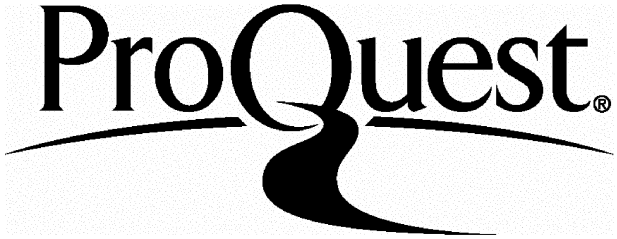
Load Sharing as a Power Management Strategy for Mobile Computers

Mazliza Othman

A dissertation submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
of the
University of London

Department of Computer Science
University College London

May 1999



Abstract

Load sharing has traditionally been used to improve system performance in distributed networks by transferring jobs from heavily loaded hosts to idle or lightly loaded hosts. Performance is improved by distributing workload more evenly among hosts, thus better utilising system resources.

This thesis investigates the use of load sharing for a different purpose, that is as a power management strategy for mobile computers. Since mobile computers operate on limited battery power, which is a scarce resource, and there is unlikely to be a vast improvement in battery capacity in the near future, it is vital that power utilisation is managed efficiently and economically.

The power management strategy proposed in this thesis is based on the concept of load sharing. The strategy attempts to reduce power consumption by the CPU, which is one of the components consuming a substantial amount of power, by off-loading computations from a mobile computer to a fixed host. A load sharing algorithm which selects suitable jobs for remote execution is proposed. When designing the algorithm, the inherent limitation of wireless networks must be taken into account. For example, low bandwidth means that communications delays are no longer negligible; sending and receiving messages must also be considered carefully as transmitting and receiving also consume a substantial amount of power. Consequently, when performing load sharing on wireless networks, more constraints have to be dealt with compared to when performing load sharing on fixed networks. In addition to reducing power consumption, transferring jobs for remote execution also gives users access to faster machines, thus improving response time.

This study identifies the conditions and factors which make job transfer a viable option. The results obtained show that under suitable conditions, load sharing can extend battery lifetime significantly. Since stability is an important concern when designing load sharing algorithms, this issue is also addressed by this study.

Acknowledgement

I would like to express my gratitude to people who have supported me throughout my postgraduate study:

- first of all, my parents, for their love and support. Where would I be without them!;
- my supervisor, Dr. Stephen Hailes, for his invaluable guidance and advice, and for being such a wonderful mentor;
- Dr Hale Cuss from the Language Centre who taught the 'Thesis Writing Course';
- Mr John Wilkes, from Hewlett Packard, for providing the hard disk trace data used in part of this study;
- fellow sufferers in room 208, Paul, Minseok and Cedric;
- Jungwon, Nadia and Kanta for 'depression outings', and for recommending good books to read;
- Faez, Gie, Anis and Norley, for putting up with my emails, moaning about my predicament;
- Raeis, for making sure I do not forget my mother tongue;
- and finally, Ronnie, for keeping me company while I was tapping away at the keyboard.

"And in the alternation of night and day,
and the provision (rain) that God sends down from the sky,
and revives therewith the earth after its death,
and in the turning about of the winds,
are signs for a people who understands."
Al-Quran 45:5

Table of Contents

1. Introduction 7
   - 1.1 Hypotheses 8
   - 1.2 Contributions 8
   - 1.3 Thesis Outline 9

2. Wireless Networks and Mobile Computing Issues 11
   - 2.1 Mobile Computing Applications and Services 12
   - 2.2 Limitations and Challenges of Wireless Networks 16
   - 2.3 Power Management Strategies 20
     - 2.3.1 Power Management Strategies for Hard Disks 22
     - 2.3.2 Power Efficient Communications 30
     - 2.3.3 Power Management Strategies for CPUs 33
   - 2.4 Conclusion 38

3. Load Distribution in Distributed Systems 40
   - 3.1 Load Sharing Policies 42
     - 3.1.1 Information and Location Policies 43
     - 3.1.2 Transfer Policy 47
     - 3.1.3 Job Selection Policy 48
   - 3.2 Effect of Communication Delays on Load Distribution 53
   - 3.3 Stability and Scalability 53
   - 3.4 Conclusion 55

4. Load Sharing in Wireless Networks 57
   - 4.1 An Approach to Load Sharing in Wireless Networks 57
   - 4.2 The Load Sharing Algorithm 59
   - 4.3 Trace Data 62
   - 4.4 Power Consumption 63
   - 4.5 Assumptions 64
   - 4.6 The Simulation Environment 65
     - 4.6.1 Entities 65
     - 4.6.2 System Parameters 68
     - 4.6.3 Experimental Design 70
     - 4.6.4 Performance Metrics 73
     - 4.6.5 Data Analysis 73
   - 4.7 Summary 74

5. Factors Influencing Load Sharing in Wireless Networks 75
   - 5.1 Bandwidth 75
   - 5.2 Processor Power of Mobile Computers 79
   - 5.3 CPU Utilisation 82
   - 5.4 History and ALS 83
   - 5.5 Combining Load Sharing with a Disk Spin-Down Strategy 85
   - 5.6 Conclusion 88

6. Stability and Scalability of the Load Sharing Algorithm 90
6.1 Backoff Algorithm 92
6.2 Slotted LS Algorithm 95
6.3 Delegating Job Transfer Requests 103
6.4 Conclusion 107
7. Emulation of Load Sharing on a Wireless LAN 108
7.1 Results of Emulation and Simulation 109
7.2 Conclusion 114
8. Future Work 115
8.1. Future Systems 115
8.2. Applicability of this Work 117
8.3. Future work 118
9. Conclusion 120
References 123

Table of Figures

3-1 Components of a load sharing algorithm 43
4-1 Entities and messages passed between them 66
5-1 Battery lifetime improvement vs. available bandwidth 77
5-2 Percentage of jobs transferred vs. available bandwidth 77
5-3 Communication delays vs. available bandwidth 78
5-4 Battery lifetime improvement vs. processor power 80
5-5 Response time improvement vs. processor power 81
5-6 Battery lifetime improvement vs. CPU utilisation 83
5-7 Performance of History and ALS compared to LS 85
5-8 Battery lifetime improvement combining load sharing with a disk spin-down strategy 87
6-1 Performance of LS vs. total users 91
6-2 Performance of load sharing degrades beyond no load sharing 91
6-3 Percentage of jobs transferred vs. total users 91
6-4 Percentage of rejected requests vs. total users 92
6-5 Battery lifetime improvement of LS and Backoff 93
6-6 Percentage of rejected requests of LS and Backoff 94
6-7 Response time improvement of LS and Backoff 94
6-8 Battery lifetime improvement for Slotted LS, Backoff and LS 99
6-9 Response time improvement of Slotted LS, Backoff and LS 100
6-10 Performance of LS with probes 104
6-11 Percentage of rejected requests of Probes and LS 104
6-12 Percentage of rejected requests of LS, *P=2* and Slotted Probe 105
6-13 Battery lifetime improvement for LS, *P=2* and Slotted Probe 105
6-14 Percentage of jobs transferred for LS, *P=2* and Slotted Probe 105
6-15 Response time improvement for LS, *P=2* and Slotted Probe 106
7-1 Battery lifetime improvement for emulation and simulation 110
7-2 Percentage of jobs transferred for emulation and simulation 111
7-3 Response time improvement for emulation and simulation 111
7-4 Emulation: battery lifetime improvement vs. CPU utilisation 112
7-5 Simulation: battery lifetime improvement vs. CPU utilisation 112

In loving memory, bapa.

Al-Fatihah.

1. Introduction

In recent years, the basic technology has become available to make wireless computing a reality. In addition, changing work patterns and an understanding of what technology might be able to provide have become business drivers pushing towards environments in which tetherless computing has a significant role to play. It is not, therefore, surprising that the research community have become increasingly interested in this area, driving it forward from the lower levels.

The technologists' goal is to produce systems which are practicable. There are several key characteristics of mobile systems which must be addressed in order to allow this to happen. Wireless links are relatively slow and unreliable; wireless hosts are resource poor relative to their fixed counterparts; mobile users may roam the globe, disconnecting from and reconnecting to the network arbitrarily; and mobile hosts must rely on battery power for significant periods of time. It is this latter problem that this dissertation seeks to address.

Under continuous use, a typical laptop battery lasts for approximately two to three hours. There has been relatively little in the way of innovation in battery technology over the past decade. In 1992, Sheng et al projected a 20% increase in battery capacity over a ten-year period [Shen92]. More recently, Cox stated that there is unlikely to be a 10x improvement in battery capacity [Cox95]. Since the power supply problem cannot easily be solved, the only practical solution to providing greater system availability is to reduce power consumption. There are several existing approaches to this problem already in use within laptops, generally based on powering down hardware components when they are not in use. However, studies based on such approaches have failed to consider how best to exploit the potential for power saving which can be derived from the existence of a communication link.

In this study, we adopt an approach based around the concept of load sharing, which has traditionally been used to improve job response time in distributed systems. It is sometimes forgotten that the question of load distribution is simply an instance of a more general class of resource allocation problems, with the objective of optimising job response time. Here, the problem is similarly to do with making the most of a resource, that resource being battery power and the optimisation constraints being to minimise the amount of power used. As in load distribution, the problem is non-trivial; there is a need to invest power in transmitting a job to the fixed network in order to receive benefit by allowing the CPU to be placed in doze mode. Again, as in load distribution, the benefit cannot be known precisely at the point of investment; the key is to estimate intelligently.

1.1 Hypotheses

Since the load sharing algorithm now operates in an environment which is much different from the environment previously studied, factors influencing the benefit gained from load sharing are also expected to be different. In this study, it is expected that:

- low bandwidth impedes load sharing as long delays might result in increased response time,
- mobile hosts with low processor power benefit from load sharing as that gives them access to faster machines on the fixed network,
- it is jobs with long execution time that brings significant benefit from transfers as they considerably reduce power consumption by the CPU,
- combining load sharing with another power management strategy will further extend battery lifetime,
- delegating transfer requests among fixed hosts will further improve performance as users are given access to other fixed hosts' spare capacity.

Experiments were carried out to verify these hypotheses, and determine if load sharing brings significant improvement in battery lifetime.

1.2 Contributions

The contributions of this thesis is listed below:

1. This study has successfully adopted a well-known concept in distributed systems to address a problem in mobile computing. It is possible to adopt load sharing for use in a mobile computing environment as a power management strategy, and load sharing was found to be a potentially effective power management strategy. On average, battery lifetime was extended by about 20% when load sharing was performed. In an experiment which combined load sharing with another power management strategy, battery lifetime was extended by an average of 33%.
2. This study identified the necessary modifications required in order to implement load sharing in a mobile computing environment, taking into considerations the constraints of wireless networks and the objective to be achieved.
3. In addition to establishing that load sharing does extend battery lifetime significantly, the findings also identify factors which are favourable, and which may impede load sharing. Among the finding of this study is that low bandwidth does not always impede load sharing; subject to a constraint, it is possible to transfer jobs even at low bandwidth. Load

sharing was also found to be useful in giving mobile hosts access to faster machines on the fixed network.

4. This study identified the type of applications which are most likely to benefit from load sharing. Previous studies on load sharing have found that it is CPU-intensive jobs that have most to benefit from load sharing. The findings of this study show that this is also the case in a mobile computing environment. Off-loading CPU-intensive jobs to fixed hosts extends the duration the CPU operates in doze mode considerably, thus reducing power consumption. A power management technique which reduces power consumption by the CPU is especially important when a user is using his mobile device to perform CPU-intensive jobs, in which case it is critical that battery power is not suddenly depleted, resulting in a considerable amount of frustration on the part of the user. It is imperative that the power management strategy should be able to prolong battery lifetime in order to allow the user to finish his tasks. It is on the basis of these considerations that we argue the merits and importance of our proposed approach.
5. This study shows that it is possible to make use of unutilised resources on fixed hosts to conserve battery power on mobile computers.

1.3 Thesis Outline

The thesis outline is as follows. The wireless networks and mobile computing issues are discussed in Chapter 2. This starts with an overview of the diversity of mobile applications, followed by a discussion of the challenges which must be met in order to support mobile users due to the inherent limitations of wireless communications. The power management strategies proposed thus far are examined.

Previous studies on load distribution in distributed systems and the extent to which they are relevant to this study are discussed in Chapter 3. This is followed by a discussion on load distribution policies and the effect of communication delays on the performance of load distribution algorithms. Finally, the importance of addressing stability and scalability issues are discussed.

In Chapter 4, the approach taken in designing a load sharing algorithm for wireless networks is described. The reasoning behind the design decisions and the factors which were taken into account in developing the algorithm are explained, and the extent to which previous work on load distribution is relevant is discussed. This is followed by a discussion on the trace data collected for use in the simulations, the assumptions made, the simulation environment, the system parameters and the experimental design.

In Chapter 5, experiments carried out to determine factors influencing load sharing in wireless networks are discussed. In addition, the benefit of combining load sharing with another power management strategy is also examined.

In Chapter 6, the issue of stability and scalability of the load sharing algorithm are addressed. The modifications carried out on the proposed load sharing algorithm to prevent instability are discussed. The way in which the algorithm is modified to disable itself when load sharing is no longer feasible and the parameters influencing that decision are explained.

In Chapter 7, the emulation carried out to verify and validate the simulation results is discussed. The emulation results are compared to the simulation results in order to determine the extent to which the simulation represents a real environment.

In Chapter 8, a vision for the future wireless and mobile applications is discussed, followed by a discussion on the applicability of this work. Possible future work is also explored. Finally, Chapter 9 concludes.

2. Wireless Networks and Mobile Computing Issues

Advancing technology in wireless communication offers users anytime, anywhere access to information and network resources without restricting them to the fixed network infrastructure. Mobility introduces new challenges as several assumptions made regarding distributed networks are no longer valid. Many of the research issues regarding wireless networks and mobile computing are not new, for they have been discussed in the context of distributed systems. However, the fact that users are no longer restricted to fixed hosts, that users are free to roam the globe and can connect to a network from various locations, and that addresses no longer give the location of a machine, have made the research problems harder and more interesting.

Wireless networks are also associated with various constraints - bandwidth is scarce, the quality of connection varies, communication delays are high and users may disconnect frequently from the network. In addition, mobile devices usually have lower computing power and storage capacity compared to a host on a fixed network. Mobile devices and applications have to address these limitations in order to deliver services which are of acceptable quality to the users. As users roam, they will encounter heterogeneous network environments and computing resources may become lost or new resources may become available. Mobile applications need to be smart enough to be able to make full use of resources as they become available.

Katz [Katz94] saw the progress in wireless communications as the next logical step in the evolution of computers, where computing resources can be used more flexibly as users are freed from being physically connected to the underlying network. He defined ***wireless information systems*** as computing systems that provide the ability to compute, communicate and collaborate anywhere at any time. In his paper, Katz gave the following definitions, though it should be noted that there is no general consensus in this matter:

- ***Wireless computing*** refers to computing systems that are connected to their environment via wireless links, such as radio frequency (RF) or infrared (IR), and usually apply to computing devices participating in a wireless LAN, with gateways to wired networks. Users are able to participate in work groups via a collection of computing devices and servers in order to share data and information, implying relatively symmetric bandwidth between the wireless node and the network, and relatively high bandwidth.
- ***Nomadic computing*** refers to the ability to compute as users relocate from one computing environment to another. In this scenario, individual organisations with their own wireless infrastructures are linked together by public wired internetworks. Nomadic computing

will make it possible for a user to use his own device within a foreign organisation's wireless infrastructure. The issues of trust, security and privacy must be addressed to enable users to roam in foreign environments while protecting their privacy, and the foreign networks against malicious users.

- ***Decoupled computing*** refers to the ability to compute when disconnected from the network. If mobile devices are full-function computers such as notebook computers, decoupled computing makes operations such as file access transparent across disconnections by using techniques such as prefetching and caching.

Katz also predicted that the distinction between communications and computing will continue to blur, leading to a new field of telecomputing.

In addition to the definitions above, Weiser [Weis93] presented the interesting idea of ***ubiquitous computing*** where computers are made *'invisible'* to users by integrating them into users' environment. He pointed out that anthropological studies of work life showed that people primarily work in a world of shared situations and unexamined technological skill. The computer technology today does not conform to this description because it remains the focus of attention instead of being a tool which disappears from users' awareness. Ubiquitous computing aims to make computers widely available throughout user environments and effortless to use. In other words, users should be able to use computing devices without having to acquire technological skills to use them.

The definitions above indicate different classes of applications and, in the following sections, the diversity of mobile/wireless applications will be described, followed by a discussion on the constraints and limitations imposed by wireless networks which must be addressed in order to provide services which are of acceptable quality to the users. Later, how this thesis addresses one of the constraints of mobile computing, that of power management, is discussed.

2.1 Mobile Computing Applications and Services

There is a wide range of applications (either under study or readily available) to support users on the move. The applications and services range from personal guides and electronic news services to collaborative applications for emergency services.

Mobile context-aware applications can be used as personal guides in museums and galleries which will allow users to take personalised tours [Long96]. Information about an exhibit is downloaded as a user moves towards it and he could then download more detailed information about it. Personal guides are different from human guides in that a user can browse and download information tailored to his own interests. Objects of interest are sensed

using active beacons or identified using computer vision recognition. The hand-held devices might use position measurement systems such as indoor beacons or The Global Positioning System (GPS) to locate users, and an electronic compass or inertial navigation system to find user orientation.

Context-aware devices can also be used as a measurement tool and to assist field studies. Simple hand-held sonar devices can be adapted to videotape and map a room along with user's comments. An ecological field study may be assisted by a device that automatically records the context of a particular species, assists the user in recognising plants, and also notes the surrounding objects. Another application is as an enhanced reality tool where a head-up display provides "*x-ray vision*" for a user surveying a building for renovation, to indicate the location of hidden plumbing or electrical conduits to the user based on information from sensors and/or building plans.

