return 0;
}

// Calculation of x!
int fl (int x)
{
 int i, y;
 y = 1;

 if (x > 1)
     for (i = 1; i <= x; i++)
          y *= i;

 return y;
}
```

Chapter C

Calculation of AMP Distribution in Heterogeneous Networks

The chapter provides an example of calculation of 16 AMP distribution ($k = 16$) in heterogeneous networks using algorithm provided in Section 5.1.6. Let a system have $q = 5$ singleton subnetworks with the following available speeds:

$$S_1 = 5, \quad S_2 = 7, \quad S_3 = 10, \quad S_4 = 12, \quad S_5 = 15. \tag{C.1}$$

The calculations are as follows:

1. Substituting $k = 16,\ q = 5$ and available speeds from (C.1) in (5.10), i.e.

$$k_{i,j} = \left[\left[\frac{(2k+q)(S_i+S_j)}{2\sum_{m=1}^{q} S_m} - 1\right]\right],$$

we find $k_{i,j}$ for all $i, j = [1, 5]$ where $i \neq j$:

$$k_{i,j} = \left[\left[\frac{37(S_i+S_j)}{98} - 1\right]\right]. \tag{C.2}$$

$k_{1,2} = [\![3.53]\!] = 4, \quad k_{1,3} = [\![4.66]\!] = 5, \quad k_{1,4} = [\![5.41]\!] = 5, \quad k_{1,5} = [\![6.55]\!] = 7,$
$k_{2,3} = [\![5.41]\!] = 5, \quad k_{2,4} = [\![6.17]\!] = 6, \quad k_{3,5} = [\![7.3]\!] = 7,$
$k_{3,4} = [\![7.3]\!] = 7, \quad k_{3,5} = [\![8.43]\!] = 8, \quad k_{4,5} = [\![9.19]\!] = 9.$

2. Then for each $k_{i,j}$ we find x_i and x_j using (5.4), i.e.

$$\begin{cases} x_i = \left[\!\left[\dfrac{S_i(k_{i,j}+1)}{S_i+S_j} - \dfrac{1}{2} \right]\!\right], \\ \\ x_j = \left[\!\left[\dfrac{S_j(k_{i,j}+1)}{S_i+S_j} - \dfrac{1}{2} \right]\!\right]. \end{cases}$$

$$\begin{aligned}
k_{1,2}: &\quad x_1 = [\![1.58]\!] = 2 &&\quad x_2 = [\![2.41]\!] = 2 \\
k_{1,3}: &\quad x_1 = [\![1.5]\!] = 1 \textit{ or } 2 &&\quad x_3 = [\![3.5]\!] = 3 \textit{ or } 4 \\
k_{1,4}: &\quad x_1 = [\![1.26]\!] = 1 &&\quad x_4 = [\![3.73]\!] = 4 \\
k_{1,5}: &\quad x_1 = [\![1.5]\!] = 1 \textit{ or } 2 &&\quad x_5 = [\![5.5]\!] = 5 \textit{ or } 6 \\
k_{2,3}: &\quad x_2 = [\![1.97]\!] = 2 &&\quad x_3 = [\![3.02]\!] = 3 \\
k_{2,4}: &\quad x_2 = [\![2.07]\!] = 2 &&\quad x_4 = [\![3.92]\!] = 4 \\
k_{2,5}: &\quad x_2 = [\![2.04]\!] = 2 &&\quad x_5 = [\![4.95]\!] = 5 \\
k_{3,4}: &\quad x_3 = [\![3.13]\!] = 3 &&\quad x_4 = [\![3.86]\!] = 4 \\
k_{3,5}: &\quad x_3 = [\![3.1]\!] = 3 &&\quad x_5 = [\![4.9]\!] = 5 \\
k_{4,5}: &\quad x_4 = [\![3.94]\!] = 4 &&\quad x_5 = [\![5.05]\!] = 5
\end{aligned}$$

3. From the above calculations

$$x_1 = 1 \textit{ or } 2, \quad x_2 = 2, \quad x_3 = 3 \textit{ or } 4, \quad x_4 = 4, \quad x_5 = 5 \textit{ or } 6.$$

Possible distributions are presented in Table C.1. Up and down arrows indicate the rounding of .5 to the upper and lower values respectively. According to the total

	Distr. 1	Distr. 2	Distr. 3	Distr. 4	Distr. 5	Distr. 6	Distr. 7	Distr. 8
x_1	1 ↓	1 ↓	1 ↓	1 ↓	2 ↑	2 ↑	2 ↑	2 ↑
x_2	2	2	2	2	2	2	2	2
x_3	3 ↓	3 ↓	4 ↑	4 ↑	3 ↓	3 ↓	4 ↑	4 ↑
x_4	4	4	4	4	4	4	4	4
x_5	5 ↓	6 ↑	5 ↓	6 ↑	5 ↓	6 ↑	5 ↓	6 ↑
Total x	15	16	16	17	16	17	17	18
	×	√	√	×	√	×	×	×

Table C.1: List of Distributions

number of AMPs x in each distribution 16 AMPs may only result in distributions 2, 3, and 5:

$$\begin{array}{lllll} x_1 = 1 \downarrow & x_2 = 2 & x_3 = 3 \downarrow & x_4 = 4 & x_5 = 6 \uparrow \\ x_1 = 1 \downarrow & x_2 = 2 & x_3 = 4 \uparrow & x_4 = 4 & x_5 = 5 \downarrow \\ x_1 = 2 \uparrow & x_2 = 2 & x_3 = 3 \downarrow & x_4 = 4 & x_5 = 5 \downarrow \end{array} \tag{C.3}$$

4. For the three distributions we check inequality (5.2) that is strictly larger for all pairs i and j, i.e.

$$\frac{S_i}{x_i} > \frac{S_j}{x_j + 1}, \tag{C.4}$$

except for the pairs where $x_i \uparrow$ and $x_j \downarrow$. In this case (5.2) is as follows:

$$\frac{S_i}{x_i} = \frac{S_j}{x_j + 1}. \tag{C.5}$$

The analysis shows that all distributions from (C.3) satisfy either condition (C.4) or (C.5). Therefore, *optimally balanced distribution of 16 AMPs may only result in one of the three distributions presented in (C.3).*

Chapter D

Theoretical Analysis of Redundant Movements

D.1 cNAMP Termination at the Root Location in an Optimally Balanced Homogeneous Network

Assume that a cNAMP terminates at the root location in an optimal balanced state. Then the root location has $\left[\left[\frac{2fk-f-3}{2(f+1)}\right]\right]$ cNAMPs, and non-root locations have $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs.