Another class of application is electronic news services discussed by Imielinski and Badrinath [Imie94b]. The electronic news services deliver and filter information according to individual user profiles. For example, traffic information or weather reports are filtered based on users' locality, while stock information is filtered according to users' portfolios. This class of application is further illustrated by Shekar and Lin [Shek94] who described an application known as Advanced Traveller Information System (ATIS), which provides up-to-date information on weather and road conditions. It also provides travel information such as diversions, construction zones, bus schedules and parking etc.

Imielinski et al [Imie95] predicted that the massive market for mobile computing which will emerge by the end of the decade will be based on mail-enabled and information services oriented applications. These applications differ from traditional applications in that they are not computationally intensive applications and will not need to run on powerful computing devices.

Duchamp [Duch92] predicted that mobile applications will metamorphosise low-skill and/or labour-intensive jobs into more information-based tasks which will affect people whose job involves movement over a wide area in order to deliver things or to visit immovable people or things, e.g. repair personnel, nurses and inspectors. Katz [Katz94] gave examples of how the metamorphosis envisioned by Duchamp will change the way people in various fields work. Wireless information systems will make possible collaborative applications that require untethered real-time access to multimedia information sources to provide support for personnel in the field, emergency services, law enforcement, mapping and location finding etc.

Katz illustrated the diversity of wireless communications in a crisis management application using a multimedia terminal for fire-fighters. The application can be used to provide maps and architectural blueprints to assist in planning fire-fighting strategy. A locator system may be incorporated to track team members as they move through the building. It should also provide voice and data communications among the team and other emergency and civil defence teams. The dynamic nature of this combined data makes it impossible to pre-store all information on the mobile device; instead, communication with a wide range of data rates must be supported, some of it unidirectional, some bidirectional and interactive. For example, symmetric communication is required to communicate with other members of the team, while downloading maps and location and tracking systems can be supported by asymmetric communication.

Mobile computing technology can also be used to support field work and increase collaboration among field workers by providing on-line access to information and interactive communication facilities. The MOST project (Mobile Open Systems Technologies for the Utilities Industries) was established to examine the impact of mobility on working practices and on the repercussions for computer systems support [Davi93, Chev94, Frid96]. It focused on the IT requirements of field engineers within the power distribution industry.

Engineers working in the field were traditionally coordinated by a single control centre which approved all switching in the power distribution network and maintained an overall picture of the current state. The centralised approach ensured that conflicting requirements were resolved safely, but the centre became a bottleneck. The main disadvantages of this approach were the global network state was not available to engineers in the field and efficiency was reduced due to the bottleneck. The second point was particularly crucial when faults occurred requiring multiple unscheduled items of work to be carried out.

In order to help field engineers work more efficiently, they were given access to information previously held only at the control centre and were allowed to collaborate by exploiting the GSM network. Information which rarely changed was provided on a CD-ROM or stored on hard disk, while dynamic information was provided via a communication link to the centre. The new system also allowed them to update the current state to reflect the operations they carried out. Engineers not physically located together were able to view and manipulate shared diagrams and information. They were also provided facilities so that they could communicate with each other to ensure that conflicting switching requirements were resolved safely. The application had real-time aspects and was based on a peer-to-peer architecture instead of a client-server architecture.

Another example of collaborative application in the field is Wireless Coyote [Gran93], which is an experiment conducted by Apple Classrooms of Tomorrow (ACOT) in cooperation with Orange Grove Middle School of Tucson, Arizona. An experiment was conducted to investigate how teachers and students could use technology in education. Mobile computers were connected by a wireless local area network (WLAN) and a wide area network (WAN) and a spreadsheet program was designed to provide real-time data sharing, immediate data display and real-time graph plotting. Students were provided with voice-activated walkie-talkies to support collaboration so that they could discuss their findings.

This experiment involved 5 groups of students, four teachers, a naturalist and personnel from Apple Computer. The students made a field trip to learn about Sabino Canyon in Tuscon. Four of the five groups were placed at different locations in the canyon, where three groups in the canyon used traditional scientific methods to measure soil and water temperature, wind speed and soil pH in their assigned locations, and the fourth group served as a base station and provided coordination among the other three groups by walkie-talkie, delivered additional supplies and coordinated requests for two digital cameras among the three groups. The base station was not involved in data collection, but was responsible for monitoring data collection activities of the field groups and transferred those data by cellular modem to the fifth group, located at the school which was 15 miles away. The school group built a database about the canyon based on the data collected by the field groups and used print and video resources at the school to add images and other content to their database. The naturalist provided them with expert opinion to help them to understand their findings.

The LAN made each group's data instantly available to all; consequently, they were analysing environmental data of the whole canyon. The walkie-talkie helped students and teachers to discuss their findings and to decide on the next step to take. The interactive nature of the application made the learning process more interesting and fun for the students.

Other collaborative applications proposed are Bayou [Deme94] and WebExpress [Hous96]. Bayou is an architecture which provides users with facilities to share appointment calendars, documents, bibliographic databases etc. in spite of intermittent network connectivity. WebExpress was designed for repetitive commercial applications and targeted for visiting medical personnel, salespersons, service workers who carry out repair at remote locations etc. It was designed to reduce data volume and latency of wireless communication.

This section illustrates the diversity of mobile applications and how the applications might change the way people work. Naturally, prediction of future trends is highly speculative and often subject to over-optimism by those in the field. However, it is undoubtedly the case that mobility has brought and will continue to bring forward new opportunities. Further, there are

now large commercial concerns whose existence derives from mobile telephony who have a vested interest in seeing (and selling) the expansion of the sort of services they offer. Consequently, the claims made for wireless computing are probably not far wide off the mark, although timescales may differ and commercial pressures are likely to play the major role in determining how far and how fast we can go. In the next section, the limitations of wireless networks and the challenges which must be met in order to deliver mobile applications at a quality of service which is acceptable to the users is discussed.

2.2 Limitations and Challenges of Wireless Networks

Several issues must be addressed in order to deliver the sort of applications and services described in the previous section. The problems encountered are not only due to the inherent limitations of wireless communications, but also due to the fact that the locations of mobile users/devices are constantly changing as users move. The configuration of a wireless network changes dynamically because users are no longer attached to a fixed point during the duration of a connection. The problem is further compounded by heterogeneous environments encountered by users as they move between various points of attachment.

The amount of bandwidth available to users as they move between networks varies greatly. While connected to a wireless LAN, a user may have available bandwidth of up to 2 Mbps, or 10 Mbps in the near future while when venturing outdoors, the available bandwidth may drop to only 9.6 kbps. Considering the amount of bandwidth currently available on fixed networks and the bandwidth-consuming applications available in the market, the low bandwidth, high latency communication medium offered by wireless networks must be considered primitive.

Satellite-based systems which provide wide area coverage are also used to provide the wireless communication infrastructure, albeit at a very high cost. Among well known satellite services are Odyssey, Globalstar and Iridium. There are three types of satellite services:

1. **Geostationary/Geosynchronous Orbit System (GEO)**, which is positioned 36,000 km above the earth, requires expensive satellites and large antennas, but with only three required to cover the earth. GEO may provide hundreds of high bit-rate data links using multiplexing, but involves a transmission delay of 0.5 second due to its high altitude. Its large regional coverage makes it difficult to provide the small-cell coverage necessary for frequency reuse to provide higher overall system capacity.
2. **Low Earth Orbit System (LEO)**, which is positioned 1000 km above the earth is the least expensive of the three types of satellite systems, but more is required to cover the earth. The coverage area is small compared to GEO, thus allowing a higher capacity

within a given spectrum allocation. The transmission delay is also significantly less than GEO.

3. **Medium Height Earth Systems (MEO)** is between the two extremes.

A much less expensive alternative to satellite systems is offered by the Stratospheric Telecommunications Services, which uses air platforms that remain geo-stationary above metropolitan cities to provide T1/E1 access to users in the service area[1]. The air platforms, which are positioned 22 km above the earth, incur very low communication delays and low infrastructure costs.

Apart from the variability of available bandwidth, users may also experience rapid and massive fluctuation in the quality of service provided by the wireless infrastructure [Davi94b]. Bagrodia et al [Bagr95] stated a fundamental way in which mobile computing differs from conventional operation is the huge variability in connectivity to users' computing environments. Even though there will be improvements as these limitations are addressed in future technologies, the discrepancy between wired and wireless networks are likely to remain [Wats94]. Badrinath and Welling [Badr95] claimed that although the constraints of mobile computing will become less noticeable, the mobility of devices will always induce constraints compared to non-mobile devices. Ebling et al [Ebli94] stated that even as global wireless connectivity becomes available in the near future, much of it will be intermittent with low bandwidth and high latency and will be limited to a few oases in a vast desert of poor connectivity. Watson [Wats94] argued that the limitation imposed calls for a software architecture which reduces the demands placed on the wireless link and supports disconnected operation.

It is very unwise to make any assumptions about the underlying support provided by the network infrastructure because, as users encounter heterogeneous environments while moving, resources and services not previously available may now be offered by the new network, or a critical service needed to run an application may be lost [Davi94a, Davi94b]. Mobile applications must deal with this heterogeneity and try to deal with failure gracefully while minimising inconvenience to users. Davies et al [Davi94b] termed the class of service which is designed to operate in a dynamic environment, and is able to adapt itself to the heterogeneity of the network as ***adaptive service***, while Katz [Katz94] termed this type of communication ***adaptive communication***. Katz regarded making applications aware of their limited and dynamically changing bandwidth as a critical challenge. Applications designed for mobile users must take into consideration the resource constraints they may face and be

[1] The information is obtained from the Sky Station International Inc. home page at http://www.skystation.com.

able to make the best possible use of available resources [Badr95], and they may need to present different views of functionality and quality depending on the location of the mobile device.

Duchamp et al [Duch91, Duch92] listed the challenges in designing system software to insulate applications from hardware and networking changes imposed by mobility:

1. *Support for mobile operation*: most current software for distributed computing assumes that computing devices do not move and packets are routed based on network number. This is especially true for routing protocols like the Internet Protocol (IP). This shortcoming, however, has been addressed by protocols such as Mobile IP [John97, Perk97a, Perk97b].
2. *User interface design for very small computers*: the user interface has to be modified to accommodate the shrinking size of portable computer's display.
3. *Adjustments to new hardware trade-off*: the hardware employed by portable devices will inevitably be different from hardware devices on fixed networks. For example, since portable devices might have reduced disk storage, new storage management algorithms will have to be designed.
4. *Emerging new technologies:* applications should be able to adjust and take advantage of any unique characteristics of new technologies that might be used in future portable workstations.
5. *Security*: providing authentication, accounting and management over a wide area and across organisations is not a new problem, but is aggravated and made more urgent by the advent of mobile computers.
6. *Compatibility*: the requirements above should be provided while retaining a reasonable level of compatibility, i.e. applications should provide interfaces and performance so that a user's desktop computing environment is available in his/her hand without the need to rewrite applications.

Davies et al [Davi93] classified the challenges of mobile computing technology as being either communications, distributed systems or cooperative working, which are explained below:

1. *Communication issues*: most existing wireless protocols are tailored for voice whereas mobile computing technology needs media access protocols for the radio channel which exploit the characteristics of the media and are suitable for transmitting a wide range of data types, e.g. voice and image.

2. *Distributed system issues*: the low bandwidth and error-prone wireless communication medium calls for object replication and message batching techniques to overcome the problems associated with poor communication channels.
3. *Implications for collaborative working*: collaborative applications in high-bandwidth networks often use multimedia to enhance interaction among users. The wireless communication medium restricts the range of services which can be offered by collaborative applications and the impact of this restriction on users' ability to collaborate has to be studied.

Davies et al's study of currently available technology and the user requirements of mobile computing systems led them to conclude that there is a significant mismatch between the two.

Katz [Katz94] stressed that in spite of the discrepancies between wired and wireless networks, mobile users should have access to the services they see in wired networks, albeit at potentially lower resolutions and possibly longer latencies. A crucial challenge is to make applications aware of their limited and dynamic environment so that they can adapt to what is available as appropriate.

Other constraints faced by mobile devices are that they have significantly lower memory capacity and computing power compared to a fixed host. For this reason, Badrinath et al [Badr93] proposed that the computation and communication load should be borne by the static network as much as possible. Doing so will reduce the burden of computation on mobile hosts and also helps conserve battery power.

The limited battery power on mobile devices is an issue which has to be addressed as users on the move will want their battery to last for as long as possible. Cox [Cox95] pointed out that the possibility of a ten-fold improvement in battery capacity in the short to medium term is essentially nil. Cox stated that, "*Frequently, the suggestion is made that battery technology will improve so that high-power handsets will be able to provide the desired five or six hours of talk time in addition of 10 or 12 hours of standby time, and still weigh less that half of today's smallest cellular handset batteries. This "hope" does not take into account the maturity of battery technology, and the long history (many decades) of concerted attempts to improve it. Increases in battery capacity have come in small increments, a few percent, and very slowly over many years, and the shortfall is well over a factor of 10. In contrast, integrated electronics and radio frequency devices needed for low-power low-tier PCS continue to improve and to decrease in cost by factors of greater than 2 in time spans on the order of a year or so. It also should be noted that, as the energy density of a battery is increased, the energy release rate per volume must also increase in order to supply the same amount of power. If energy storage density and release rate are increased significantly, the*

difference between a battery and a bomb become indistinguishable! The likelihood of a x10 improvement in battery capacity appears to be essentially zero."

Since there is unlikely to be much improvement in battery capacity, it is important that power utilisation is managed efficiently and economically. Without this, the potential of mobile system, and the commercial realisation of projects such as described above will be difficult to realise. It is hard to overstate its importance in the future development of mobile systems. This is the issue addressed by this thesis. The following section discusses the importance of power management and various power management strategies proposed to date.

2.3 Power Management Strategies

Power management strategies deal with techniques to reduce power consumption by power-hungry components, which often involve powering down the components during idle period in order to reduce power consumption. Imielinski et al [Imie94a] gave three reasons for the importance of power management. Firstly, it makes possible the use of smaller and less powerful batteries to run the same applications for the same duration, which is important from the portability point of view. Secondly, extending battery life allows a unit to run longer without the trouble of recharging which also results in monetary savings. The third advantage is from an environmental perspective, where disposal of batteries is an environmental hazard.

In this section, the studies which have identified the power-hungry components and the strategies proposed to reduce power consumption are described. Marsh and Zenel [Mars94] discussed the results of measuring power consumption of three components of portable computers which consumed a significant amount of power. They measured power consumption of the CPU, hard disk and display of four notebook computers, Toshiba 2200SX, the Toshiba 3300SL, the Dell 320SLi and the Zenith MasterSport SLe. The results of their measurements are as follows.

CPU: Each machine was measured at fast and slow clock speeds. The amount of possible saving varied from 11 - 31% of total system power. In the best case, slowing down the CPU clock speed extended battery life for over an hour for Dell 320. Since consumption varies linearly with CPU clock speed, Marsh and Zenel extrapolated power consumption of the CPU of each notebook by a simple calculation:

$$\Delta Clock.X = \Delta Watts$$

and solving for X. The calculations showed that all CPUs consume approximately the same amount of power, where the total power consumed by the CPU is 16 - 35% of total power. The high percentage is due to the 5V components used in the machines. More modern

machines which use 3.3V components reduce this figure significantly, because power consumption varies quadratically with voltage.

Hard disk: Measurements taken for hard disk power consumption showed that it consumes 8 -22% of total power. Not spinning the disk extends battery lifetime for 20 - 30 minutes. Results showed that a bigger drive consumes even more power. The trend in the marketplace however, is to increase density which does not help towards shrinking the disk size, indicating that disks will consume increasingly more power. Results also showed that a decrease in power due to lower-voltage CMOS logic does not reduce the power consumption of disk drives.

Display: The measurements were made with the backlight on and off. Turning the backlight off saved over 4 Watts on the Zenith and 2 Watts on Toshiba, but saved less than 1 Watt on Dell.

In summary, the measurements showed that slowing the CPU clock rate saved between 11% - 31%, spinning down the disk saved between 20% - 54% and turning the display off saved between 44% - 61% power. The CPU, disk and display were found to account for 44 - 60% of total system power.

Marsh and Zenel used three different strategies in powering down by the CPU, which are described below, in increasing order of complexity:

1. **Halt** is a technique for reducing CPU power by executing the halt instruction. Once halted, most of the transistors do not change state, reducing power significantly. The halt instruction is simple and consists of only 6-10 assembly instructions. The CPU still runs at full speed because the clock rate is not altered. When halted, no instructions are fetched and nothing happens until an interrupt, which may be generated by the real-time clock or any peripheral devices, is sent.
2. **Clock** relies on hardware and its i82360 power management chip. When the idle thread finds no other threads in the ready queue, it asks the i82360 to reduce the clock speed. The ready queue is continually inspected at the lower speed until a new thread appears. The clock is then reset to full speed and control is transferred to the new thread. The clock speed is also increased if an asynchronous software trap (AST) is received.
3. **Clock/Halt** first reduces the clock speed and then calls the halt instruction. Unlike the first approach, some logic may change state when halted. This approach reduces the CPU's power consumption while waiting for an interrupt. The idle thread checks the ready queue and reduces the clock speed and calls the halt instruction if there is no ready thread. When an interrupt is received, the chaining code turns the clock back to full speed. When the interrupt completes, control returns to the instruction following the halt instruction.

Results showed that Halt reduces total system power consumption by about 22%. Clock reduces power consumption by an additional 8% for Dell, and saved 11% less than Halt for Toshiba. Halt/Clock gave the best performance, reducing power consumption by 35%.

Marsh and Zenel demonstrated that using suitable techniques, power consumption by power-hungry components can be reduced significantly. Forman and Zahorjan [Form94] suggested that power is conserved by careful design and efficient operation, while applications can conserve power by reducing their appetite for computation, communication and memory. There have been several studies which address the power management issue, by attempting to reduce power consumption by the CPU, hard disk and also by using the communication medium economically. In the following sub-sections, the proposed strategies are discussed.

2.3.1 Power Management Strategies for Hard Disks

Power management strategies for hard disks involve techniques for spinning down the disk during an idle period. The concerns which have to be addressed are summarised as follows:

- an appropriate threshold for the idle period so that the disk can be spun down safely,
- keeping access latency to a minimum, because once a disk is spun down, a delay is introduced when spinning up the disk, thus delaying access to data,
- spinning down the disk to reduce power consumption should not adversely affect the disk life span,
- spinning down the disk should not result in higher power consumption compared to when it is not spun down.

In the following paragraphs, the way in which these concerns are addressed by various studies is discussed.

Greenwalt [Gree94] pointed out that recent efforts to achieve low-power operating conditions have primarily targeted the hardware, where low-power subsystems are given controllable power and clocks to allow them to operate in multiple power states. Advanced Power Management provides a mechanism whereby system software can control the transitions between power states.

Greenwalt established that hard disk drives consume between 15-30% power depending on the amount of memory and disk usage. The amount of power consumed by the disk can be reduced by spinning down the disk during idle period, when there is no request to access it. The disk is spun up again on the next access request, and spinning it up consumes a considerable amount of power and introduces access latency. As a result, spinning down the hard disk vigorously can actually increase power consumption, in addition to increasing access delay. Hence, the disk must be idle for a minimum period of time in order to offset the

additional power consumed when spinning it up again and to justify the increased response time.

Greenwalt defined a disk critical rate, R_{cr}, as the rate where the average power consumed for a single access if the disk continually spins and the power needed to spin the disk up after spinning it down are equal. A timeout value, which is the minimum period of time a disk has to be idle before being spun down, is selected based on R_{cr}. His experimental results showed that:

- Since spinning up the disk after spinning it down consumes a considerable amount of power, spinning it down too quickly, might result in an increase in power consumption instead of decreasing it. On the other hand, if the timeout value is too high, the probability of spinning down the disk is almost zero.
- The power consumed by a disk is equal to the power consumed for the on/off cycle plus the extra power needed to keep the disk spinning until the timeout occurs.
- The on/off cycle is costly. Low timeout values reduce the reliability of a disk because rapid on/off cycles reduces the life span of a disk whereas large timeout values minimise power consumption and lengthen its life span. A common practice is to use a timeout value of several minutes in order to balance disk life span and reduce power consumption.
- As there is a delay incurred when spinning up the disk, performance improves as timeout values becomes longer. However, users may tolerate the increased delay if it happens infrequently.

Greenwalt also pointed out that power, performance and reliability are not always compatible goals and a compromise must be reached in choosing an appropriate timeout value.

Although Greenwalt proposed that a spin down policy should strike a balance between reducing power consumption and latency, Douglis et al [Doug94a] proposed an adaptive threshold policy which aggressively spins down the disk at the expense of higher latency. Douglis et al claimed that the high fixed threshold values used by manufacturers, which vary from 3 seconds to several minutes, was too high, and that spinning the disk for a few seconds without accessing it can consume more power than spinning it up again upon the next access. To examine the trade-off, they ran trace-driven simulations to evaluate different disk spin-down policies. They evaluated two algorithms: an off-line algorithm[2], which is optimal in

[2] An off line algorithm assumes it has a priori knowledge of disk access pattern and is able to make optimal spin-down decisions.

terms of power consumption; an online threshold algorithm[3]; and a predictive online algorithm which uses past history to predict the next access. They found that threshold policies which spin down the disk after 1-10 seconds perform almost as well as the optimal off-line algorithm. Whilst this approach reduces the timeout value recommended by manufacturers by approximately half, in some cases, the threshold significantly increased access latency. They claimed that the delays can be avoided if access time could be predicted accurately, but admitted that predictive strategies that narrow the gap between optimal off-line and threshold-based algorithms are difficult to formulate.

A later work by Douglis et al [Doug95] proposed an adaptive disk spin-down policy which tries to balance between reducing power consumption and reducing access latency. They developed a method for distinguishing between undesirable and acceptable spin-up delays. Undesirable delays are referred to as ***bumps*** and the timeout value was varied based on users' tolerance of bumps. The adaptive policy is aimed either:

- to reduce the number of bumps without adversely affecting energy consumption compared to a fixed-threshold policy, or
- to reduce energy consumption without adversely affecting the number of bumps.

A fixed threshold policy spins down the disk if it has not been accessed for T seconds. As a spin-down policy has two contradicting goals, which is to reduce energy consumption while minimising access latency, using a low timeout value may reduce power consumption, but increases the number of bumps. Moreover, as mentioned earlier, a short timeout may increase instead of reducing power consumption due to the cost of spinning up. A fixed threshold policy usually uses a large timeout value to minimise bumps.

Douglis et al used a break-even point, T_d, which is the critical value R_{CR} discussed earlier, as the minimum period a disk remains idle before spinning it down. The adaptive spin-down policy monitors the spin-down threshold and updates the threshold to balance between energy consumption and bumps. They defined the measure of acceptability as $\rho = \delta/I$, where δ is the spin up delay and I is the idle time of the disk prior to spin-up. A spin-up delay is a bump if $\delta > \rho I$, all spin-up delays are bumps if $\rho=0$.

The threshold value is adjusted using either an additive or multiplicative approach. Using an additive approach, a value α_a or β_a is added to T, depending on whether an undesirable or acceptable spin-up occurs respectively, where $\alpha_a > 0$, $\beta_a < 0$ and $\alpha_a > |\beta_a|$. The values are chosen so that when a bump occurs, T is increased to avoid the possibility of future bumps

[3] A threshold algorithm uses a fixed timeout value, T, where the disk is spun down if it has been idle for T seconds.

and when an acceptable spin-up occurs, T is decreased, but more gradually. Using a multiplicative approach, T is multiplied by α_m or β_m where $\alpha_m > 1$ and $1 > \beta_m \geq 1/\alpha_m$.

Simulations were carried out using traces from a Macintosh Power Book, Windows 3.1 and HP-UX. The results obtained, which are summarised in Table 2-1, showed different levels of power saving were achieved depending on the trace data used.

Table 2-1

Table summarising simulation results of [Doug95]

$\rho=0.05$	Windows	Macintosh	HP
Fixed threshold T=2, 5, 10, 30 and 300 seconds	A fixed threshold of 2s consumed the least energy, but caused over 50 bumps in a 1.5 hour period. Increasing the fixed threshold or using an adaptive policy reduced the number of bumps at the expense of higher energy consumption. E.g. T=30s resulted in no bump, but increased energy consumption by 48%. A fixed threshold of 10s decreased bumps by 66% and increased energy consumption by 15%.	Results showed no significant improvement of adaptive policy over a fixed threshold policy.	A fixed threshold of T=30s reduced bumps by about 50% with an energy increase of 18%.
Adaptive policy (adaptive approach)	T was varied between 5 and 30 seconds, α_a=2s, β_a=1s. Energy consumption increased by 8% compared to fixed threshold of 2s and the number of bumps decreased by 65%.		Varying T between 10-70 seconds with α_a=2s, β_a=0.2s reduced bumps by 50% with an energy increase of 3%.
Adaptive policy (multiplicative approach)	T was varied between 5 and 30s, α_m=1.5s, β_m=0.5s Consumed only 0.5% more energy than a fixed threshold of 5s and reduced bumps by 33%.	A threshold between 2-20s using a multiplicative approach consumed 1% more energy and reduced bumps by 22%.	Not discussed.

The algorithm offers a method of varying the threshold dynamically to reduce bumps at the expense of power consumption. Users can make the trade-off by varying the threshold parameter manually, or the value may be varied dynamically by the algorithm based on recent history, which is preferable as user work patterns may change with time.

Li et al [Li94] reduced power consumption and avoided latency by using a disk cache to extend the duration a disk operates in sleep mode. Disk caches have been traditionally used to reduce disk access response time. From the perspective of power management, disk caches will not only reduce access response time, but also incur less energy cost compared to a system without a cache by filtering read accesses. Using a disk cache also reduces the overall impact of spinning down the disk and the number of spin ups required to service write requests. The cache used for the study was LRU (least recently used) with a block size of 4 kilobytes.

Li et al divided the hard disk operation into 4 modes. ***Off mode*** consumes no power and cannot perform any function except power up. ***Sleep mode*** is when the disk is powered up, but the disk platter is not spinning. In ***idle mode***, the disk is spinning, but there is no disk activity. In ***active mode***, the disk platter is spinning, and the disk head is either seeking, reading or writing and consumes the most power; this occurs only for short periods of time.

Li et al also studied the effect of a write-back policy which reduces the frequency of write accesses to a disk by delaying write operations to a disk for a specified period of time. The policy proved to be especially useful in cases where files are quickly over-written. Trace data for the study was collected from Microsoft DOS and Sprite file systems. The simulation results showed that a minimum amount of energy was consumed using a timeout value of 2 seconds, where it saved nearly 90% of energy consumed. A timeout value of zero increased power consumption due to frequent disk spin ups, as expected.

The performance of disk cache policy with a timeout value of 6 seconds was compared to an optimal spin down algorithm with the same timeout value. This value was chosen based on a critical value, R_{cr}, which was identified as 6.2 seconds. Results showed that the 6-second timeout policy saved 30% less energy compared to disk spin down with caching.

Li et al established that two factors which determine how much energy saving is achieved are, how frequently a disk sleeps and for how long it sleeps. The timeout value should be chosen so that it is small enough that it includes a cluster of disk activities, but large enough that the energy saved by sleeping is significantly larger than the energy consumed when spinning up. Decreasing the timeout value also increases the sleep time, which reduces energy consumption.

An analysis of the trade off between energy consumption and delays showed that a large saving can be achieved by tolerating a small amount of user delay. ***User delay*** was defined as the sum of all the spin up delays which are synchronous with user activities. Asynchronous spin ups from delayed writes were not considered because they were transparent to users. Results showed that a 2 second timeout did not carry a heavy performance penalty.

The disk cache was found not to have a profound impact on energy savings, even though it did reduce energy consumption by filtering disk traffic. The choice of timeout value was found to play a more prominent role in energy consumption. A disk cache of 1 megabyte reduced power consumption by half, and a larger cache did not further reduce energy consumption. The effect of varying write delays was found to be similar to the effect of varying the disk cache size. A write delay of 30 seconds reduced energy consumption by half, and a write delay greater than 60 seconds did not yield further improvement.

Krishnan et al [Kris95] took a different approach of spinning down the disk by using a sequential rent-to-buy problem, which is described as follows. If a resource is required for an unknown amount of time, and there is an option of renting it for £1 per unit time, or buying it for £c, for how long should the resource be rented before it is bought? An offline algorithm with a priori knowledge of resource requirement will immediately buy the resource if it is needed for at least c time units, and rent it otherwise. An online algorithm with no a priori knowledge will rent the resource for c time units before buying it, incurring a cost of at most twice of an offline algorithm. In the context of the spin down algorithm, keeping the disk spinning is a rent, while spinning it down is a buy, because spinning it down and then up again consumes a fixed amount of energy which is independent of the amount of time before the next access.

The disk spin-down algorithm is modelled as a sequence of single rent-to-buy problems, where the time between any two disk accesses is termed a ***round***. For each round t, the algorithm uses information from the previous $(t-1)$ rounds to decide if it should continue spinning the disk (renting) or spin down the disk (buying).

Krishnan et al proposed the "L" algorithm which sets a cut-off on the cost it is willing to accrue before buying, where the cut-off value is the critical value, R_{CR}. Two requirements for the algorithm are that it should produce good cut-offs, and do so using minimum space and time. The L algorithm assumes that a spinning disk consumes P_s power and a spun down disk consumes $P_{sd} > 0$ power, where P_{sd} is much smaller than P_s. T is the net idle time of a disk, which means the disk consumes at least $T.P_{sd}$ energy independent of the disk spin-down algorithm.

A parameter ε_X was used to compare disk spin-down algorithms performance based on the excess energy consumed by the algorithm, where ε_X = *total energy consumed* - $T.P_{sd}$. The effective cost of an algorithm X was defined as

$$E_{CX} = \varepsilon_X + a.O_x$$

where a = relative importance of latency with regards to conserving energy
O_x = the number of operations delayed when spinning up the disk

The buying cost c was derived as follows. A spin-down delays one operation and, therefore, the effective cost of a spin-down is $E_{sd}+a$, where E_{sd} is the total energy consumed by a spin-down and a spin-up. The effective cost per unit time to keep the disk spinning is P_s-P_{sd} and hence,

$$c = (a+E_{sd})/(P_s-P_{sd})$$

Simulation results showed that the effective cost for L algorithm was the smallest amongst online algorithms tested. Its effective cost was 6-25% lesser than the effective cost of a 2-competitive[4] algorithm. It saved 17-60% more excess energy than a 2-competitive algorithm, and 6-42% more excess energy than the 5 second spin-down threshold for $a < 25$. For sufficiently large a, it reduced the number of operations delayed over both the 2-competitive and the optimal offline algorithm. In addition, the rent-to-buy model was found to allow an effective trade-off between energy and response time.

An alternative to spinning down the disk to reduce power consumption was proposed by Douglis et al [Doug94b], who discussed flash memory as a storage alternative to hard disks, since it consumes low power. While hard disks provide large capacity at low cost and have high throughput for large transfers, they consume a lot of power and take time to spin up and down. On the other hand, flash memory consumes little power, has low latency and high throughput for read accesses. A disadvantage is that flash memories cost substantially more than disks and require erasing before writing to a segment, giving them a shorter life span.

There are two types of flash memory, a ***flash memory card***, which is accessed as main memory and a ***flash disk emulator***, which is accessed through a disk block interface. The access time and bandwidth of the devices are different, and so is the file system organisation. Since each organisation has its own strategies for power management and performance enhancement, when evaluating the suitability of flash memory as a storage alternative, these factors must also be considered.

[4] An algorithm is *c-competitive* if it never uses more than c times the energy used by an optimal algorithm [Helm96]. A 2-competitive algorithm uses a pre-determined fixed timeout value which is guaranteed to perform well for all sequences of idle time and its performance never exceeds the performance of a best fixed timeout algorithm.

Simulation results showed that flash memory achieved a power reduction by an order of magnitude compared to aggressive disk spin-down policies. The flash disk file system saved 88% power, while the flash memory file system saved 93% power in the presence of power management.

Douglis et al also tested flash memory's read and write performance compared to hard disk in the absence of disk spin-down. A flash disk file provided similar read response time, but its write performance was 50% worse than for the disk file system. The flash memory file system achieved faster read response time, while its write response time is comparable to a disk file system. Although the results indicate that flash memory is a good alternative to hard disks, at present, its capacity is not big enough to replace hard disks. If this constraint is overcome, accompanied with a reduction in price, flash memory may become an attractive alternative to hard disks.

While the strategies to spin down the disk vary, they attempt to achieve a common goal: that is, to identify a suitable time to spin down the disk in such a way that overall power consumption is reduced, while trying to keep latency within acceptable limits. It is difficult to compare the efficiency of the different techniques against each other as they used different performance metrics, e.g. [Li94] measured performance in terms of percentage of energy saved, while [Kris95] measured it in terms of excess energy consumed.

Since the power reduction achieved by a disk spin down strategy relies on the idle period, combining disk spin down with disk caching, as proposed by Li et al, seems a good way of lengthening idle periods, in addition to reducing access latency. Even though the strategies proposed have been successful in reducing power consumption by the disk, the different levels of saving achieved by Douglis et al when the spin-down algorithm was simulated using trace data from Macintosh, Windows and HP, suggests that the benefit of spinning down the disk might be influenced by the operating system used.

2.3.2 Power Efficient Communications

Transmitting and receiving data have been identified as one of the activities which drain battery power. The studies which have dealt with power efficient communication demonstrate that there are various ways to address the issue. Power consumption by transmitting/receiving may be reduced by using data filtering techniques, varying the transmitter power, or by using an adaptive error correction method. These are discussed individually below.

Imielinski et al [Imie94a] discussed an energy efficient data filtering technique which enables palmtops to operate in doze mode frequently and only wake up when data of interest is

expected to arrive. The ability to wake up when data is expected is termed ***selective tuning***. They illustrated the scale of potential saving achieved by operating in doze mode by referring to the power consumption of a Hobbit chip from AT&T which consumes only 50μW in doze mode, compared to 250mW in active mode.

Imielinski et al proposed two selective tuning techniques, where clients (mobile devices) tune in periodically to the broadcast channel to download required data. When a server broadcasts the data, it also broadcasts a directory which consists of an index indicating the time when particular records are broadcast. The index, which provides a sequence of pointers which eventually leads to the required data, is interleaved with data.

The two proposed techniques were (1, m) Indexing and Distributed Indexing. Using (1, m) Indexing, an index is replicated *m* times during the broadcast of one version of a file, where the index is broadcast every *1/m* fraction of the file. Distributed Indexing is an improved version of (1, m) Indexing, where only part of the index is replicated, as it was observed that it is not necessary to replicate the entire index between successive data segments. Only part of the index indexing the data segments which follows it is replicated.

Imielinski et al established that using Distributed Indexing, the same number of filtering requests could be served with 100 times less energy. Four hours of data filtering took only 1% of the energy used without filtering for the same number of requests. The savings almost doubled the working time.

In a later study, Imielinski et al [Imie95] proposed protocols which offer power saving for applications such as news groups and electronic mail. The protocol refines indexing of multicast data and saves energy by keeping the CPU in doze mode. The protocol keeps the receiver off most of the time and was inspired by a similar protocol used in cordless telephones and pagers, which extends battery life by switching off their receivers for much of the time.