According to (5.14) and (2.3), before a cNAMP movement to the root location from a non-root location, the cNAMP execution time on the non-root location is

$$T_h = \frac{W_r}{S} \cdot \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]. \tag{D.1}$$

After the cNAMP movement the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs and execution time on it becomes

$$T_n = \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]. \tag{D.2}$$

Thus, substituting (D.1) and (D.2) in (2.5), we get the following cNAMP movement condition:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right] > \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] + T_{comm}. \tag{D.3}$$

If condition (D.3) holds, then the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs, one non-root location has $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ cNAMPs, and the rest of the non-root locations have $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs. The system enters a near-optimal balanced state and cannot have any more movements.

So, *after a cNAMP termination at the root location in an optimally balanced homogeneous network, only one movement may occur for rebalancing and, hence, there is no greedy effect*.

D.2 cNAMP Termination in a Near-Optimally Balanced Homogeneous Network

In near-optimal balance a system has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs on the root location, and either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on non-root locations (Section 5.1.5).

Theorem 6. *The greedy effect causes at most one redundant movement, when a cNAMP terminates in a near-optimally balanced homogeneous network.*

Proof of Theorem 6 follows directly from Lemma 7.

Lemma 7. *A redundant movement occurs only in two cases: of a cNAMP termination in near-optimal balance on the root location which is discovered first by a cNAMP from a light location, and of a cNAMP termination in near-optimal balance on a light location which is discovered first by a cNAMP from the root location.*

Proof. The proof of Lemma 7 again proceeds by case analysis on the location where termination occurs, and where the first movement is initiated.

Termination at the Root Location. After the cNAMP termination the root location has $\left[\left[\frac{2fk-f-3}{2(f+1)}\right]\right]$ cNAMPs. The movement to the root location may occur either from a light location that has $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ cNAMPs or from a heavy location that has $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs.

1. *cNAMP movement from a light location*. From (5.14) and (2.3), before the cNAMP movement to the root location the cNAMP execution time on the light location is

$$T_h = \frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]. \tag{D.4}$$

After the cNAMP movement the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs and execution time there becomes

$$T_n = \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]. \tag{D.5}$$

Thus, substituting (D.4) and (D.5) in (2.5) the cNAMP movement condition is:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right] > \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] + T_{comm}. \tag{D.6}$$

If the system satisfies condition (D.6), then after the cNAMP movement from the light to the root location, the system has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs on the root location, $\left[\left[\frac{2k-5f-3}{2(f+1)}\right]\right]$ cNAMPs on one non-root location, $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ and $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on the remaining non-root locations. According to the minimum difference criterion, there cannot be movements between either light and very light locations or heavy and light locations. As condition (D.6) is satisfied, a cNAMP will not transfer between the root and the very light locations.

However, a cNAMP might move from a heavy location that has $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs to the very light location that has $\left[\left[\frac{2k-5f-3}{2(f+1)}\right]\right]$ cNAMPs. Before the cNAMP movement execution time on a heavy location is as follows:

$$T_h = \frac{W_r}{S} \cdot \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]. \tag{D.7}$$

After the cNAMP movement the very light location becomes a light location with $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ cNAMPs and execution time on it becomes

$$T_n = \frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]. \tag{D.8}$$

Substituting (D.7) and (D.8) in (2.5), we get the following condition:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right] > \frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right] + T_{comm}. \tag{D.9}$$

If the system satisfies condition (D.9), then the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs, and the remaining locations have either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs, i.e. the system is in a balanced state. This means that *there can be at most one redundant movement, when a cNAMP terminates at the root location in a near-optimal balanced state. The movement occurs when a cNAMP from a light location discovers the opportunity to move first*.

2. *cNAMP movement from a heavy location*. Before the cNAMP movement the heavy location has $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs, and its execution time is given by (D.7). After the movement the root location has $\left[\left[\frac{f(2k+1)-1}{2(f+1)}\right]\right]$ cNAMPs, and its execution time becomes (D.5). Substituting (D.7) and (D.5) in (2.5) we get the following cNAMP movement condition:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right] > \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] + T_{comm}. \tag{D.10}$$

If condition (D.10) holds, then the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs, and non-root locations have either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs. The system enters a near-optimal balanced state and cannot have any other movements. Therefore, *there is no greedy effect in a near-optimal balanced state, when a cNAMP terminates at the root location, and a cNAMP from a heavy location discovers a better opportunity for execution first*.

The above analysis leads to the conclusion that after a cNAMP termination at the root location, the system may rebalance to improve execution parameters. In case of rebalancing, the system might make one movement. The second movement may occur only if condition (D.6) holds in a near-optimal balanced state and a cNAMP from a light location discovers a better execution opportunity first. However, *besides the movement for rebalancing, there cannot be more than one redundant movement*.

Termination at a Light Location. After the cNAMP termination, the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs, one non-root location has $\left[\left[\frac{2k-5f-3}{2(f+1)}\right]\right]$ cNAMPs (i.e. very light location), and the remaining non-root locations have either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ cNAMPs (i.e. light locations) or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs (i.e. heavy locations). According to the minimum difference criterion there cannot be any movements between light and very light locations.

Thus, a cNAMP movement to the very light location might occur from the root location or a heavy location.