Rulnick and Bambos [Ruln96] addressed the question of how to determine when, and at what power, a mobile terminal should attempt transmission. While overcoming interference in wireless networks is a focal concern, it should be done in a manner which conserves energy. Their aim was to minimise energy consumption on condition that the quality of service requirements are met.

Rulnick and Bambos considered a transmitter sending data to a remote terminal over a communication channel which is subject to time-varying interference. Transmitting at a higher power results in a higher SIR (signal to interference ratio) and success rate, but at the expense of rapidly depleting energy supply and increased interference with other users.

A dynamic power management algorithm (DPMA) which requires no prior knowledge of its operating environment was proposed. Its only requirement is a good estimate of the error probability function, using which it should be able to decide when to delay transmission if interference was too high. The performance of DPMA was compared to constant-SIR and constant-power solutions.

Simulation results showed that DPMA achieved power savings of at least 70% and battery life was improved by a factor of at least 3.3 compared to any other constant-SIR solutions. DPMA was also more stable compared to the constant-SIR scheme which rapidly became unstable due to the transmitters competing until power was saturated at exceptionally high levels. DPMA did not seem to induce a strong TDMA-like effect, but there was enough alternating behaviour which improved SIR per bit, hence suggesting a weak TDMA-like effect. Transmitters which used DPMA uniformly provided the same performance (in terms of rates and delays) as constant-SIR, and achieved energy savings of 10%- 92%. Battery life was extended by a factor of up to 12.

Lettieri et al [Lett97] described an architecture for low power error control over wireless links. Error control schemes such as forward error correction (FEC), or automatic repeat request (ARQ) are used to increase the reliability of the communication links. In wireless links, error control is used to provide sufficient reliability for an end-to-end transport layer to ensure that quality of service requirements are met. While error control has a direct and substantial effect on power consumption, studies on error control schemes have not addressed the amount of energy consumed to transmit bits across wireless links. Lettieri et al assessed the impact of error control schemes on battery life and showed that the choice of energy efficient control strategy is a strong function of quality of service parameters, channel quality and packet size. The error control scheme was adapted to changing channel conditions for maximum energy efficiency.

Any overhead associated with sending one bit of useful data was accounted for when calculating the energy cost of sending that bit. For example, the overhead for ARQ schemes is comprised of the redundancy in the packet for error detection, the acknowledgement packet and retransmitted packets, if any. The overhead for FEC is the redundancy in each packet, but no retransmission cost is incurred. FEC may be computationally expensive and the computational burden on power consumption cannot be ignored. A hybrid scheme incurs the fixed overhead of FEC, but with lower retransmission overhead.

Simulations were carried out with IP and ATM data packets, assuming data and speech transmissions, and users were assumed to move at car or pedestrian speed. Error control schemes used were SACK (Selective Acknowledgement), SACK+FEC and Reed-Solomon

(an FEC scheme which is capable of detecting multi-bit error). SACK was chosen because it requires the fewest retransmissions compared to other ARQ schemes, while Reed-Solomon was chosen because it incurs relatively low redundancy. SACK or SACK+Reed-Solomon generally gives the best performance.

The first simulations assumed data transmission. For ATM packet size, the improvement was minimal over a wide range of bit error rates (BER) when using a hybrid error control scheme, but the improvement was much greater when ARQ was used. Schemes which employed FEC performed better for poorer quality channels, but performance improvement decreased as the quality of channels increased due to the fixed cost of FEC schemes.

For IP packets, only SACK reduced energy consumption. FEC reduced the average latency for better channel quality, but did not improve energy consumption. In some cases, energy consumption was worse when FEC was used. Similar observations were made for both pedestrian and car speeds.

Simulations of speech transmission were only run assuming ATM packets and assuming that data had to be delivered within 50 msec. For both pedestrian and car speeds, no data was lost for low BER. The performance of SACK and SACK+Reed-Solomon were comparable, with the hybrid scheme consuming slightly more power. When BER rose above 10^{-3}, the ARQ scheme deteriorated to the point that packet drop rate was unacceptable and power consumption grew out of control.

The results obtained indicated that the error scheme should be adapted according to the type of transmission and different channel conditions in order to achieve efficient energy consumption. There is no single scheme which is suitable for all conditions.

The studies above demonstrate that there are various ways to reduce power consumption by transmitting/receiving. Even though the techniques proposed depend on which aspect was being considered and optimised in order to reduce power consumption by transmitting/receiving, there appears to be no reason why the techniques cannot be combined. In a real implementation, it is very likely that different power saving techniques will be combined to extend battery lifetime as long as possible.

2.3.3 Power Management Strategies for CPU

Power management of the CPU is similar to that of hard disks, in that it relies on powering down the CPU during idle periods. However, unlike disks, the cost and delay involved in powering down the CPU is negligible. Below is a discussion of studies which have addressed how power consumption by the CPU may be reduced.

Weiser et al [Weis94] believed that it would be more efficient to spread workload over time than to execute it at full speed and then idle. Spreading the workload is achieved by reducing the CPU clock speed (i.e. a job is executed at a slower rate), and the speed is chosen so that the job is completed before its deadline. Reducing the clock speed alone does not reduce power consumption because since power consumption is proportional to the clock speed, a job takes longer to execute, thus consuming an equal amount of power as if the clock was run at full speed. Hence, power consumption is reduced by reducing the voltage level as the clock speed was reduced. The proposed approach stems from the fact that energy is proportional to the square of voltage $E/clock \propto V^2$. Reducing the clock speed creates an opportunity for quadratic energy savings, because if the voltage level is reduced linearly as the clock speed is reduced, it has the potential to result in a saving proportional to the square of voltage reduction.

Trace data for the simulations were collected from UNIX workstations. The trace period was divided into intervals, which was varied during the experiments. The run time and idle time during an interval was used to decide on a suitable clock speed, where the speed was chosen so that the runtime of individual segments were lengthened in order to eliminate idle time. Excess cycles left over at the end of an interval were carried over into the next interval. The excess cycles indicated that the speed chosen was too slow, and was used as a performance penalty.

Three algorithms were used in the simulations. Unbounded-delay perfect-future (OPT) takes the entire trace and stretches all the run times to fill all idle times. OPT assumes perfect future knowledge of jobs to be done, and that all idle times can be filled by stretching run lengths and reordering jobs. Bounded-delay limited-future (FUTURE) is like OPT, but it examines a small window into the future and optimises energy over that window. Jobs are never delayed past the window, which was varied between 1 millisecond to 400 seconds. FUTURE approaches OPT in energy savings at window period of 400 seconds. FUTURE is impractical because it requires future knowledge, but is desirable because no jobs are delayed longer than the window period. Bounded-delay limited-past (PAST) is a practical version of FUTURE, which examines a window in the past and assumes the next window will be similar to the previous one.

A summary of the experiment results are as follows. Results of varying the scheduling interval on the algorithms showed that OPT was unaffected by the interval length, while FUTURE and PAST approached OPT as the interval increased. For the same interval, PAST performed better than FUTURE because it is allowed to defer excess cycles into the

next interval. Overall, results on PAST showed that a low minimum voltage of 1.0V resulted in more excess cycles, while longer intervals resulted in more accumulated excess cycles.

There was a trade off between excess cycle penalty and energy savings as a function of interval length. As the interval decreased, the CPU speed was adjusted at a finer grain and matched the offered load better, resulting in fewer excess cycles but did not achieve an optimal level of energy savings.

The amount of energy saving achieved for three minimum voltages, 3.3V, 2.2V and 1.0V, was examined using an interval of 20 msec. The best energy saving was not achieved by a minimum voltage of 1.0V because it tends to cause a build up of excess cycles, resulting in less efficient energy consumption. On the occasions when 1.0V did provide minimum energy consumption, 2.2V minimum voltage performed almost as well. Overall, energy saving achieved was 25% - 65%. PAST, with 50 msec interval, achieved up to 50% savings for 3.3V and up to 70% savings for 2.2V.

A later study by Lorch and Smith [Lorc96] tried to reduce power consumption of the CPU by optimising the operating systems for power management. They discussed strategies of reducing power consumption by the processor in a single-user operating system, focusing specifically on Apple's MacOS. When analysing the power consumption of Macintosh PowerBook computers, they discovered that up to 18-34% of total power was consumed by components whose power consumption could be reduced by power management of the processor.

As processors now are capable of operating in doze mode which consume little power, the operating system instructs the processor to operate in doze mode when it is predicted that the resulting savings will offset the overhead of entering and leaving doze mode. Unlike hard disks, the delay and power cost of doing so are typically low, which makes optimal CPU power management strategy trivial because the CPU can simply be turned off/down whenever there is no useful work to do.

The ***basic strategy*** turns off the processor when there is no process to run. The ***current strategy*** used by MacOS is based on an ***inactivity timer***, where the operating system instructs the processor to operate in doze mode if no activity is detected in the last 2 seconds, and no I/O activity is detected in the last 15 seconds. The processor leaves its doze mode when activity is detected, where an ***activity*** is defined as any user input, I/O, change in the appearance of the cursor or time spent with the cursor as a watch.

Lorch and Smith identified two problems with the existing strategy. Firstly, since the operating system was not optimised for power management, a process was sometimes scheduled to run even though it had no useful work to do. Secondly, applications were

usually written with the assumption that they will run in the foreground and, therefore, were justified in taking as much processing time as they wanted.

Lorch and Smith developed three techniques to deal with these two problems. The ***simple scheduling technique*** deals with the first problem by making sure the operating system does not schedule a process when it has asked to be blocked. The second problem is addressed using one of the following techniques. The ***greediness technique*** uses a heuristic to forcibly block any process which is making unnecessary requests for processor time. A process is considered as acting greedily when it specifies a sleep period of zero even though it is not actively computing; and is considered not actively computing if it explicitly blocks or yields control several times consecutively without receiving an event or showing signs of activity. The parameter of this technique is called the ***greediness threshold***, and is the number of control-yield times. The ***sleep extension technique*** multiplies all sleep times requested by a process by a constant factor known as a ***sleep multiplier***. It ensures a reasonable trade-off between energy savings and performance, and is chosen by the user or the operating system.

Simulations were run using trace data collected from six users of Apple Computers and with different combinations of the techniques described above. Two parameters were used to evaluate the performance of the proposed techniques: ***processor energy savings*** was the percentage decrease in the time the processor spent in the active state and was deduced from the simulations; ***performance impact*** was the percentage of overall time increase in running the processes as a result of using a power-saving strategy. The performance penalty is due to the fact that a power saving strategy sometimes caused the processor to be turned off when it should be performing useful work, resulting in the process being scheduled later and increasing the time taken to complete it.

Experimental results showed that an optimal strategy which has foreknowledge of work pattern achieved power savings of 82.33% with no performance impact (there is no increase in the average time taken to run processes), while the basic strategy achieved power savings of 31.98% with no performance impact. Various combinations of the proposed techniques achieved power savings of 28% - 66% with performance impact of less than 2%. A sensitivity analysis showed that the performance of the techniques was relatively insensitive to their parameter values.

The results of this study indicates that if the operating system is optimised for power management, a significant amount of power can be saved by turning off/down the processor when there are no processes actively performing computations.

A more recent study by Rudenko et al [Rudn98] proposed a strategy which transfers computation to a fixed host in order to keep the CPU idle as much as possible, thus reducing

power consumption. They observed that while some jobs have to be executed locally, some can be performed anywhere as long as the results are returned to the portable computer. They reasoned that if the cost of sending the job elsewhere and receiving the results is lower that the cost of running it locally, remote execution could save battery power. The scenario envisioned is an office where untethered users move through a ubiquitous wireless network.

Three types of jobs which were large and time-consuming were selected for the experiment:

1. Program compilation, which required significant CPU processing and a good deal of disk activities (read and write accesses).
2. Calculation of Gaussian elimination problem, which performed few disk accesses, but made very heavy demands on the CPU and memory.
3. Text formatting of a 200-page LaTeX document, which performed a moderate amount of CPU processing and had relatively little disk activity.

The amount of information sent from the portable to the server was varied. For compilation and text formatting, the server held a copy of the files locally and the portable was only required to send the changes to the server. The amount of changed code was assumed to correspond to the amount of work done by the compilation while the amount of data returned were always the same. In the case of the Gaussian solution process, the entire matrix was transferred and the matrix size was varied, thus varying the amount of data exchanged between the portable and the server. The size of results returned was assumed to correspond to the size of the matrix.

The experiments were first run in a noiseless environment, and later in a noisy environment. The results which are summarised in Table 2-2 shows that in a noisy environment, collisions and retransmissions greatly reduced the benefits of remote executions and only very large jobs benefit from it. Rudenko et al, however, pointed out that wireless communications cards, in general, are fairly new phenomena that are not yet ubiquitous and not yet optimised. They expected that there will be improvements in the future.

A shortcoming of this study is the fact that the experiments were carried out with the different job types tested separately. They did not consider a mix of jobs, which would be the case in real systems. In addition, all jobs were transferred even though they acknowledged that some jobs must be executed locally due to the nature of the jobs themselves. If these factors were taken into consideration, the results obtained might not be so optimistic. Our own work which has been briefly outlined in [Othm97] and [Othm98], takes into consideration several factors neglected by Rudenko et al.

Table 2-2

Summary of results obtained by Rudenko et al [Rudn98]

Job Type	Noiseless Environment	Noisy Environment
Program compilation	For small compilation jobs, the power cost of remote execution outweighed its benefits. However, as the size of altered code increased, the benefit of remote execution became more prevalent. For 500 kilobytes of altered code, remote execution consumed less than half the power consumed by local execution.	Only very large compilations saved a significant amount of power. The amount of saving was reduced to 20% from 51% for the noiseless environment because collisions and retransmissions increased the amount of power consumed.
Gaussian solution	For matrices of size less than 500x500, the power cost of remote execution were greater than the cost of local execution. For larger matrices, savings of up to 45% were achieved.	The results obtained were similar to program compilation in a noisy environment.
Text formatting	LaTeX documents of 30 to 200 pages were used in the experiments. There was no significant difference between local and remote execution except when the amount of altered text was 439 kilobytes, where remote execution performed worse than local execution. The reason for this observation was text formatting consumed less than 1% of total battery power. Remote execution added to the overhead of power costs and, therefore, did not help power saving.	Experiment for remote execution of text formatting in a noisy environment was not carried out.

In summary, while studies have shown that it is possible to reduce power consumption by turning down the CPU during idle periods, and by optimising the operating system, it is possible to further reduce power consumption by transferring computations to a fixed host.

2.4 Conclusion

In this chapter, the diversity of mobile applications was illustrated, and the limitations and constraints of mobile computing which must be addressed in order to support roaming users was discussed. One of these constraints is the focus of this thesis, that is power management. Several power management strategies have been proposed, most of which focused on

designing algorithms to power down hardware components which consume a substantial amount of power.

The power management strategy proposed in this thesis takes a different approach, and is based on the concept of load sharing, where suitable jobs are selected for remote executions in order to reduce power consumption by the CPU. It is, to some extent, similar to the strategy proposed to Rudenko et al [Rudn98], but predates their work and also takes into consideration several issues which they neglect. In the next chapter, studies regarding load sharing are discussed.

3. Load Distribution in Distributed Systems

Since load sharing has been extensively studied, there are numerous papers discussing various load sharing strategies. In this chapter, aspects of load sharing relevant to this study are discussed. As it is our intention to use load sharing within a wireless network environment, which is significantly different to the fixed network environment that is assumed within the literature to date, the extent to which such a strategy can be adopted is examined. We begin with a discussion on how load sharing has been used, and the general issues which need to be addressed when implementing load sharing on fixed distributed networks.

Load sharing, or load balancing, is a strategy which distributes workload among processors in a distributed system. Some literature distinguishes between the two (e.g. [Eage86b], [Krue88], [Krem92]). ***Load balancing*** is defined as a strategy which continually attempts to equalise workload across a distributed system, while ***load sharing*** is defined as a strategy which attempts to share loads in a distributed system without attempting to equalise them. For the sake of simplicity, a more generic term, ***load distribution***, which was introduced by Jacqmot and Milgrom [Jacq93], will be used when referring to both load balancing and load sharing in general. The goal of load distribution is to make better use of the system resources (usually the CPU) by making sure that no nodes are idle, and this is achieved by transferring jobs from a busy host to an idle/underloaded host. The transfer may be in form of ***job placement***, i.e. a non pre-emptive transfer of a newly arrived job, or ***migration***, i.e. a pre-emptive transfer of an executing job.

The primary reason for distributing load is to improve performance in distributed systems by making sure that no node is idle while other jobs wait for service and to minimise the total time taken to process all jobs [Eage86a]. Other reasons are to minimise inter-processor communication cost caused by accessing remote resources [Ezza86], and to improve performance by smoothing out transient peak overload periods [Krem92]. Shin and Chang [Shin89] stated that in real-time systems, load distribution is one way to alleviate a problem where some jobs cannot meet their deadlines even though the overall system has the capacity to meet all deadlines. This happens when job arrivals are not uniformly distributed causing some nodes to be overloaded, while some are underloaded. There are many other such reasons given in the literature. Regardless of why or how load distribution is performed, it should be performed in a transparent way so that users are not aware of the resources distributed across the network.

A load distribution algorithm may distribute workload based on the average behaviour of the system (***static algorithm***) or on the current workload information (***dynamic algorithm***). The

first case is simple and straightforward as job allocations are pre-determined, and the algorithm does not incur any overhead in collecting/disseminating system state information. In the latter case, information about the current system state must be collected and disseminated among hosts in the system, and the algorithm must be able to deal with information which is out of date due to communication delays. If this is not addressed, the inevitable existence of such delays may have a marked effect on the performance of a load distribution algorithm.

When designing a load distribution algorithm, it is imperative that the algorithm should be stable, i.e. it does not behave in a way which will cause system performance degradation under detrimental conditions. For example, when the system is under heavy load, it is perhaps better to disable the algorithm as the cost of executing it might degrade system performance instead of improving it. We will discuss how the definition of a "*stable algorithm*" is dependent on the designer's view of what stable behaviour is, and on the environment and system for which the algorithm is designed. A consequence of this is, there is no standard procedure for determining the characteristics of a stable algorithm, and it is up to the designer to identify factors which may cause instability and take the appropriate measures to prevent them.

As there are numerous load distribution algorithms proposed throughout the literature, it is necessary to examine the nature and precise properties of the algorithms in some detail in order to decide which are the most fruitful avenues for exploration with regard to their use in power saving within mobile systems. The remainder of the chapter is structured as follows. The next section describes the load distribution policies, the assumptions under which they were designed, how they have been used and whether the strategies are applicable in mobile systems. The effect of system load on the benefit of load distribution is also discussed, followed by the impact of communication delays on load distribution. Finally, the issues of stability and scalability of load distribution algorithms are addressed, and this chapter is concluded with a summary of the problems we wish to address.

Before proceeding to the next section, four commonly used load distribution algorithms are briefly described. These algorithms are popular and often used as a baseline comparison to other algorithms. In the coming sections, Random, Shortest, Threshold and Central algorithms will refer to the ones described below, unless stated otherwise.

A. Random Algorithm

An overloaded host selects a destination host randomly and transfer a job to that host. No probes are sent and absolutely no state information is used in selecting a destination host.

B. Threshold Algorithm

An overloaded host probes L_p hosts and selects a destination host whose queue length is less than a pre-determined threshold T, if one exists. The probe terminates when a suitable destination host is found, or L_p has been probed.

C. Shortest Algorithm

An overloaded host probes L_p hosts and selects a host with the shortest queue as a destination host to transfer its job, providing the queue length of the probed host does not exceed a threshold, T. A short-cut which can be used is immediately to select a probed host with queue length zero.

D. Central Algorithm

One host acts as a scheduler, and receives updates of workload information from other hosts in the network every t seconds. Each host sends its current workload information to the scheduler if there is a significant change in its workload since the previous update. If the number of jobs at a host exceeds a threshold T, it sends a transfer request to the scheduler, who selects a host with the shortest queue length and informs the requesting host to transfer its job there.

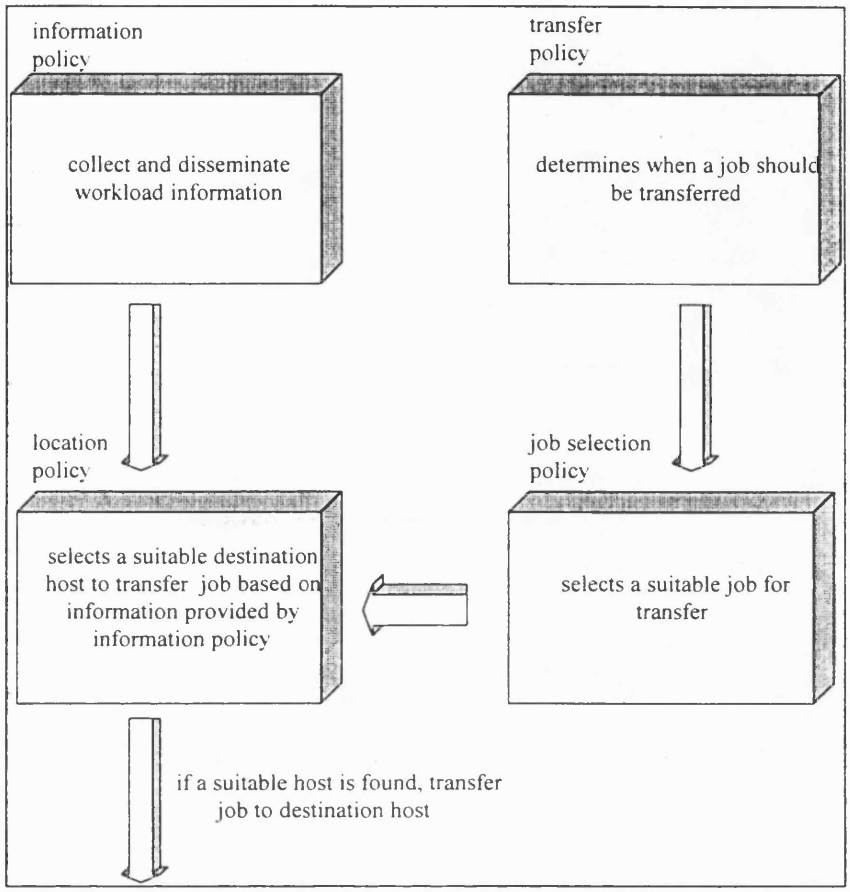
3.1 Load Distribution Policies

A load distribution algorithm is comprised of 4 policies: an information policy, a location policy, a transfer policy and a job selection policy[1]. The policies perform functions which enable a load distribution algorithm to determine how best to distribute load across the network in a manner which improves overall performance based on available workload information. Figure 3-1 shows a diagram of the policies and how they relate to each other. In the rest of this section, how and when each of these policies are used is described, followed by a discussion on the extent to which they are applicable to this study, and the modifications which have to be made to apply them to a mobile computing environment. The information and location policies are discussed together as they are very closely related.

[1] Watts et al [Watt96] divided load sharing algorithms into five phases, i.e. load evaluation, profitability determination, work transfer vector calculation, task selection and task migration. Phase 1 and 2 are analogous to the transfer policy, while phase 3 and 4 are a combination of the location and job selection policies. Phase 5 is the job transfer itself.

Figure 3-1

Components of a load distribution algorithm and their relationship to each other



3.1.1 Information and Location Policies

An ***information policy*** deals with the collection and dissemination of system state information used by the location policy. Load distribution algorithms vary from those which make no use of system state information (e.g. Random algorithm of [Eage86a]) to those which attempt to make use of global state information (e.g. [Krue94]).

The workload information may be obtained by probing a subset of nodes, by collecting the information periodically, or by having each node advertises its load information without explicit requests from other nodes. The workload metric most frequently used is run queue length. If the information is collected periodically, an optimal period must be determined. Collecting the information frequently will result in accurate and up-to-date information, but is expensive: $O(n^2)$ traffic overhead is incurred each time the information is disseminated, where n is the number of nodes in the system. A less frequent period may result in out-of-date information being used.

A concern when gathering workload information is the effect of communication delays. If we are to assume that communication delays are not negligible, as is the case when dealing with

low bandwidth wireless links, the delays involved when gathering state information makes it difficult to guarantee that the information obtained is up to date. Kremien and Kramer [Krem92] stated that, "*due to communication delays and state distribution, a complete and consistent view of the entire system, or even of a subset, may never be available at a node of the system*". Mirchandaney et al [Mirc89] found that as delays increase, the state information obtained becomes so outdated that it is useless. In order to overcome this problem, a number of studies have incorporated techniques to deal with the possibility of out-of-date information, e.g. Stankovic [Stan84] and Ezzart et al [Ezza86] introduced a bias and a window period to deal with the probability of using outdated information when making a decision.

If use of outdated information is a concern on fixed networks which have high bandwidth available, it is even more so over low bandwidth wireless links. In a wireless network, a mobile support station (MSS), or a base station, is the only fixed host a mobile host can communicate with directly. When implementing load distribution in that environment, the mobile host may request a transfer of a job to the MSS, or it may request that the job be transferred to a certain host on the fixed network. If the first approach is used, the MSS may decide if it is able to accept the transfer request and executes the job itself, or it may transfer the job to another fixed host for execution[2]. If the second approach is used, mobile hosts need to have information about the current workload status of fixed hosts. Even though it is possible to disseminate workload information to mobile hosts, the communication latency involved increases the probability of the information being outdated, and might even render the information useless. Taking this factor into consideration, the approach adopted by this study is to send transfer requests to the MSS, and let the MSS decide where the job should be executed. Doing so also reduces the complexity of the load distribution algorithm on a mobile host.

Selecting a destination host to transfer a job is the task of the ***location policy***. The selection is carried out either by choosing a host randomly, or by using workload information gathered by the information policy. For adaptive load distribution algorithms which make use of current system information state, there is a question of how much state information should be used in order to arrive at a decision. Attempting to acquire the most accurate and up-to-date information may lead to better decisions, but the overhead incurred may nullify any benefit of distributing load.

Eager et al [Eage86a] investigated the appropriate level of complexity for load sharing policies. They stated three concerns which arise from the complexity of adaptive algorithms.

[2] In this study, it was assumed that the MSS has the capability to perform these functions.

Their first concern is that the value of a policy depends critically on the overhead incurred, where excessive overhead may negate the benefits of load sharing. Secondly, some state information quantities such as the expected congestion at nodes in the near future, or the amount of processing required by a particular job, cannot be known precisely. Therefore, decisions reached based on this information may not be as good as expected and may even be relatively poor. The third concern is the potential for instability, where algorithms may react to small distinctions in system load due to the rapidly changing nature of the system state.

Eager et al tested three algorithms: Random, Threshold and Shortest. Threshold and Shortest randomly probe at most $L_p=3$ nodes to determine if a job transfer to the probed node causes the load at that node to be above some threshold, T, and if it does not, the job is transferred to that node.

All three algorithms use a threshold transfer policy which uses local state information, where probes are initiated when the local queue length exceeds $T=2$. If the number of jobs in a probed node is less than T, the new job is transferred to the probed node. No state information is exchanged in order to determine if a job should be considered for transfer, and transfer of jobs is given pre-emptive priority over processing of jobs.

The performance of the algorithms were compared to two bounding cases, no load sharing (K independent M/M/1 queues), and perfect load sharing at zero cost (M/M/K queue). The mean response time as a function of system load was used as a performance index.

The results obtained showed that Random yields substantial improvement over no load sharing and Threshold achieved further improvement for system loads[3] greater than 0.5. The further performance improvement achieved by Shortest compared to Threshold is negligible. Based on these results, Eager et al concluded that relatively simple information concerning potential destination nodes is sufficient to obtain essentially all of the benefits available.

There are other more sophisticated load sharing algorithms utilising more complicated information and location policies, e.g. the algorithm proposed by Shin and Chang [Shin89] for distributed real-time systems, where jobs must be executed before the given deadlines. They gave two reasons in support of an argument which says that probing a subset of nodes to find a destination node is not suitable for real-time systems. First, probing introduces additional delays in completing a job transfer. Second, if only a few nodes are underloaded, the sender may not be able to locate a receiver by probing a subset of nodes, which will lead to overloaded nodes executing their jobs locally and missing the deadlines of some of their

[3] System load indicates the level of workload at a host, and is defined as $\rho=\lambda\mu$, where λ=job arrival rate and μ=mean service time.

jobs. In real-time systems, the probability of missing deadlines must be kept to a minimum since the outcome may be disastrous.

A brief description of the proposed algorithm is as follows. Each node maintains state information for a small set of nodes called a ***buddy set***. The buddy sets overlap each other to distribute load evenly over the system. Three thresholds are used to determine the state of a node, and a node is classified as either *underloaded, fully loaded* or *overloaded*. Whenever a node changes state from underloaded to overloaded (and vice versa), it broadcasts its new state to other members of its buddy set, which eliminate a fully loaded node from, or add the underloaded node, accordingly, to its ordered list, named a ***preferred list***. An overloaded node selects the first node in its preferred list and transfers a job to that node without incurring any probing delay. The preferred list is permuted so that a node is the most preferred node of one and only one other node in a corresponding buddy set. The preferred lists of nodes in the same buddy set are different from each other to avoid flooding, and the order of preference may change over time.

The algorithm proposed by Shin and Chang is rather more complex than that of Eager's which simply probes L_p nodes. Although Eager et al argued that simple load sharing algorithms are sufficient to achieve good performance, it is difficult to say for certain what level of complexity is appropriate. The system and environment for which the algorithm is designed are determinant factors. While a simple algorithm may be sufficient in some cases, for the real-time system studied by Shin and Chang, a more complicated algorithm proved to be useful in order to arrive at better decisions.

In this thesis, we are not proposing new information and location policies for use on the fixed network. Assuming that load sharing is already implemented on the fixed network, whatever policies already in place may be used. If they are not already implemented, the policies must be chosen so that they fulfil the requirements of the system. There are, literally, hundreds of load sharing algorithms proposed in literature which were designed according to a set of requirements and assumptions. It is quite impossible to choose an information/location policy and say that this is the best policy that should be implemented as its suitability would depend on the type of tasks performed by the system, and the constraints under which it operates. The policies may vary according to the system, and that can be made transparent to the mobile hosts. That said, even the simplest of policies are likely to give significant benefits and operate well under a wide variety of condition if no further information about the job mix is available. Moreover, since it is the MSS which decides on a suitable destination host on which to execute a job, mobile hosts need not concern themselves with the policies used on the fixed network. An advantage of taking this approach is half of the burden is shifted onto

the fixed network, thus reducing the complexity of the load sharing algorithm on the mobile hosts.

3.1.2 Transfer Policy

A transfer policy determines when a job should be transferred in order to improve performance. The decision is usually based on the number of jobs in the run queue. Once the number of jobs exceeds a pre-defined threshold, the job selection policy is triggered. Previous studies on transfer policies established that the threshold value should be adjusted according to current system load to avoid performance from degrading at high system loads [Koya93][Zhan95]. At high system load, it is very unlikely that hosts would have spare capacity to execute jobs for another host. Under these circumstances, load distribution might cause performance to degrade if it continues to attempt to transfer jobs, wasting resources on fruitless transfer attempts. [Eage86a], [Koya93] and [Zhan95] established that a low threshold is appropriate at low loads, and the threshold should be increased at high loads.

As load sharing in this study is centred on the premise of making use of available spare capacity on the fixed network, system load is expected to play a prominent role in determining how many jobs can be transferred to the fixed hosts. Under heavy load, it is unlikely that hosts will be able to meet the requests from mobile hosts and it would be wise for load sharing to be disabled, not only to avoid performance degradation, but also to prevent mobile hosts from wasting battery power on fruitless transfer attempts.

In this study, since mobile hosts do not retain information regarding the current load at the fixed network and the transfer policy does not use a threshold policy, a different approach is required to prevent the load sharing algorithm from causing performance degradation at high system load. The method by which this is accomplished is discussed in Chapter 6 which addresses the issue of stability.

Another important consideration when designing a transfer policy is the cost incurred by the policy. Eager et al [Eage86a] represented the cost of job transfer as a processor cost only and did not consider the communication network cost. They assumed that the average job transfer cost was $0.10S$, where S=job processing cost, and claimed that transfer cost higher than $0.10S$ was unlikely because that indicates jobs with very low processing requirement are being transferred. They expected that any practical implementation of load distribution would select jobs with a relatively high ratio of processing cost to transfer cost. An analysis of the average response time vs. system load for four average transfer costs showed that performance was insensitive to transfer cost below $0.05S$, and degrades rapidly as the cost exceeds $0.25S$. Eager et al stated that the average transfer cost may be used to select a value for the transfer policy threshold, where low thresholds are appropriate for low transfer costs,

while high costs require higher thresholds.

When designing the load sharing algorithm for this study, the transfer costs includes both the processor cost and communication network cost. Considering that bandwidth is a scarce resource in wireless networks, it would be unwise to ignore the communication costs. The cost incurred is explained in Chapter 4.

3.1.3 Job Selection Policy

The task of a job selection policy, when triggered by the transfer policy, is to select a suitable job for transfer. A number of studies have paid little attention to how a job is selected for transfer and, often, it is assumed that jobs are homogeneous, and a newly arrived job is selected for transfer if the local queue length exceeds a specified threshold. Wang et al [Wang93] claimed that these are not practical assumptions as jobs have different characteristics, which should be considered when deciding if a job is suitable for transfer. It is, therefore, inadequate simply to choose a newly arrived job for remote execution.

In addition to local queue length, Zhou [Zhou88] considered job execution time when deciding if a job should be transferred. He introduced a threshold, T_{CPU} , which is the job execution time threshold. If the local queue length exceeds T and the job execution time is more than T_{CPU}, the job is transferred if a suitable destination host can be found. He stated that although job execution time is difficult to predict, it is possible to classify jobs into two categories: ***big jobs***, which are worth considering for load balancing; and ***small jobs***, which should not be considered. Trace data collected for the study showed that it was possible to classify jobs simply by looking at job names.

Experimental results showed that performance is relatively insensitive to the value of T_{CPU}. The average response time of $T_{CPU}=1.0$ second gave similar performance to $T_{CPU}=0.5$ or $T_{CPU}=2.0$ seconds, confirming Zhou's suspicion that only an approximate separation between large and small jobs is necessary to achieve good performance.

Zhou also observed that not all jobs are suitable for transfer, and that some jobs are immobile, i.e. they must be executed locally. Examples of such jobs are those which perform local services and/or require local resources, such as system daemons, and mail and message handling programs. He introduced an ***immobility factor***, which is the percentage of eligible jobs that have to be executed locally, and varied the value to study the effect of immobile jobs. The results obtained indicate that effective load sharing is still possible even if a significant proportion of jobs are immobile. Results showed that out of 50-70% eligible jobs, only 10-20% were actually transferred, and the small amount still brought performance benefits.

A later study by Svensson [Sven90] proposed a filter to be incorporated in load sharing algorithms to detect short-lived jobs and pass only long-lived jobs to the load sharing algorithm. The filter factor is $0 \leq F \leq 1.0$, where all jobs are passed to the load sharing algorithm when $F = 0$ and the algorithm is disabled when $F = 1.0$. A filter factor of 0.3 means only 70% jobs are passed to the load sharing algorithm.