1. *cNAMP movement from the root location*. Before the movement the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs, and cNAMP execution time on the root location is

$$T_h = \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]. \tag{D.11}$$

   After the movement the very light location becomes a light location. The execution time on a light location is given by (D.8). Substituting (D.11) and (D.8) in (2.5), we get the following condition of the cNAMP movement from the root to the very light location:

$$\frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] > \frac{W_r}{S} \cdot \left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right] + T_{comm}. \tag{D.12}$$

   If condition (D.12) holds then the system has $\left[\left[\frac{2fk-f-3}{2(f+1)}\right]\right]$ cNAMPs on the root location, $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on heavy locations, and $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ on light locations. Because of condition (D.12) there can not be movements from a light

location to the root location. However, a cNAMP might move from a heavy location to the root location. Before the movement the cNAMP execution time on a heavy location is given by (D.7), and after the movement the cNAMP execution time on the root location is given by (D.5). Substituting (D.7) and (D.5) in (2.5) gives the following cNAMP movement condition:

$$\frac{W_r}{S} \cdot \left[\left[\frac{2k-f+1}{2(f+1)}\right]\right] > \frac{W_r}{f \cdot S} \cdot \left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right] + T_{comm}. \tag{D.13}$$

If condition (D.13) holds the system enters a balanced state where the root location has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs and non-root locations have either $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs or $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ cNAMPs. Therefore, *there can be at most one redundant movement, when a cNAMP terminates at a light location in a near-optimal balanced state, and a cNAMP from the root location discovers the opportunity to move first*.

2. *cNAMP movement from a heavy location*. Before the movement a heavy location has $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs and execution time on it is given by (D.7). After the cNAMP movement, the very light location becomes a light location, execution time on a light location is given by (D.8). Substituting (D.7) and (D.8) in (2.5), gives condition (D.9). If condition (D.9) holds, then after the movement the system has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs on the root location, and either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on non-root locations, i.e. the system enters a balanced state. Thus, *when a cNAMP terminates at a light location, and then a cNAMP from a heavy location discovers the opportunity to move first the system might make only one movement to rebalance*.

The analysis shows that after a cNAMP termination at a light location, the system might rebalance to improve execution parameters. In case of rebalancing, the system makes one movement. The second movement might occur only if condition (D.12) holds in a near-optimal balanced state and a cNAMP from the root location discovers a better execution opportunity first. However, *besides the movement for rebalancing, there cannot be more than one redundant movement*.

cNAMP Termination at a Heavy Location. After the cNAMP termination at a heavy location, the system has $\left[\left[\frac{2fk+f-1}{2(f+1)}\right]\right]$ cNAMPs on the root location, and either $\left[\left[\frac{2k-3f-1}{2(f+1)}\right]\right]$ or $\left[\left[\frac{2k-f+1}{2(f+1)}\right]\right]$ cNAMPs on non-root locations. The system enter a balanced state, and *there is no greedy effect in this case*. □

D.3 Proof of Lemma 10

Lemma 10. *A system makes at most $q-1$ redundant movements after a cNAMP termination from a near-optimally balanced heterogeneous network.*

Proof. The proof of Lemma 10 again proceed by case analysis on the location where termination occurs, and the location where the first movement is initiated. From Section 5.1.4 in a near-optimal balanced state all subnetworks except for a few are optimally balanced. Thus, a cNAMP can terminate from an optimally or a near-optimally balanced subnetwork.

cNAMP terminates in an optimally balanced subnetwork. Let a heterogeneous network have $q-q'$ optimally balanced subnetworks numbered in the descending order of their cNAMP relative speeds starting from 1, and $1 \leqslant q' < q$ near-optimally balanced subnetworks (subnetworks q, $q-1$, ...). As heavy locations of near-optimally balanced subnetworks always have the slowest cNAMP relative speed in near-optimal balance, we can write the following:

$$\frac{S_1}{x_1} > \frac{S_2}{x_2} \geqslant ... \geqslant \frac{S_q}{x_q}$$

where S_i is the available speed and x_i is an optimal number of cNAMPs on a location of subnetwork i. Following the proof of Lemma 9 the maximum number of redundant movements occurs when a cNAMP terminates from an optimally balanced subnetwork with the highest cNAMP relative speed, (i.e. cNAMP relative speed should also be larger than cNAMP relative speeds on light locations of near-optimally balanced subnetworks)

and cNAMPs discover a better opportunity for execution in the descending order of cNAMP relative speeds. If these two conditions hold then there can be at most $q - 1$ redundant movements.

cNAMP termination from a near-optimally balanced subnetwork. A cNAMP can terminate from a heavy or a light location. If a cNAMP terminates *from a heavy location*, the system stays either in near-optimal balance or enters another optimal balance, and there cannot be any more movements. In case of a cNAMP termination *from a light location* a movement is possible from:

- *a heavy location of a near-optimally balanced subnetwork*. Then the system either stays in near-optimal balance or enters another optimal balance, and there are no redundant movements.
- *a location of an optimally balanced subnetwork*. Here, applying analysis made for a cNAMP termination from an optimally balanced subnetwork, we find that the maximum number of redundant movements is equal to $q - 1$. This happens when a cNAMP terminates from a light location that has the highest cNAMP relative speed and then other cNAMPs discover better opportunity for execution in the descending order of cNAMP relative speeds.
- *a light location of another near-optimally balanced subnetwork.* The maximum number of redundant movements in this case is again $q - 1$. For these $q - 1$ redundant movements to occur the following conditions must be satisfied:
  - a network should have at list two near-optimally balanced subnetworks, i.e.
  $$\frac{S_q}{x_q} = \frac{S_{q-1}}{x_{q-1}} = ...$$
  - a cNAMP should terminate from a light location that has the maximum cNAMP relative speed.

– other cNAMPs should discover opportunity to move in a descending order of their relative speeds.

□

D.4 Probability of $q-2$ Redundant Movements after a cNAMP Termination from Optimal Balance

The probability, P, of $q-2$ redundant movements after cNAMP termination from an optimally balanced heterogeneous network of q subnetworks is a product of the probabilities that:

1. a cNAMP terminates from a location with the highest cNAMP relative speed, P_{termRh};
2. cNAMPs discover better opportunities for execution in the descending order of cNAMP relative speeds, P_{des}, i.e.

$$P = P_{termRh} \cdot P_{des}. \tag{D.14}$$

The calculation presented below is similar to the calculation of the probability for homogeneous networks discussed in Section 5.2.1 (page 98). From (5.23) the rate of cNAMP termination, ν_i, at a location of subnetwork i is

$$\nu_i = \frac{S_i}{W \cdot x_i} = \frac{R_i}{W}. \tag{D.15}$$

Assume that each location of subnetwork i has x_i cNAMPs in the optimally balanced state, the total number of locations in subnetwork i is N_i, and $R_1 > R_2 > ... > R_q$. Then the probability of cNAMP termination at a location with the highest cNAMP relative speed is

$$P_{termRh} = \frac{\nu_1 N_1}{\sum_{i=1}^{q} \nu_i N_i}$$

that can be written as

$$P_{termRh} = \frac{R_1 N_1}{\sum_{i=1}^{q} R_i N_i}. \tag{D.16}$$

The probability of a cNAMP information discovery from locations in a descending order of cNAMP relative speeds is

$$P_{des} = \frac{x_2 N_2}{\sum_{i=2}^{q} x_i N_i} \cdot \frac{x_3 N_3}{\sum_{i=3}^{q} x_i N_i} \cdot \ldots \cdot \frac{x_{q-1} N_{q-1}}{\sum_{i=q-1}^{q} x_i N_i}$$

or

$$P_{des} = \prod_{j=2}^{q-1} \frac{x_j N_j}{\sum_{i=j}^{q} x_i N_i}. \tag{D.17}$$

Substituting (D.16) and (D.17) in (D.14) we get the following probability of $q-2$ redundant movements after a cNAMP termination in an optimally balanced heterogeneous network:

$$P = \frac{R_1 N_1}{\sum_{i=1}^{q} R_i N_i} \cdot \prod_{j=2}^{q-1} \frac{x_j N_j}{\sum_{i=j}^{q} (x_i N_i)}. \tag{D.18}$$

Equation (D.18) shows that the probability of the maximum number of movements depends on the following parameters: available speeds of locations, and the number of cNAMPs, locations and subnetworks. The calculation does not take into account cases when locations of a subnetwork have no cNAMPs. The calculations of probability (D.18) are made on the basis of the following scenario:

- *Scenario 4*: There are six subnetworks, $Subnet1 - Subnet6$, locations of which have 1793 MHz, 2168 MHz, 3193 MHz, 3608 MHz, 4377 MHz and 4512 MHz available speeds respectively. $Subnet1$ is singleton, the sizes of other subnetworks can vary.

In the first experiment k is taken equal to 50, subnetworks $Subnet2-Subnet6$ have seven location each. The experiment varies the number of subnetworks from 3 to 6. *In all cases the probability does not exceed 5%*; and *the decrease of probability with the increase of*

the number of subnetworks is observed. Thus, for further experiments the minimum number of subnetworks, $q = 3$, is used to investigate the maximum probability.

In the second experiment subnetworks $Subnet2$ and $Subnet3$ have seven locations each, and the number of cNAMPs, k, varies from 1 to 500. The results show that *the number of cNAMPs has no significant effect on the value of the probability*.

Evidently, the probability increases when the subnetwork with the highest cNAMP relative speed has the largest number of locations. As locations that have the highest cNAMP relative speed in a particular optimal balanced state are not known in advance, the numbers of locations in subnetworks $Subnet2$ and $Subnet3$ are varied from 2 to 100 simultaneously. The experiment is conducted for $k = 100$ cNAMPs. The observations show that *the increase in the number of locations results in the increase of the probability*. The highest value of the probability is 12% in a network where each subnetwork has 100 locations.

The fourth experiment examines the probability depending on the difference between available speeds of locations. The experiment is conducted using the following parameters: $Subnetwork2$ and $Subnetwork3$ have 100 locations each, $k = 100$ cNAMPs, and the available speed of $Subnetwork3$ varies from 34 MHz to 3460 MHz. The result analysis shows that *the difference between location available speeds has no significant effect on the probability*.

Therefore, the probability increases as the number of subnetworks decreases, and also increases as the number of locations in subnetworks increases. The number of cNAMPs and the difference between available speeds show no significant impact. To estimate the probability an experiment with the following parameters is conducted: a network has six subnetworks where $Subnetwork1$ has one location, $Subnetwork2 - Subnetwork6$ have 100 locations each. The number of cNAMPs, k, varies from 290 to 310. The available speeds of all subnetworks are presented in scenario 4, except the available speed of $Subnetwork3$ that varies from 34 MHz to 3460 MHz with 1 MHz step. The distribution of probability (D.18) is presented in Tables D.1 and D.2.

Probability, %	Number of Subnetworks			
	3	4	5	6
$0 < P < 10$	32116	41750	45759	45759
$10 \leqslant P < 20$	113	3892	0	0
$20 \leqslant P < 30$	4255	96	0	0
$30 \leqslant P < 40$	5962	21	0	0
$40 \leqslant P < 50$	2989	0	0	0
$50 \leqslant P < 60$	324	0	0	0

Table D.1: Probability Distribution of the Maximum Number of Movement

The first column in Tables D.1 and D.2 indicates the ranges of the probability values. The remaining columns show the number of cases with a particular probability in networks of three to six subnetworks. Table D.1 gives the number of cases with 10% step probability. The majority of the cases have probability less than 10%. Table D.2 shows that in these 10% in a heterogeneous network of three subnetworks 70% cases have probability less than 1%. The probability of 28% of cases varies between 20% and 50%. The maximum values of the probability occur when cNAMPs on the root location have the slowest cNAMP relative speed.

Table D.3 gives the maximum and the minimum values of the probability when a network has up to six subnetworks. Experiments show that for a network of three subnetworks the maximum value of the probability of $q - 2$ redundant movement is 55%, and for a network of six subnetworks the probability does not exceed 5%. The above leads to the conclusion that *the median probability of* $q - 2$ *redundant movements after a cNAMP termination from optimally balanced heterogeneous network does not exceed 1%.*

Probability, %	Number of Subnetworks			
	3	4	5	6
$0 < P < 1$	31796	33922	37269	45437
$1 \leqslant P < 5$	208	697	8217	322
$5 \leqslant P < 10$	112	7131	273	0

Table D.2: Distribution in the First 10% of Table D.1

Probability	Number of Subnetworks			
	3	4	5	6
P_{max}	55%	37%	7%	4.9%
P_{min}	$4 \cdot 10^{-7}\%$	$3.5 \cdot 10^{-9}\%$	$4.3 \cdot 10^{-11}\%$	$5.2 \cdot 10^{-14}\%$

Table D.3: Maximum and Minimum Values of $q-2$ Redundant Movement Probability

D.5 Probability of $q-1$ Redundant Movements after a cNAMP Termination from Near-Optimal Balance

After cNAMP termination from near-optimally balanced heterogeneous network of q subnetworks the remaining cNAMPs might make at most $q-1$ redundant movements to rebalance (Lemma 10). The probability of these $q-1$ redundant movements is the sum of probabilities of three independent events, i.e.