Three filters were proposed: History filter based its decision on the average CPU time required to execute a job; Optimal filter (the upper bound) assumes a priori knowledge of the CPU time required by a job; and Random filter (the lower bound) selects jobs to be passed to the load sharing algorithm randomly. The load sharing algorithms used to test the filter are Shortest and Central algorithms.

The load level, load index and response ratio were used as performance metrics. The ***load level*** is the average CPU utilisation of all workstations, while the ***load index*** indicates the current workload at a workstation and is based on resource queue length. The ***response ratio*** is a function of response time, total CPU time taken to execute a job, mean disk access time, number of disk operations made by a job and total jobs.

Simulation results showed that the performance of Shortest with History was best when $F \approx 0.8$ and degraded for larger values, implying that effective load sharing can be achieved by transferring a small portion of jobs, if the right jobs were chosen. The RR when $F=0.95$ was found to be much the same as when $F=0$. Svensson also found that minimum system overhead should be spent on short-lived jobs, and the best strategy was to execute them locally.

There was a small difference between the performance of History and Optimal. Their performances were almost identical for moderate to high loads, while at high loads, Optimal outperforms History because the mistakes committed by History carry a higher penalty at high loads. The RR when $F=0.98$ was much the same as when $F=0$ for Optimal. Experiments with Central showed similar results to those with Shortest.

To ascertain the benefit of having a filter, Shortest was run without any filter and results showed that Shortest without a filter degrades rapidly with increasing job transfer cost due to the transfer of short jobs. An added advantage of using a filter was reduced traffic on the communication links. Using Shortest, the amount of traffic reduction was proportional to the value of F, where traffic was reduced from 90% to 20% when the filter value increased from 0.2 to 0.9. The traffic reduction with Central was lower because update messages were sent to a central machine.

The filter approach proposed by Svensson concurs with [Eage86a] which stated that a practical implementation of load sharing should attempt to select jobs with a relatively high

ratio of processing costs to transfer cost. It is also supported by Krueger and Livny [Krue88] who found that it is long jobs that benefit most from response time improvement.

A study by Koyama et al [Koya93] briefly examined three job selection policies: selecting a job randomly, selecting a job with the lowest priority and selecting a job with the smallest size. As their simulation results showed that the smallest size policy yielded the best performance, they expected that in a practical load balancing implementation, job size will be an important factor for selecting a job to be migrated.

A later study by Wang et al [Wang93] addressed the concern of selecting suitable jobs for remote execution by proposing a scheduler which learns the behaviour of a job and determines whether it is suitable for transfer. They stated three advantages of the proposed scheduler:

- A job is selected for transfer based on the characteristics of the specific job, in addition to the current network conditions;
- Knowledge of job characteristics is accurate because it is acquired through continuous observation;
- Whenever the system configuration or workload changes, the scheduler relearns and adapts itself automatically.

Jobs are classified as either CPU-bound, local I/O-intensive or memory intensive. Each job is associated with a weight, ranging between -100 and 100. Initially, the scheduler transfers jobs randomly. When the result of a transferred job is returned, its execution time is compared to the expected execution time, which is an average of recent execution times of the same type of job executed on the local machine. If the execution time is longer than expected, transferring the job is considered an incorrect decision and the corresponding weight is decreased; otherwise it is increased. If the system configuration changes, the corresponding weights are adjusted, and the scheduler relearns without human intervention.

The job selection policy was incorporated into StealthGS, which is a component of the Stealth Distributed Scheduler [Krue91]. Three factors were considered in making a transfer decision :

- the difference between local and remote CPU queue length,
- the difference between local and remote available memory sizes,
- the remote processor type.

Results of experiments with CPU-bound jobs showed that when a local node had more jobs than a remote node, the weights corresponding to these conditions rose to 100 after 40 executions. As a result, the policy almost certainly transferred the job. Short jobs were

considered suitable for transfer only when a local node had at least two more jobs than a remote node. If a local node had at least one job less than a remote job, the communication overhead of transferring a short job would negate its benefit. The selection policy learnt the behaviour of a job within 20 commands.

For local I/O-intensive jobs, the weights dropped below zero in all situations regardless of the local CPU queue length, and this type of job was never considered suitable for transfer. Memory-intensive jobs still benefit from remote execution even if a remote node contained one more job than a local node. In the case where both nodes contained the same number of jobs, but differed in the amount of available memory, the policy chose to transfer a job only when the remote machine had more available memory than the local machine. The results showed that once the policy learns a job is suitable for remote execution, it chooses to transfer the job most of the time.

Wang et al compared the performance of their intelligent job selection policy to the Threshold policy with $T=2$. Table 3-1 compares the performance of the two algorithms. For both CPU-bound and memory-intensive jobs, the intelligent job selection policy outperformed Threshold. The third performance comparison involved a mixed-job type. Results showed that Threshold caused performance to degrade because such jobs were not suitable for transfer. The job selection policy learnt not to transfer the job and, eventually, the average execution time dropped below local execution time. Finally, Wang et al examined the adaptability of the policy and found that it adapted itself after about 50 commands, leading them to conclude that it was capable of identifying changes and adjusting itself.

Table 3-1

Table summarising the average execution time of Threshold and intelligent job selection policy

Job type	Average job execution time (as a fraction of average local execution time)	
	Threshold	Intelligent job selection policy
CPU-bound	0.95	0.60
Memory intensive	0.90	< 0.40

An earlier a study by Hac [Hac89] proposed an algorithm which determines if it is better to move jobs which performs remote I/O to the remote machine, or to move the files it is accessing to the local machine. She found that for small files, moving the files resulted in a performance which is as good as transferring the job; when the files are large, transferring jobs resulted in better performance.

In summary, the studies above emphasised the importance of selecting suitable jobs for remote executions. It is not practical to simply transfer a newly arrived job as jobs have different characteristics and requirements. The following factors must be considered when selecting a job:

- Job execution time: short jobs should not be transferred as the overhead of transferring them outweighs any benefit which might be obtained. [Zhou88] established that only an approximate classification is required between big and small jobs.
- Job size: [Koya93] found that its policy of selecting the smallest size job yields the best performance, which is not surprising as small size jobs incur low communication costs.
- Available memory: a selected job should not exceed the available memory at the remote host. In addition, a memory-intensive job benefits from being transferred to a host with larger memory rather than to a host which is less loaded.
- I/O: jobs which perform a great deal of local I/O should not be selected for remote executions, and neither should interactive jobs. For jobs which perform remote I/O, the decision of whether the job or the files should be transferred, depend on the size of the files accessed by the job.

Interestingly, [Zhou88] and [Sven90] showed that only a small proportion of jobs, between 10-20%, have to be transferred in order to improve system performance, so long as the right jobs are selected.

As not all jobs are suitable for transfer, we classify jobs into migratable and non-migratable jobs. When a new job arrives, the transfer policy determine if the job is migratable or non-migratable. If it is migratable, it triggers the job selection policy which determines if the job should be transferred, or if it should be executed locally. The objective of the job selection policy is not to transfer as many jobs as possible, but to select suitable jobs so that power consumption is reduced. The job selection policy makes its decision based on job execution time and job size. We are unable to take available memory into consideration because that information is not available from the trace data used in this study (the trace data is discussed in Chapter 4). Even though the trace data provides information about the average amount of I/O performed by a job, it is not known if the operations were carried out on local or remote machines. Neither were there any information about the files accessed. Due to this lack of information, we were unable to examine the effect of remote I/O operations in wireless networks. The way in which job selection is carried out is further discussed in the next chapter.

3.2 Effect of Communication Delays on Load Distribution

A number of previous studies have assumed negligible communication delays, e.g. [Wang85], [Eage86a] and [Shin89]. On the other hand, there are several studies which considered communication delays as an important factor influencing the performance of load distribution. Since jobs are transferred to a less loaded host in expectation of improving response time, communication delays may have an adverse effect on performance. Stankovic [Stan84] and Hsu and Liu [Hsu86] found that even though response time does not increase linearly as communication delay increases, high delays do lead to increased response time. When studying the effect of communication delays on the performance of load sharing, Mirchandaney et al [Mirc89] assumed that the relative size of probes and jobs were different. If the size of probes was significantly smaller than the job size, the delays incurred by probes was assumed to be negligible. Mirchandaney et al established that, at low delays, performance improvement was substantial and was greater than that at higher loads. All three studies agreed that at high delays, the best strategy is to disable load sharing.

Given that load sharing now has to operate over low bandwidth wireless links, communication delay is a factor which cannot be ignored. In fact, high delay is a factor which may impede load sharing. In the load sharing algorithm we are proposing, the algorithm takes into account the available bandwidth and calculates the amount of delay which may be involved to perform remote execution. If the delays involved would cause worse response time than if the job was to be executed locally, the job is considered not suitable for transfer, and is executed locally.

3.3 Stability and Scalability

Stability is an important issue to address to prevent load distribution causing performance degradation under unfavourable conditions. Factors which cause instability and the measures taken to avoid instability differ according to the environment for which an algorithm is designed. Stankovic [Stan85] defined stability as reasonable behaviour of an algorithm within a bounded environment, where a bounded input will produce a bounded output, e.g. response time must have a finite bound which may be a non-linear function. Choosing the function is a subjective matter and when one has been chosen, it might be difficult to prove that the algorithm will always produce a response time lower than that given by the function. What constitutes reasonable behaviour is determined by the environment in which the algorithm operates, and what the designer of the algorithm perceives as reasonable behaviour. He stated that it is very difficult to define what is proper under all conditions - to ensure that a distributed algorithm always achieves the defined stability characteristics; and to evaluate or

identify all possible functional dependencies that exist. A stable algorithm should be robust, i.e. it should be able to handle failures.

Kremien and Kramer [Krem92] claimed that while "*a bounded input produces a bounded output*" condition is necessary for an algorithm to be stable, it is not sufficient. In addition, an algorithm should make local decisions and minimise incorrect decisions. They introduced ***hit-ratio*** and the percentage of remote execution in the system as measures of stability, where hit-ratio is the ratio of remote executions concluded successfully. A high hit-ratio indicates that an algorithm is successful in making correct transfer decisions, thus avoiding worthless information exchange and possible fruitless job movements. In order to be stable, remote executions should be bounded and restricted to a small percentage of system activities.

Kremien and Kramer considered algorithm stability as a pre-condition to scalability, and is an indication of the ability of the algorithm to avoid poor allocation decisions. An algorithm is scalable if it is independent of the system size and, in order to be independent, it should be symmetrically distributed, maintaining only a partial view of the system at each node, and it must be capable of making the most of partial information. A stable algorithm not only avoids a single point of failure, but is also fault-tolerant.

Stankovic illustrated how the stability of scheduling algorithms is a subjective issue using two algorithms, stochastic learning automata (SLA) and a bidding algorithm for jobs with deadlines. Using SLA, a host maintains the state of at most M underloaded host at a time. For stability purposes, each host has a priority sequence of hosts for which it observes if the hosts are underloaded, where the sequence is varied between hosts using a cyclic chain. By doing so, hosts will react differently even if they recognise the same network state. A destination host is selected based on a probabilistic vector which further reduces the chance of more than one host selecting the same destination host.

The bidding algorithm was designed for a real-time distributed systems with jobs having explicit deadlines. If a local scheduler cannot guarantee a job can be executed before its deadline locally, a bid is sent to other hosts. A host which receives the bid replies to the bid only if it has enough surplus (the surplus is the resource required to execute the job, e.g. CPU cycle, memory etc.) to guarantee the job. The stability issue here concerns the estimation of surplus, which is calculated using a technique called *Length of the Memory Accumulation Period* (LMAP), also known as the *moving average forecasting technique*. It is based on the assumption that the surplus in the future is similar to that in the recent past, which is a window of length t seconds. The performance metric used was the percentage of jobs guaranteed in the system and, since the algorithm deals with jobs with deadlines, a good estimate of the surplus is important because is affects how many jobs can be guaranteed. An

incorrect estimate will cause either jobs not being executed even though there is enough surplus, or jobs missing deadlines even though they have been guaranteed due to overestimates of surplus.

The examples given by Stankovic demonstrate how stable behaviour depends on the system for which an algorithm is designed. In the context of our study, the load sharing algorithm is considered stable if it does not cause power consumption to be worse than in a no load sharing case. This is especially important when there is a high number mobile users competing for a fixed host's spare capacity. The issue is elaborated in Chapter 6.

3.4 Conclusion

In this chapter, we have discussed aspects of load sharing relevant to this study. We first discussed how load sharing has been used in distributed systems and then proceed to explain how relevant the approaches taken by previous studies are to the problem we wish to resolve. Although to some extent the approaches may be adopted, modification is inevitable as load sharing now operates in a different environment, and under different constraints.

When designing the load sharing algorithm, the objective is to extend the period the CPU operates in doze mode and select jobs for transfer to reduce power consumption. A good load sharing algorithm is not necessarily one which moves a high number of jobs, but one which chooses the right jobs for transfer. In fact, previous studies have shown that less than 20% job movement is required in order to achieve performance improvement.

In Chapter 2, the limitations of wireless networks were discussed. Due to these limitations, there are additional factors which must be considered when performing load sharing in a wireless network environment and the assumptions made are inevitably different. Factors which are non-issues in fixed distributed networks now must be considered. One of the factors is available bandwidth.

When performing load sharing on fixed networks, bandwidth is not an issue as high bandwidth is available. This, however, is not the case with wireless networks. While on fixed network an assumption of negligible communication delays may be reasonable, this is certainly not the case in wireless networks which are associated with a low bandwidth, high latency communication medium. Consequently, when making transfer decisions, the available bandwidth and the communication delays associated with transferring a job must be taken into account to avoid degradation of job response time. At the beginning of this chapter, the goal of load sharing is stated as to distribute workload evenly across the system so that resources are better utilised. Doing so improves overall system performance, which is often measured in terms of improved response time, compared to the case when no load

sharing is performed. While the main objective of this study is to conserve power, load sharing is expected to give mobile users access to faster machines; thus transferring jobs is also expected to improve response time.

In section 3.3, the importance of addressing the issue of stability was discussed. Depending on the design of the algorithm, different factors have been identified to avoid unstable behaviour of a load sharing algorithm which may lead to performance degradation. The discussion has also demonstrated that the definition of stable behaviour depends on the environment in which an algorithm operates. In the context of this study, the load sharing algorithm is considered stable if, under unfavourable conditions, it does not cause power consumption to be worse than when there is no load sharing. Unfavourable conditions, in this case, are when there is a high number of users competing for the spare capacity on the fixed network. Under these circumstances, it is crucial that mobile hosts do not continue to send transfer requests when they are unlikely to be accepted, as doing so would only waste precious battery power. A method of disabling the algorithm is required to avoid instability.

Once the issue of stability has been addressed, we proceed to determine if the algorithm is scalable. It is probably not practical to expect only one fixed host to cater for job transfer requests, especially when there are a high number of users wishing to transfer jobs, as one fixed host might not be able to cope with the demand. When examining the scalability of the load sharing algorithm, the benefit of delegating transfer requests to other fixed hosts, and how well this approach scales, were investigated.

Another issue examined in this study is combining different power management strategies. Various power management strategies were discussed in Chapter 2. In a real implementation, it is desirable that various strategies be combined in order to provide maximum extension of battery lifetime. In one of the experiments conducted, the benefit of combining load sharing with another power management strategy was examined.

In the next chapter, the way in which the load sharing algorithm proposed in this study is designed to take into consideration the characteristics of the mobile computing environment is discussed.

4. Load Sharing in Wireless Networks

The load sharing algorithm was designed with the objective of extending the duration the CPU operates in doze mode by making use of spare capacity on the fixed network to execute jobs. In Chapter 2, the constraints associated with wireless networks were discussed. The way in which the constraints were considered when designing the load sharing algorithm is discussed in this chapter.

The remainder of this chapter is structured as follows. Firstly, the modifications on load sharing policies for a mobile computing environment discussed in the previous chapter are briefly revisited. Additional factors which must be considered and parameters influencing remote execution decisions are also discussed. Next, the approach taken in designing the load sharing algorithm is described, and the cost of remote execution and the trace data used in the simulations are explained. This is followed by a discussion on the simulation environment, experimental design, and the performance metrics and method used for data analysis. This chapter is concluded with a summary and the hypothesis for this study.

4.1 An Approach to Load Sharing in Wireless Networks

This section discusses how the assumptions made regarding load sharing in a mobile computing environment differ from those in fixed distributed networks and, consequently, the extent to which previous work in this area is likely to have a bearing on the problem we seek to address. The differences are now explained, starting with the load sharing policies.

In Chapter 3, the extent to which previous studies on load sharing are relevant to this study and the modifications necessary were discussed. The load sharing policies are now briefly revisited.

1. *Information and location policies*: We have argued that the policies chosen for a distributed system depends on the tasks it performs and the requirements of the system. While simple policies may suffice for one system, more sophisticated policies may be necessary in another system. Hence, for the purpose of this study, new information and location policies are not proposed as existing policies currently implemented on the system can be used. We chose not to disseminate workload information to mobile hosts because, even if the information are disseminated, mobile hosts are still required to send a transfer request to the fixed host in order to ascertain that the fixed host is willing to execute a job on their behalf. Omitting this phase might cause flooding if several mobile hosts simultaneously attempt to transfer their jobs to an underloaded fixed host. Instead of disseminating the information, transfer requests are sent to the MSS which invokes the

location policy to find a suitable destination host to execute a job on a mobile host's behalf, or it may execute the job itself. In this study, it is assumed that the MSS is capable of performing this function. In cases where the MSS does not offer this service, the request can be forwarded to a fixed host which offers the service. By taking this approach, half of the computation and communications burden of load sharing is transferred onto the fixed network. This is based on the suggestion of Badrinath et al [Badr93], [Badr96] that when designing an algorithm for mobile computing networks, the costs of computation and communications should be borne by the fixed network as much as possible. The core objective of the algorithm is achieved through distributed executions among fixed hosts and a mobile host only performs functions necessary for the overall functionality.

2. *Transfer policy*: Unlike previous studies, where the transfer policy triggers the job selection policy when the number of jobs waiting in the queue exceeds a pre-defined threshold, the policy does so on the arrival of a new job classified as migratable. The different approach is due to the fact that previous load sharing algorithms are trying to distribute load in order to improve performance while in this case, the goal is to conserve battery power. The classification of jobs is explained in section 4.3.
3. *Job Selection Policy*: The policy selects a job for transfer with the objective of extending the period the CPU remains in doze mode. Jobs are selected in such a way that power consumption of transferring jobs is less than that when the job is executed locally. The factors considered when selecting a job for transfer are execution time and job size. An additional factor considered is communication delays: jobs are only transferred if they do not increase job response time.

In addition to the policies, the cost assumption in this study is also different from previous studies, which usually assumed the cost of transferring a job as a function of probe cost, or time taken to transfer the job, or as a fraction of job service time, or a function of communication delays. Since the purpose of our load sharing algorithm is to minimise power consumption, the cost is calculated in terms of power consumed by transmitting/receiving and by the CPU. Even though communication delays are taken into consideration to avoid increased job response time, this is not factored into the cost. The cost of load sharing is explained in section 4.5.

Another assumption which is essentially different from traditional load sharing regards communication delays. Due to the limited bandwidth inherent in wireless networks, it is no longer valid to assume negligible communication delays, and probes can no longer be assumed to take zero time. In fact, the limited bandwidth and communication latency may become factors which impede load sharing in wireless networks.

Now that the differences between load sharing in fixed networks and wireless networks have been explained, the next section describes the proposed load sharing algorithm.

4.2 The Load Sharing Algorithm

When designing the load sharing algorithm, we decided to keep it as simple as possible for two reasons. Firstly, previous studies have shown that simple load sharing algorithms are often adequate to achieve good performance. Secondly, and most importantly, the algorithm should not be computationally expensive as that would put a heavy demand on the CPU, thus contradicting our goal to extend the period the CPU remains in doze mode.

It is assumed that only newly arrived jobs may be considered for remote executions. Jobs are classified as either *migratable* or *non-migratable*. When a job arrives and is identified as a migratable job, the load sharing algorithm determines the trade off between local and remote execution. Since the main goal of remote executions is to conserve battery power on mobile hosts, a job is only transferred if the amount of power consumed transferring it is less than the amount of power consumed by the CPU, if it is to be executed locally. If that is the case, the mobile host sends a transfer request to the MSS. Depending on its current workload, the fixed host sends a reply accepting or rejecting the request.

Three algorithms were used in the simulations. The algorithms perform the same calculations, but differ in the way they estimate the CPU requirement of a job. The base algorithm is as follows:

At a mobile host.

1 For each new job, determine if it is migratable.
  - 1.1 If it is migratable, go to step 2.
  - 1.2 If it is non-migratable, send job to local queue. Go to step 5.

2 Determine if a job transfer is feasible.
  - 2.1 Calculate the amount of power consumed to execute the job locally.
  - 2.2 Calculate the amount of power consumed for remote execution.
  - 2.3 Calculate the response time of local execution vs. remote execution.

3 Send a transfer request iff:
  - 3.1 The amount of power consumed transferring the job is less than the amount consumed to execute it locally, and
  - 3.2 The response time of remote execution will not exceed that of local execution.

4 Wait for a reply:
  - 4.1 If the request if accepted, transfer the job.
  - 4.2 If the request is rejected, schedule the job for local execution.

5 Algorithm terminates.

Step 1 is the transfer policy and step 2 is the job selection policy. In step 2.2, the amount of power consumed for remote execution is the amount of power consumed transmitting and receiving messages. The messages exchanged between the mobile and the MSS are:

- a transfer request sent by the mobile host to the MSS,
- a reply from the fixed host accepting or rejecting the request,
- the job transfer itself,
- execution result returned by the fixed host.

The transfer request sent in step 3 specifies a maximum period of time, *max-wait-time*, it is willing to wait for the fixed host to execute the job on its behalf. The maximum waiting time is based on the time required to execute the job locally. Condition 3.2 is imposed to ensure that job response time does not increase due to remote execution. The underlying assumption is that the mobile is aware of the processing capability of the fixed host and is able to predict how much faster the job will execute there, and specifies a value, *reserve-time*, which is used to reserve the CPU time required to execute the job at the fixed host.

At a fixed host

1 Upon receiving a transfer request, the fixed host checks its queue length and determines if it will be able to execute the job within the specified time, *max-wait-time*, based on the execution time of jobs it has waiting in its queue.

2 If it is possible to execute the job within the specified time:

2.1 Send a reply accepting the request.

2.2 Reserve the processor cycles required to execute the job. Go to step 4.

3 If it is not possible to execute the job within the specified time, send a reply rejecting the request.

4 Algorithm terminates.

Transfer requests are dealt with in a first-come-first-served manner. The reservation in step 2.2 is to guarantee job response time and also to prevent the fixed host from accepting too many transfer requests. The reservation is carried out as follows: if $J + R <$ *max-wait-time*, reserve processor cycles to execute the job,

where $J = \sum_{i=1}^{N} J_i$ and $R = \sum_{j=1}^{M} R_j$,

J_i = CPU time required to execute job i in queue,
N = total jobs in queue,
R_j = CPU time reserved to execute job j for an accepted transfer request,
M = number of transfer requests accepted by the fixed host.

After a request is accepted, R is updated,

$$R = R + \textit{reserve-time}.$$

It is assumed that the fixed host knows the CPU requirements of the jobs in its queue. In reality, this information might not be available, in which case an estimate must be used. As mentioned earlier, the algorithms (at a mobile host) differ in the way that they estimate the CPU time required to execute a job. How each algorithm estimates the CPU requirement is explained below.

Algorithm 1: Optimal Load Sharing (LS)

This algorithm assumes a priori knowledge of CPU requirements, where the CPU time required to execute a job used by this algorithm is obtained from trace data. It is used in calculation 2, and as *max-wait-time*. The calculation performed by this algorithm always gives accurate estimates and it is, therefore, used as an upper bound algorithm.

In reality, it is highly unlikely that this information will be available in advance, hence algorithm 2 and algorithm 3 make no assumption of this a priori knowledge and the CPU time used in calculation 2 is an average value.

Algorithm 2: History

The simulation is first run in a no load sharing mode (NLS) and the average CPU time taken to execute each job is calculated. For example, the CPU time taken for each execution of job A is totalled and the average value is calculated at the end of the simulation. Later, when the job is run in load sharing mode using History, the average value calculated in the previous simulation run is used in calculation 2. This value is also used as *max-wait-time*.

Since the same trace data is used when running simulations in NLS and History mode, the user's workload is exactly the same. In reality, it is highly unlikely that the same workload will be reproduced. Therefore, this algorithm provides an upper bound on performance for the use of history information.

Algorithm 3: Adaptive Load Sharing (ALS)

The adaptive algorithm learns and adapts its decisions based on previous executions of jobs. It works as follows. When job A is executed for the first time, its CPU requirement is unknown and, therefore, no assumption can be made regarding the feasibility of transferring it. It is executed locally and the algorithm keeps a record of the CPU time taken to execute the job.

The next time job A is executed, the CPU time from the previous execution is used in calculation 2 to estimate if remote execution is beneficial. Each time job A is executed, the

CPU time taken to execute the job is used to calculate a new average value which will be used in future calculations. Hence, when job A is executed for the *n-th* time, where $n>1$, the average CPU time calculated from the previous ($n-1$) executions is used as an estimate. Like History, this average value is also used as the value of *max-wait-time*.

This algorithm is expected to be the most practical of the three algorithms since it makes no assumption of a priori knowledge and is able to adapt its behaviour according to a user's working pattern.

The approach taken by History and ALS is similar to the filter approach proposed by Svensson [Sven90] which selects long-lived jobs for remote execution.

4.3 Trace Data

The discussion in Chapter 2 demonstrated the diversity of mobile applications. The type of workload generated by an application is likely to differ depending on the application, e.g. the workload generated by emergency services which mainly involves downloading information from a central database is probably different from the workload generated by the spreadsheet application used in the Wireless Coyote experiment. We would like the workload generated for the simulations to represent the workload generated by mobile applications as closely as possible. However, due to the diversity of mobile applications, it is impossible to guarantee that the workload generated is representative of all types of possible applications.

There are two options which could be taken to generate the workload: the use of a stochastic model or the use of trace data. A stochastic model was not selected because it does not capture user's behaviour accurately and, in particular, tends to misrepresents the bursty nature of users' activities. On the other hand, the major question with trace data collected from conventional network is: how well does the trace data represent the type of workload generated by mobile users? It is not possible to say for certain what type of workload is generated by mobile applications because, at present, people are not yet using mobile applications on a regular basis. Consequently, it is not possible to gauge the type of workload generated by mobile applications. However, it is plausible to assume that users will expect a seamless environment. In addition to mobile-specific applications, users will expect to be able to use similar applications and tools they are currently using at the workplace. If a seamless environment is not provided, users will have to face the inconvenience of working in two different environments. Taking this into consideration, trace data collected from existing applications provides the approximation most rooted in reality, and requiring the least amount of supposition about factors which are currently unknowable.

For this study, trace data for the experiments were collected from Sun workstations in the undergraduate labs at the Department of Computer Science, UCL, where process accounting was used to collect trace data for 24-hour periods. The trace data shows that the jobs executed consists of text processing, program compilation, email, web browsing etc.

The data was summarised to contain only the time period when the machines were being used. The time period during which no users were logged on were deleted, giving trace data for a period of 4 to 8 hours. [Zhou88] identified that some jobs are immobile and, consequently, must be executed locally. In this study, each job type is classified as either migratable or non-migratable. Examples of migratable jobs are program compilation, simulations and program runs, while non-migratable jobs are interactive jobs, text formatting and email. Among information provided by the trace data are job name, job start and end time, the CPU time (in seconds) taken to execute a job and the average amount of I/O (bytes) performed by a job. The trace data does not provide information regarding job size (size of executable files). This information was obtained by using the Unix command ls -l and incorporated into the trace data.

4.4 Power Consumption

An AST Power Executive 325/SL NiMh battery provides (14.4 V * 2.4 A-hr) ≈ 34.6 W-hr. Transmitting and receiving are assumed to consume 3.4W and 1.7W respectively [Imie96]. [Form94] lists power consumption of hardware components of a portable computer. A simple estimate of battery lifetime based on this information, is that a brand new battery lasts for about 3.4 hours, assuming that general power consumption (i.e. power consumption by the basic components such as the display, hard drive, keyboard etc.) is approximately 10.1 W.

$$\text{Power consumed when transmitting} = 3.4 \cdot \frac{t_r}{3600} \; W\text{-}hr$$

t_r = time taken to transmit a message (sec)

Power consumed when receiving is calculated in a similar way.

$$\text{Power consumed by the CPU is} = P_{CPU} \cdot \frac{t_m}{3600} \; W\text{-}hr$$

P_{CPU} = power consumed by the CPU

t_m = CPU time taken to execute a job on a mobile (sec)

Based on the information provided in Intel's Application Note [Inte94], the average active CPU power consumption (with Advanced Power Management) is assumed to be 4.59 Watts and idle power consumption (doze mode) is 1.24 Watts.

4.5 Assumptions

As mentioned earlier, since the aim of the load sharing algorithm is to minimise power consumption, the cost of transferring jobs is calculated in terms of power consumed by local and remote executions. The parameters used in order to determine if a job transfer is feasible are:

- available bandwidth
- job size
- power consumed by the CPU to execute a job on the mobile host
- power consumed transmitting and receiving messages.

B = *available bandwidth*
R_1 = *packet size of request message*
R_2 = *packet size of transfer reply*
J = *size job to be transferred*
R_3 = *packet size of the returned result*
P_t = *power consumed transmitting packets*
P_r = *power consumed receiving packets*
P_{CPU} = *power consumed to execute a job on the mobile host*

cost of sending transfer request $= \frac{R_1}{B} \cdot P_t$

cost of receiving a reply $= \frac{R_2}{B} \cdot P_r$

cost of transferring a job = cost of transmitting a job + cost of receiving result

$$= \frac{J}{B} \cdot P_t + \frac{R_3}{B} \cdot P_r$$

cost of remote execution = cost of sending a request + cost of receiving a reply + cost of transferring a job

$$= \left(\left(\frac{R_1}{B} \cdot P_t\right) + \left(\frac{R_2}{B} \cdot P_r\right) + \left(\frac{J}{B} \cdot P_t + \frac{R_3}{B} \cdot P_r\right)\right)$$

cost of executing a job on a mobile host $= \left(P_{CPU} \cdot \frac{t_m}{3600}\right)$

The job service time and job inter-arrival time of the mobile hosts were obtained from the trace data. Additional workload on fixed hosts was generated using an exponential

distribution because, in reality, a fixed host may have its own jobs to execute. If so, it can only accept a job transfer request if it has some spare capacity. By taking this approach, we create an environment in which a mobile host must compete for a fixed host's resources. By doing so, we hope to reduce the chances of obtaining over-optimistic results.

In the next section, the simulation environment and the experiments are described.

4.6 The Simulation Environment

The simulation was written using Maisie [Shor95], a C-based simulation language which is designed specifically for simulating wireless networks. Maisie provides a discrete event simulation environment, where an event is signified by the arrival of a message. It introduces the concept of an entity, where an entity may be defined to represent a component involved in a simulation. The entities communicate with each other using buffered message passing. An entity is similar to an object in C++.

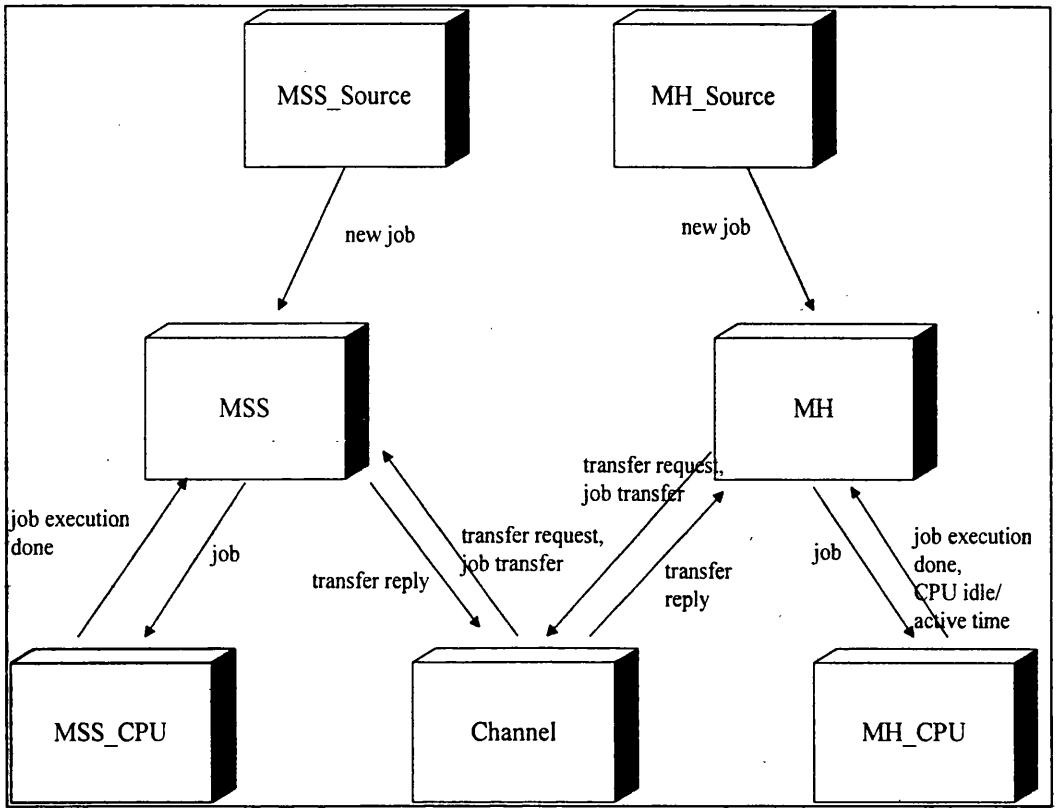
Maisie was chosen as a simulation tool for this study because the concept of an entity makes the representation of various components simple and the message passing facility makes managing communication between entities effortless; and its simplicity makes it very flexible to use. It has been used to simulate wireless networks in other studies, e.g. Bagrodia et al [Bagr95] and Short et al [Shor95]. The environment is assumed to consist of cells with a MSS in each cell. At the start of the simulation, mobile hosts are evenly distributed among the cells. The rest of this section discusses the entities and parameters for the simulation.

4.6.1 Entities

For the purpose of this study, entities were defined to represent mobile hosts, mobile support stations, fixed hosts and communication channels. The entities defined for the simulation and their interaction with each other are depicted in Figure 4-1. Below is a description of the functions performed by each entity.

Figure 4-1

Entities and messages passed between them.



MSS-Source Entity

This entity generates the additional workload on fixed hosts, where fixed hosts are assumed to have an exponential service time of μ and job arrival rate of λ. It notifies the *MSS* entity of new job arrivals by sending a message to it. There is one *MSS-Source* for each *MSS* entity.

MSS Entity

When it receives a message from *MSS-Source* notifying it of a new job arrival, the jobs is scheduled for execution and the *MSS* keeps track of the CPU requirement of jobs in its queue. The *MSS* executes the algorithm described in section 4.2 when it receives a transfer request and sends a reply message accordingly. When a transferred job is received, it is scheduled for execution along with local jobs and the result is returned to the *MH* entity when its execution is completed.