$$P = P_1 + P_2 + P_3. \tag{D.19}$$

The conditions for each event and their probabilities are as follows:

1. Probability P_1 is the product of the probabilities that:
   - a cNAMP terminates from an optimally balanced subnetwork that has the highest cNAMP relative speed, P_{termRh};
   - cNAMPs discover a better opportunity for execution in the descending order of their relative speeds, P_{des1}.

   $$P_1 = P_{termRh} \cdot P_{des}. \tag{D.20}$$

2. Probability P_2 is the product of the following probabilities:
   - a cNAMP terminates at a light location of a near-optimally balanced subnetwork that has the highest cNAMP relative speed, P_{termRl};

- cNAMPs discover a better opportunity for execution in the descending order of their relative speeds, P_{des2}.

$$P_2 = P_{termRl} \cdot P_{des2}. \tag{D.21}$$

3. Probability P_3 is the product of the probabilities that:

   - the system has at least two near-optimally balanced subnetworks, P_{near2};
   - a cNAMP terminates at a light location of a near-optimally balanced subnetwork that has the highest cNAMP relative speed, $P_{termRl2}$;
   - cNAMPs discover a better opportunity for execution in the descending order of their relative speeds, P_{des3}.

$$P_3 = P_{near2} \cdot P_{termRl2} \cdot P_{des3}. \tag{D.22}$$

Because the number of cNAMPs is an integer and the near-optimally balanced subnetworks depend on the number of cNAMPs, it is difficult to estimate mean and maximum values of probability (D.19). However, experiments on the basis of scenario 4 show that the probability is less than 30%, and it rapidly decreases as the number of subnetworks increases.

Bibliography

[AcbraUBP11] USC/ISI A collaboration between researchers at UC Berkeley, LBL and Xerox PARC. *The ns-2 Manual*. http://www.isi.edu/nsnam/ns/ns-documentation.html, 2011.

[ACL00] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *STOC '00: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 171–180, New York, NY, USA, 2000. ACM.

[AD96] Jong Suk Ahn and Peter B. Danzig. Packet network simulation: Speedup and accuracy versus timing granularity. *IEEE/ACM Trans. Netw.*, 4(5):743–757, 1996.

[ama11] Amazon elastic compute cloud (Amazon EC2), 2011. http://www.aws.amazon.com/ec2.

[Ant05] R.J. Anthony. Engineering emergence for cluster configuration. *Journal of Systemics, Cybernetics and Informatics*, 3:17–26, 2005.

[Bar04] Rimon Barr. *SWANS - Scalable Wireless Ad hoc Network Simulator. User Guide*. Cornell Research Foundation, http://www.jist.ece.cornell.edu, 2004.

[BMH+02] C. J. Bovy, H. T. Metrodimedjo, G. Hooghiemstra, H. Uijterwaal, and P. Van Mieghem. Analysis of end-to-end delay measurements in Internet. In *PAM '02: Proceedings of the Passive and Active Measurement Workshop*, 2002.

[Boi06] André Rauber Du Bois. *Mobile Computation in Purely Functional Language*. PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK, 2006.

[BPZ96] Martin Backschat, Alexander Pfaffinger, and Christoph Zenger. Economic-based dynamic load distribution in large workstation networks. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing*, volume 2, pages 631–634, London, UK, 1996. Springer-Verlag.

[BT02] Tian Bu and Don Towsley. On distinguishing between Internet power law topology generators. In *INFOCOM '02: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communication Societies*, volume 2, pages 638–647. IEEE Computer Society, 2002.

[BTL05] A. Rauber Du Bois, P. Trinder, and H.-W. Loidl. mHaskell: Mobile computation in a purely functional language. *Journal of Universal Computer Science*, 11(7):1234–1254, 2005.

[Car99] Luca Cardelli. Mobility and security. In *Proceedings of the NATO Advanced Study Institute of on Foundation of Secure Computation*, pages 3–37, Marktoberdorf, Germany, August 1999. IOS Press.

[Cas01] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 430–437, 2001.

[CB04] Arjav J. Chakravarti and Gerald Baumgartner. Self-organizing scheduling on the organic Grid. *Int. Journal of High Performance Computing Applications*, 20:115–130, 2004.

[CDZ97] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modelling Internet topology. *IEEE Communications Magazine*, 35:160–163, 1997.

[CF11] Robert M. Cubert and Paul Fishwick. *Sim++. Version 1.0*. http://www.cise.ufl.edu/ fishwick/simpack/simpack.html, 2011.

[CHLE80] W. W. Chu, L. J. Holloway, Min-Tsung Lan, and K. Efe. Task allocation in distributed data processing. *Computer*, 13(11):57–69, 1980.

[CK87] T. L. Casavant and J. G. Kuhl. Analysis of three dynamic distributed load-balancing strategies with varying global information requirements. In *DCS '87: Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 185–192, New York, USA, 1987. IEEE Press.

[CK88] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, 1988.

[CK06] Mark Crovella and Balachander Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2006.

[CKPT09] Natalia Chechina, Peter King, Rob Pooley, and Phil Trinder. Simulating autonomous mobile programs on networks. In *PGNet '09: Proceedings of the 10th Annual Conference on the Convergence of Telecommunications, Networking and Broadcasting*, pages 201–206, Liverpool, UK, 2009. Liverpool John Moores University.

[CKT10] Natalia Chechina, Peter King, and Phil Trinder. Using negotiation to reduce redundant autonomous mobile program movements. In *IAT '10: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 343–346, Toronto, Canada, 2010. IEEE Computer Society.

[CKT11] Natalia Chechina, Peter King, and Phil Trinder. Redundant movements in autonomous mobility: Experimental and theoretical analysis. *Journal of Parallel and Distributed Computing*, 71(10):1278–1292, 2011.

[CLHZ97] Wentong Cai, Bu-Sung Lee, Alfred Heng, and Li Zhu. A simulation study of dynamic load balancing for network-based parallel processing. In *ISPAN '97: Proceedings of the 1997 International Symposium on*

Parallel Architectures and Networks, pages 383–389, Washington, DC, USA, 1997. IEEE Computer Society.

[Cyb89] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279–301, 1989.

[Den07] X. Y. Deng. *Cost Driven Autonomous Mobility*. PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK, 2007.

[Dev87] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole Publishing Company, California, USA, second edition, 1987.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19:1–12, 1989.

[DL93] Matthew Doar and Ian Leslie. How bad is naive multicast routing. In *INFOCOM '93: Proceedings of the Twelfth Annual Joint Conference of the IEEE Computer Societies*, volume 1, pages 82–89, San Francisco, CA, USA, 1993. IEEE Computer Society.

[DMT10] X. Y. Deng, G. J. Michaelson, and P. W. Trinder. Cost-driven autonomous mobility. *Computer Languages Systems and Structures*, 36(1):34–59, 2010.

[Doa96] Matthew B. Doar. A better model for generating test networks. In *GLOBECOM '96: Proceedings of the Global Communication Conference*, pages 86–93. IEEE Computer Society, 1996.

[DTM06] X. Y. Deng, P. W. Trinder, and G. J. Michaelson. Autonomous mobile programs. In *IAT '06: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 177–186, Washington, DC, USA, 2006. IEEE Computer Society.

[EAEB97] Aly E. El-Abd and Mohamed I. El-Bendary. A neural network approach for dynamic load balancing in homogeneous distributed systems. In *HICSS '97: Proceedings of the 30th Hawaii International Conference on System Sciences*, volume 1, pages 628–629, Washington, DC, USA, 1997. IEEE Computer Society.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM '99: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 251–262, New York, NY, USA, 1999. ACM.

[FML+03] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely, and Christophe Diot. Packet-level traffic measurements from the sprint IP backbone. *IEEE Netw.*, 17(6):6–16, 2003.

[FPV98] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Trans. Softw. Eng.*, 24(5):342–361, 1998.

[FT91] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, USA, 1991.

[GA91] Arif Ghafoor and Ishfaq Ahmad. An efficient model of dynamic task scheduling for distributed systems. In *COMPSAC '90: Proceedings of the Fourteenth Annual International Computer Software and Applications Conference*, pages 442–447. IEEE Computer Society Press, 1991.

[Gar11] Ryan Gardiner. *Autonomous Cloud Brokering*. Final year dissertation, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK, 2011.

[GGK+01] Fotis Georgatos, Florian Gruber, Daniel Karrenberg, Mark Santcroos, Ana Susanj, Henk Uijterwaal, and Ren Wilhelm. Providing active meas-

urements as a regular service for ISPs. In *Proceedings of Passive & Active Measurement (PAM)*, 2001.

[GHCN99] R. Ghanea-Hercock, J. C. Collis, and D. T. Ndumu. Co-operating mobile agents for distributed parallel processing. In *AGENTS '99: Proceedings of the Third Annual Conference on Autonomous Agents*, pages 398–399, New York, NY, USA, 1999. ACM.

[GR03] Christos Georgousopoulos and Omer F. Rana. Combining state and model-based approaches for mobile agent load balancing. In *SAC '03: Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 878–885, New York, NY, USA, 2003. ACM.

[Hal92] David Lee Hall. *Mathematical Techniques in Multisensor Data Fusion*. Artech House, Inc., Norwood, MA, USA, 1992.

[HLMV87] S. H. Hosseini, B. E. Litow, M. I. Malkawi, and K. Vairavan. Distributed algorithms for load balancing in very large homogeneous systems. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology*, ACM '87, pages 397–404, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[HM01] G. Hooghiemstra and P. Van Mieghem. Delay distributions on fixed Internet paths. Technical Report 20011031, Delft University of Technology, Delft, Netherlands, 2001.

[HMH07] Markus C. Huebscher, Julie A. McCann, and Asher Hoskins. Context as autonomic intelligence in a ubiquitous computing environment. *International Journal of Internet Protocol Technology*, 2:30–39, 2007.

[HY00] Masatomo Hashimoto and Akinori Yonezawa. MobileML: A programming language for mobile computation. In *Coordination Languages and Models*, volume 1906 of *Lecture Notes in Computer Science*, pages 557–593. Springer Berlin / Heidelberg, 2000.

[IKKW02] Torsten Illmann, Tilman Krueger, Frank Kargl, and Michael Weber. Transparent migration of mobile agents using the Java platform debugger architecture. In *MA '01: Proceedings of the 5th International Conference on Mobile Agents*, pages 198–212, London, UK, 2002. Springer-Verlag.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KK80] L. Kleinrock and F. Kamoun. Optimal clustering structures for hierarchical topological network design of large computer networks. *Networks*, 10:221–248, 1980.

[KKP+04] Laxmikant V. Kale, Sameer Kumar, Mani Potnuru, Jayant DeSouza, and Sindhura Bandhakavi. Faucets: Efficient resource allocation on the computational Grid. In *ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing*, pages 396–405, Washington, DC, USA, 2004. IEEE Computer Society.

[Kle93] Lawrence A. Klein. *Sensor and Data Fusion Concepts and Applications*. Society of Photo-Optical Instrumentation Engineers (SPIE), Bellingham, WA, USA, 1993.

[Kuo85] H. Kuolin. *Allocation of Processors and Files for Load Balancing in Distributed Systems*. PhD thesis, University of California at Berkeley, USA, 1985.

[LE96] Robert Lloyd-Evance. *Wide Area Network Performance and Optimization*. Addison Wesley Longman, UK, 1996.

[LGY09] Wei-Zhou Lu, Wei-Xuan Gu, and Shun-Zheng Yu. One-way queuing delay measurement and its application on detecting DDoS attack. *J. Netw. Comput. Appl.*, 32(2):367–376, 2009.

[LK87] Frank C. H. Lin and Robert M. Keller. The gradient model load balancing method. *IEEE Trans. Softw. Eng.*, 13(1):32–38, 1987.

[LM82] Miron Livny and Myron Melman. Load balancing in homogeneous broadcast distributed systems. *SIGMETRICS Perform. Eval. Rev.*, 11:47–55, 1982.

[LR92] Hwa-Chun Lin and C. S. Raghavendra. A dynamic load balancing policy with a central job dispatcher (LBC). *IEEE Trans. Softw. Eng.*, 18(2):148–158, 1992.

[LRRV04] Arnaud Legrand, Hélène Renard, Yves Robert, and Frédéric Vivien. Mapping and load-balancing iterative computations. *IEEE Trans. Parallel Distrib. Syst.*, 15(6):546–558, 2004.

[LW00] C. P. Low and N. Wang. An efficient algorithm for group multicast routing with bandwidth reservation. *Computer Communications*, 23(18):1740–1746, 2000.

[LWZ05] Marin Litoiu, Murray Woodside, and Tao Zheng. Hierarchical model-based autonomic control of software systems. In *DEAS '05: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, pages 1–7, New York, NY, USA, 2005. ACM.

[Mac98] Lewis Mackenzie. *Communications and Networks*. McGraw-Hill, England, 1998.

[MB03] Daniel A. Menascé and Mohamed N. Bennani. On the use of performance models to design self-managing computer systems. In *Proceedings of the 2003 Computer Measurement Group Conference*, pages 7–12, Dallas, TX, USA, 2003.

[MDW99] Dejan Milojičić, Frederick Douglis, and Richard Wheeler. *Mobility: Processes, Computers and Agents*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[MI08] Naoki Miyata and Toru Ishida. Community-based load balancing for massively multi-agent systems. *Massively Multi-Agent Technology*, pages 28–42, 2008.

[MM10] Louis Mandel and Luc Maranget. *The JoCaml Language: Documentation and User's Manual*. http://jocaml.inria.fr/manual/index.html, 2010.

[MMB03] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Messor: Load-balancing through a swarm of autonomous agents. In *AP2PC'03: Proceedings of the 1st International Conference on Agents and Peer-to-Peer Computing*, pages 125–137, Berlin, Heidelberg, 2003. Springer-Verlag.

[MMSA+96] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: a fault-tolerant multicast group communication system. *Commun. ACM*, 39(4):54–63, 1996.

[MO04] Paulo Eduardo Merloti and June Of. Optimization algorithm inspired by biological ants and swarm behaviour. Technical report, San Diego State University, 2004.

[MP95] V. C. Marney-Petix. *Bridges, Routers, Gateways*. Numidia Press, 1995.

[MSC+86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. Rosenthal, and F. Donelson Smith. Andrew: A distributed personal computer environment. *Commun. ACM*, 29(3):184–201, 1986.

[MvS95] Henk Sips Maarten van Steen. *Computer and Network Organization*. Prentice Hall, UK, 1995.

[nct11] SimReal technology, 2011. http://nsl10.csie.nctu.edu.tw.

[NLJU02] D. Nicholson, C.M. Lloyd, S.J. Julier, and J.K. Uhlmann. Scalable sistributed data fusion. In *Proceedings of the Fifth International Conference on Information Fusion*, volume 1, pages 630–635, 2002.

[NNMNA+05] Lilian Norohan Nassif, Jose Marcos Nogueira, Mohamed Ahmed, Ahmed Karmouch, Roger Impey, and Flavio Vinicius de Andrade. Job completion prediction in Grid using distributed case-based reasoning. In *WETICE '05: Proceedings of the 14th IEEE International Workshop on Enabling Technologies*, pages 249–254, Washington, DC, USA, 2005. IEEE Computer Society.

[NSS+04] Makoto Nakamura, Junsuke Sembon, Yutake Sugawara, Tsuyoshi Itoh, Mary Inaba, and Kei Hiraki. End-node transmission rate control kind to intermediate routers - towards 10gbps era. In *PFLDnet '04: Proceedings of the Second International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[Nun92] Greg Nunemacher. *LAN Primer*. M&T Publishing, USA, second edition, 1992.

[NWG+09] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus open-source Cloud-computing system. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[NXG85] Lionel M. Ni, Chong-Wei Xu, and Thomas B. Gendreau. A distributed drafting algorithm for load balancing. *IEEE Trans. Softw. Eng.*, 11(10):1153–1161, 1985.

[OGP98] Wolfgang Obelöer, Claus Grewe, and Holger Pals. Load management with mobile agents. In *EUROMICRO '98: Proceedings of the 24th Conference on EUROMICRO*, pages 1005–1012, Washington, DC, USA, 1998. IEEE Computer Society.

[OO06] Natalia Olifer and Victor Olifer. *Computer Networks: Principles, Technologies and Protocols for Network Design*. Wiley, 2006.

[opn11] OPNET modeler. network simulation, 2011. http://www.opnet.com/solutions/network_rd/modeler.html.

[otc11] OTcl, 2011. http://www.otcl-tclcl.sourceforge.net/otcl.

[oxf08] *Dictionary of Computing*. Oxford University Press, 6 edition, 2008.

[PBKY02] Taesoon Park, Ilsoo Byun, Hyunjoo Kim, and H.Y. Yeom. The performance of checkpointing and replication schemes for fault tolerant mobile agent systems. In *RDS'02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, pages 256–261, Washington, DC, USA, 2002. IEEE Computer Society.

[PDR94] D.K. Panda and V.A. Dixit-Radiya. Message-ordering for wormhole-routed multiport systems with link contention and routing adaptivity. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 191–198, 1994.

[pen08] *Penguin Dictionary of Mathematics*. Penguin Books, 4 edition, 2008.

[Pin08] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3 edition, 2008.

[PV03] R. Percacci and A. Vespignani. Scale-free behaviour of the Internet global performance. *The European Physical Journal B*, 32(4):411–414, 2003.

[PZMH07] Himabindu Pucha, Ying Zhang, Z. Morley Mao, and Y. Charlie Hu. Understanding network delay changes caused by routing events. *SIGMETRICS Perform. Eval. Rev.*, 35(1):73–84, 2007.

[Rec10] Recursion Software, Inc., http://www.recursionsw.com/Products/voyager.html. *Voyager Technical Documentation*, 2010.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31:161–172, 2001.

[rip11] The RIPE Network Coordination Centre, 2011. http://www.ripe.net/.

[RLS+03] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured P2P systems. In M. Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 68–79. Springer Berlin / Heidelberg, 2003.

[RM90] Andrew Ross and Bruce McMillin. Experimental comparison of bidding and drafting load sharing protocols. In *Proceedings of the Fifth Distributed Memory Computing Conference*, volume 2, pages 968–974. IEEE Computer Society Press, 1990.

[RN04] Tiberiu Rotaru and Hans-Heinrich Nägeli. Dynamic load balancing by diffusion in heterogeneous systems. *J. Parallel Distrib. Comput.*, 64(4):481–497, 2004.

[Rot94] H. G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proceedings Computers & Digital Techniques*, 141(1):1–10, 1994.

[SBK04] Jaroslaw Sliwinski, Andrzej Beben, and Piotr Krawiec. EmPath: Tool to emulate packet transfer characteristics in IP network. *Traffic Monitoring and Analysis*, 6003/2010:46–58, 2004.

[SBK06] Tino Schlegel, Peter Braun, and Ryszard Kowalczyk. Towards autonomous mobile agents with emergent migration behaviour. In *AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 585–592, New York, NY, USA, 2006. ACM.

[SKA06] Jan Stender, Silvan Kaiser, and Sahin Albayrak. Mobility-based runtime load balancing in multi-agent systems. In *SEKE '06: Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering*, Reedwood City, CA, USA, 2006.

[SKH95] Behrooz A. Shirazi, Krishna M. Kavi, and Ali R. Hurson. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[SKL89] Kang G. Shin, C. M. Krishna, and Yann-Hang Lee. Optimal dynamic control of resources in a distributed system. *IEEE Trans. Softw. Eng.*, 15(10):1188–1198, 1989.

[Sof11] Erlang Software. Erlang: Traffic and queuing software, 2011. http://members.iinet.net.au/ clark.

[SS06] Angela B. Shiflet and George W. Shiflet. *Introduction to Computational Science. Modelling and Simulation for the Sciences*. Princeton University Press, New Jersey, USA, 2006.

[SSDNB95] J.A. Stankovic, M. Spuri, M. Di Natale, and G.C. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, 1995.

[SSY00] Tahakiro Sakamoto, Tatsurou Sekiguchi, and Akinori Yonezawa. Bytecode transformation for portable thread migration in Java. In *ASA/MA '00: Proceedings of the Second International Symposium on Agent Systems and Applications*, pages 16–28, London, UK, 2000. Springer-Verlag.

[Sta85] J. A. Stankovic. An application of Bayesian decision theory to decentralized control of job scheduling. *IEEE Trans. Comput.*, 34(2):117–130, February 1985.

[sta11] StACC - collaborative research in cloud computing, 2011. http://www.cs.st-andrews.ac.uk/stacc.

[STD02] Ming-Shan Su, K. Thulasiraman, and A. Das. A scalable on-line multi-level distributed network fault detection/monitoring system based on the SNMP protocol. In *GLOBECOM '02: Proceedings of the Global Telecommunications Conference*, volume 2, pages 1960–1964. IEEE Computer Society, 2002.

[Tan03] Andrew S. Tanenbaum. *Computer Networks*. Pearson Education Inc., New Jersey, USA, fourth edition, 2003.

[TF95] M. Takano and K. Fujita. Multilevel network management by means of system identification. In *Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communication Societies*, volume 2, pages 538–545, Washington, DC, USA, 1995. IEEE Computer Society.

[TMR97] K. Thompson, G. J. Miller, and Wilder R. Wide-area Internet traffic patterns and characteristics. *Network, IEEE*, 11(6):10–23, 1997.

[TT85] Asser N. Tantawi and Don Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32:445–465, April 1985.

[Var10] Andras Varga. *OMNeT++. User Manual. OMNeT++ version 4.0*. http://www.omnetpp.org/doc/omnetpp40/manual/usman.html, 2010.

[Wax91] Bernard M. Waxman. Routing of multipoint connections. *Broadband Switching: Architectures, Protocols, Design, and Analysis*, pages 347–352, 1991.

[WCS93] David C. M. Wood, Sean S. Coleman, and Michael F. Schwartz. Fremont: A system for discovering network characteristics and problems. In *USENIX '93: Proceedings of the USENIX Winter Conference*, pages 335–348, 1993.

[Wei99] Gerhard Weiss, editor. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Massachusetts, USA, 1999.

[WHH+92] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: a distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[WhS03] Ming Wu and Xian he Sun. A general self-adaptive task scheduling system for non-dedicated heterogeneous computing. In *CLUSTER '03: Proceedings of IEEE International Conference on Cluster Computing*, pages 354–361. IEEE Computer Society, 2003.

[WT98] Jerrell Watts and Stephen Taylor. A practical approach to dynamic load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 9(3):235–248, March 1998.

[WX99] Brian Wims and Cheng-Zhong Xu. TRAVELER: A mobile agent based infrastructure for wide area parallel computing. In *ASAMA '99: Proceedings of the First International Symposium on Agent Systems and Applications, and the Third International Symposium on Mobile Agents*, pages 258–259, Washington, DC, USA, 1999. IEEE Computer Society.

[YP98] Tse-Yu Yeh and Yale N. Patt. Alternative implementations of two-level adaptive branch prediction. In *ISCA '98: 25 Years of the International Simposia on Computer Architecture (selected papers)*, pages 451–461, New York, NY, USA, 1998. ACM.

[ZCB96] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM '96: Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, volume 2, pages 594–602. IEEE Computer Society, 1996.

[ZCBD04] A. Zietoun, Chen-Nee Chuah, S. Bhattacharyya, and C. Diot. An AS-level study of Internet path delay characteristics. In *GLOBECOM '04: Proceedings of the Global Telecommunications Conference*, volume 3, pages 1480–1484. IEEE Computer Society, 2004.

[Zho10] Shangqin Zhong. Secure multilevel management of large-scale network. In *GMC '10: Proceedings of the Global Mobile Congress*, pages 1–7. IEEE Computer Society, 2010.

[ZZWD93] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computer systems. *Softw. Pract. Exper.*, 23(12):1305–1336, 1993.