MSS-CPU Entity

There is one *MSS-CPU* entity for each *MSS* entity. Each time a new job or a transferred job arrives at the *MSS* entity, the job is sent to the *MSS-CPU* entity for execution on a first-come-first-served basis. Once a job is executed, the *MSS-CPU* entity sends a message to the *MSS* entity informing it that the jobs has finished executing.

MH-Source Entity

This entity reads the trace data and generates new jobs for entity *MH*. There is one *MH-Source* for each *MH* entity. The job inter-arrival time is calculated from the trace data based on job start time. If the algorithm is operating in NLS or LS mode, the CPU time for a job is obtained from the trace. If it is operating in History or ALS mode, a table lookup is carried out in order to determine the average CPU time for the job. Once this is done, a message is sent to the *MH* entity to inform it of the arrival of a new job.

MH Entity

Upon receiving the message from *MH-Source* notifying it of the arrival of a new job, the *MH* entity executes the load sharing algorithm described in section 4.2. It also calculates the remaining power on the mobile host periodically (every 60 seconds). The entity and, therefore, the simulation, terminates if the remaining power is less than the amount of power required by the basic components.

The entity outputs data which is used to calculate:

- battery lifetime,
- the percentage of transfer requests, rejected transfer requests and jobs transferred,
- the mean and standard deviation of job response time,
- the mean and standard deviation of communication delays.

There is one output file for each *MH* entity.

MH-CPU Entity

This entity performs 2 functions:

- simulate the execution of jobs sent to it by the *MH* entity and informs the *MH* entity when the jobs finish executing,
- keep a record of the amount of time the CPU is active and idle.

The amount of time the CPU remains idle/active is sent to *MH* periodically (every 60 seconds) and is used by the *MH* entity when calculating remaining power on the mobile host.

Channel Entity

Channel is used to simulate communication delays. When *MH* sends a message to its *MSS*, the message is sent through *Channel* entity, where the communication delay is calculated based on the packet size and available bandwidth. The same procedure is followed when *MSS* sends a message to *MH*. A reliable communication medium was assumed, and channel allocation was not simulated to simplify the simulation.

4.6.2 System Parameters

The 9 system parameters, listed in Table 4-1, were varied to allow us to examine the effectiveness of load sharing under different operating conditions. In the experiments, we are interested in establishing:

- the influence of available bandwidth on load sharing,
- the effect of job transfers on response time,
- the effects of processor power of a mobile host on load sharing,
- the effectiveness of the adaptive algorithms compared to the optimal load sharing algorithm,
- the benefit of combining load sharing with another power management strategy,
- the required measures to ensure the stability of the load sharing algorithm,
- further benefits which may be obtained from delegating job transfers to other fixed hosts.

Table 4-1

Table listing parameters used to vary the simulation environment.

Parameter	Specifies ...
param.alg	algorithm mode during a simulation
param.mss	total number of MSS
param.mh	total number of mobile hosts
param.bw	available bandwidth
param.proc-speed	processor speed of the mobile relative to the fixed host's
param.mss-load	(additional) job arrival rate at a fixed host
param.disk	enable/disable disk spin-down strategy
param.probe, *param.fixed-hosts*	enable/disable delegating transfer request; *param.fixed-hosts* is enabled when *param.probe* is enabled

Param.alg was used to vary the algorithm used in a simulation to either NLS, LS, History or ALS. The total number of users during a simulation was represented by *param.mh*. Each trace data represented one user and the initial experiments were run assuming *param.mh=30*. Later, when examining the stability of the algorithm, some of the traces were replicated to

provide a total of 40 users. In order to achieve a reasonable density of users, *param.mss* was set to 7. Since there is an MSS in every cell, *param.mss* not only specifies the number of MSS in the environment, but also the number of cells.

Using *param.bw*, the available bandwidth was varied to 9.6, 20, 28, 56 and 100 kbps. Low available bandwidth was chosen because it was expected that it is a factor which may impede load sharing, and would like to test to what extent this limitation may hamper load sharing. The highest data rate provided by a GSM call is 9.6 kbps. High-speed circuit-switched data (HSCSD) addresses this constraint by specifying the use of multiple slots for a single call. Each GSM radio carrier can support up to 8 simultaneous calls, each call occupying a single time slot. HSCSD specification allows a call to be allocated up to 8 time-slots, thus providing as much bandwidth as 8 calls. As a result, data rates up to 76 kbps can be supported. A reliable communication link was assumed to exist between the mobile hosts and the MSS, and possible interference or packet loss were not simulated. No attempt was made to economise power consumption by the transceiver.

The next parameter is *param.proc-speed.* In the experiments, it was assumed that mobile hosts have lower processing capacity than fixed hosts. Mobile hosts with low processing capacity are expected to benefit more from load sharing as doing so gives them access to faster machines. A *param.proc-speed* of $1/n$ means that mobile hosts are n times slower than fixed hosts. If *param.proc-speed*$=1/2$, jobs which previously take t seconds to execute, now require $2t$ seconds to execute.

System load on the fixed network is represented by *param.mss-load*, where system load is defined as $\rho=\lambda\mu$, μ=mean service time and λ=job arrival rate. Even though it is acknowledged that mobile hosts may encounter heterogeneous environments, we made a simplifying assumption that extra/external job arrival rates at fixed hosts is homogeneous. If a heterogeneous inter-arrival rate is assumed, it is possible that there are mobile hosts which may be served by fixed hosts with higher spare capacity compared to other mobile hosts. As a result, the percentage of jobs transferred and saving achieved may be influenced by the fixed hosts serving them. By making a homogeneous inter-arrival rate assumption, this possibility is eliminated and allows a fairer comparison of the results obtained among the mobile hosts.

As mentioned earlier, the additional load at fixed hosts are generated to create an environment where mobile hosts have to compete for spare capacity on the fixed network. Previous studies have shown that at low to moderate load ($\rho<0.6$), load sharing improves system performance, while at high load ($\rho\geq0.6$), it is best to disable load sharing as it is difficult, if not impossible, to find a suitable destination host. Performing load sharing under

high loads may lead to performance degradation as precious resources, which are better spent to execute jobs locally, are wasted on load sharing efforts. In initial experiments, system load was maintained at $\rho=0.1$ to allow a reasonable amount of load sharing. Later, when the stability and scalability of load sharing was examined, fixed hosts were put under heavy load to examine the impact of system load on load sharing. However, instead of varying ρ between *0.1* and *0.9*, the system was put under heavy load by concentrating users in one cell to create a heavy demand for the fixed host's spare capacity. The reason for this approach is as follows. Putting the fixed host under heavy load by increasing ρ means that the fixed host have little spare capacity. Under such condition, there really is no action which can ameliorate the situation, and previous studies have shown that the best course of action is to disable load sharing. We are more interested in investigating a situation where a fixed host has spare capacity, but is put under heavy load due to competition from a high number of users. Under these conditions, the load sharing algorithm should be able to react in an appropriate way.

The last two parameters are *param.disk* and *param.probe*. *Param.disk* is set to 1 when running simulations examining the benefit of combining load sharing with a disk spin-down strategy; otherwise, the value is set to 0. *Param.probe* is enabled by setting its value to 1 in order to establish the benefit of delegating transfer requests to other fixed hosts. Delegating transfer requests is disabled when *param.probe=0*. *Param.fixed-hosts* is enabled when *param.probe=1* and specifies the number of fixed hosts, in addition to the MSS, which offers its services to the mobile hosts.

Simulations were run until the battery was flat, which usually took between 2 to 2.5 simulation hours in no load sharing mode and between 3 to 3.5 simulation hours in a load sharing mode.

4.6.3 Experimental Design

An exponential distribution was used to determine the additional job inter-arrival time at fixed hosts and the srand(x) function was used to reset the random number generator to a random starting time. The same value x was used for all experiments so that the same workload is reproduced at the fixed hosts for each experiment.

The experiments were carried out in five stages to investigate factors which influence the benefit of load sharing. In the first two stages, *param.disk* and *param.probe* were set to 0.

Stage 1:

The first set of experiments were carried out in no load sharing (NLS) mode. The values of the parameters were as follows:

Parameter	Values
param.alg	NLS
param.mss	does not have any influence on the result as jobs are not transferred
param.mh	32
param.bw	does not have any influence on the result as jobs are not transferred
param.proc-speed	varied to *1, 1/2, 1/3, 1/4* and *1/5*
param.mss-load	does not have any influence on the result as jobs are not transferred

Stage 2:

Experiments were carried out in load sharing (LS) mode and the influence of available bandwidth, processing power and CPU utilisation were identified.

Experiment 1:

Determine the effect of available bandwidth on load sharing.

Parameter	Values
param.alg	LS
param.mss	7
param.mh	32
param.bw	varied to 9.6, 20, 28, 56 and 100 kbps
param.proc-speed	1/5
param.mss-load	0.1

Experiment 2:

Determines the impact of processing power on load sharing.

Parameter	Values
param.alg	LS
param.mss	7
param.mh	32
param.bw	56 kbps
param.proc-speed	varied to 1, 1/2, 1/3, 1/4 and 1/5
param.mss-load	0.1

Experiment 3:

Determines the influence of CPU utilisation on the gain from load sharing. Battery lifetime improvement the CPU utilisation of each trace was calculated and a graph was plotted to determine if there was any relationship between them.

Experiment 4:

Examined the effectiveness of ALS and History. From this experiment onwards, the values of *param.bw* and *param.proc-speed* were fixed at 56 kbps and 1/5 respectively.

Parameter	Values
param.alg	History, ALS
param.bw	56 kbps
param.proc-speed	1/5
param.mss	7
param.mh	32
param.mss-load	0.1

Stage 3:

Examined the benefit of combining load sharing with a disk spin down algorithm. *Param.disk* was set to 1 and the results were compared to load sharing without a disk spin-down strategy.

Stage 4:

Examines the stability and scalability of the load sharing algorithm.

Experiment 1:

Users were concentrated into one cell to create a heavy demand for the fixed host's spare capacity by setting *param.mss* to 1. Experiments were run using the LS algorithm and two modified load sharing algorithms (discussed in Chapter 6) which were designed to prevent instability.

Parameter	Values
param.alg	LS, Backoff, Slotted LS
param.bw	56 kbps
param.proc-speed	1/5
param.mss	1
param.mh	varied between 1 and 40
param.mss-load	0.1

Experiment 2:

Examines the benefit of delegating transfer requests to other fixed hosts. The parameters were set to the same values as the previous experiments except *param.probe* which was set to 1 and *param.fixed-hosts* which was varied to 2 and 3. Whenever the MSS is unable to accept a transfer request, it sends probes to other fixed hosts to delegate the requests.

Stage 5:

The simulation results were verified by running an emulation of the load sharing algorithm on a wireless LAN. The emulation results were compared to simulation results run under a similar operating condition as the live environment. The parameter values of the simulation were as follows:

Parameter	Values
param.alg	LS
param.bw	2 Mbps
param.proc-speed	1/5
param.mss	1
param.mh	1
param.disk	1

4.6.4 Performance Metrics

Studies discussed in Chapter 3 have used performance metrics which were concerned with measuring performance of the load sharing algorithm as a result of transferring jobs to less loaded hosts in expectation of improved performance. The performance metrics were usually a function of the distribution of workload across the network. Performance was also measured in terms of improved response time.

The metrics chosen to measure the performance of the load sharing algorithm in this study were different from those of previous studies because the distribution of workload on the fixed network is not a primary concern from a mobile host's point of view. The objective is to make use of spare capacity on the fixed network whenever it becomes available in order to conserve battery power. Consequently, the performance metric chosen reflects of this concern. The performance metrics are percentage of battery lifetime improvement, percentage of jobs transferred, response time improvement and percentage of rejected requests.

4.6.5 Data Analysis

For each experiment, the results were compared to a no load sharing case and statistical analysis using either z-test or ANOVA, was carried out at 95% confidence interval to see if there is a significant improvement from the no load sharing case.

Where appropriate, graphs show error bars which express potential error amounts relative to each datum in a data series, calculated at a 5% interval.

4.7 Summary

In this chapter, the assumptions made when designing the load sharing algorithm, and the way in which the assumptions are different from those in previous studies were discussed. This was followed by a discussion on the trace data used in the experiments and the simulation environment; the system parameters and the experimental design were also described. Before proceeding to the next chapter, the hypotheses for this study is once again listed below:

- low bandwidth impedes load sharing as long delays might result in increased response time,
- mobile hosts with low processor power will benefit from load sharing as that gives them access to faster machines on the fixed network,
- it is jobs with long execution time that will bring significant benefit from transfers as they will considerably reduce power consumption by the CPU,
- combining load sharing with another power management strategy should further extend battery lifetime,
- delegating transfer requests will further improve performance as users are given access to other fixed hosts' spare capacity.

The next chapter discusses the first three stages of experiments carried out to identify factors influencing load sharing in wireless networks and the effectiveness of load sharing in conserving battery power.

5. Factors Influencing Load Sharing in Wireless Networks

In this chapter, experiments carried out to determine the factors influencing load sharing are discussed. Five experiments were carried out to investigate:

1. the influence of available bandwidth on job transfers,
2. the influence of a mobile host's processor power on the job transfer decision,
3. the relationship between CPU utilisation and gain from load sharing,
4. the effectiveness of History and ALS compared to LS in extending battery lifetime,
5. the benefit of combining load sharing with a disk spin-down strategy.

For each experiment, the motivations, hypothesis and methodology of the experiment are explained, followed by a discussion of the result obtained and an analysis of the results. For all experiments discussed in this chapter, *param.mss-load* was set to 0.1 to create an environment conducive to load sharing. In the next chapter which addresses the issue of stability and scalability, the impact of putting the fixed hosts under heavy load is examined.

5.1 Bandwidth

Motivation:

As has been mentioned previously, bandwidth is a scarce resource in wireless networks. While bandwidth is much less of an issue on a fixed network, and can consequently be assumed to be so as in most previous studies, bandwidth might become a factor which hampers load sharing in a wireless network. Consequently, we wish to determine the extent to which it impedes load sharing.

Hypothesis:

The discussion in Chapter 3 regarding the effect of communication delays on load sharing came to a conclusion that in the presence of high communication delays, load sharing should be disabled to avoid an increase in job response time. Consequently, it is expected that at low bandwidth, very few jobs can be transferred as the delays involved may cause an increase in job response time. As more bandwidth is available, it is expected that users will be able to transfer more jobs, thus a higher level of saving may be achieved.

Methodology:

Param.bw was varied to 9.6, 20, 28 and 56 kbps. It is expected that slow mobile devices would benefit more from load sharing as they are given access to faster machines. The next experiment will test how true this is, but for this experiment, *param.proc-speed* was set to *1/5*.

Simulations were first run in a no load sharing mode (NLS), then in a load sharing (LS) mode. Simulations were run with 32 users and detailed figures are given for five of those users, together with an average over all users. In this way, it is possible to see both general behaviour and the way that behaviour may vary between different classes of users. The five users were selected to show how the work pattern of a user may influence the gain obtained from load sharing. As will be shown later, given the same set of conditions, users exhibit different behaviour when load sharing was performed. Table 5-1 lists the characteristics of the five users.

Table 5-1
Table showing the characteristics of five users

users	average job size (bytes)	average job execution length (sec)	CPU utilisation
user1	8303	10.17	0.69
user2	30441	11.07	0.89
user3	46154	13.50	0.94
user4	22383	0.73	0.01
user5	4980	0.94	0.02

Results:

Figure 5-1 shows the battery lifetime improvement for different bandwidths. The graph shows that for user1, user2 and user3, battery lifetime improvement increased as the available bandwidth increased, while user4 and user5 did not show significant improvement in battery lifetime. The reason for this is discussed in section 0. Table 5-2 gives a summary of the improvement of battery lifetime of each user.

At low bandwidth, few jobs were transferred because communication delays involved would cause worse response time than if the jobs were executed locally. Furthermore, the amount of time spent transmitting and receiving would consume more power than executing the jobs locally, on the basis of the assumption that a transmitter consumes the same power regardless of bandwidth. As available bandwidth increased, more jobs were transferred, as shown in Figure 5-2 and, therefore, more power saving was achieved. On average, between 20% to 30% jobs transferred brings improvement in battery lifetime.

Figure 5-1

Graph showing the percentage of battery lifetime extension as available bandwidth increases

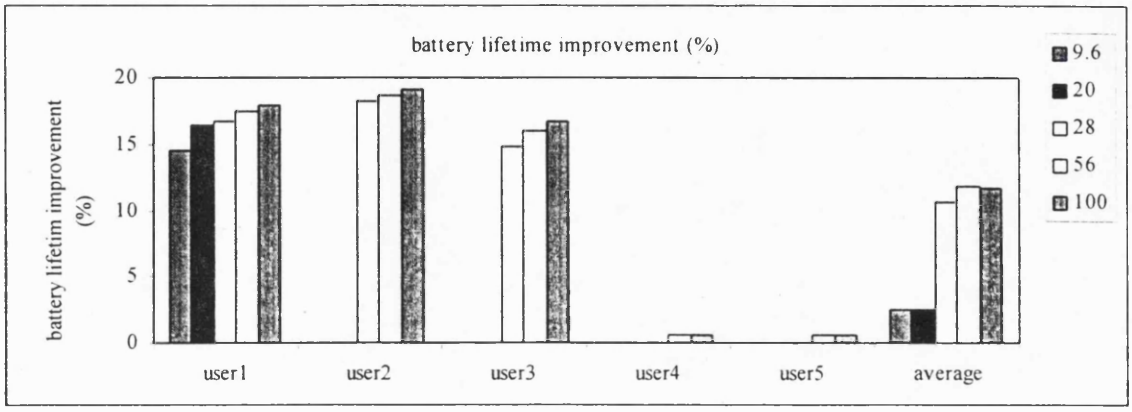


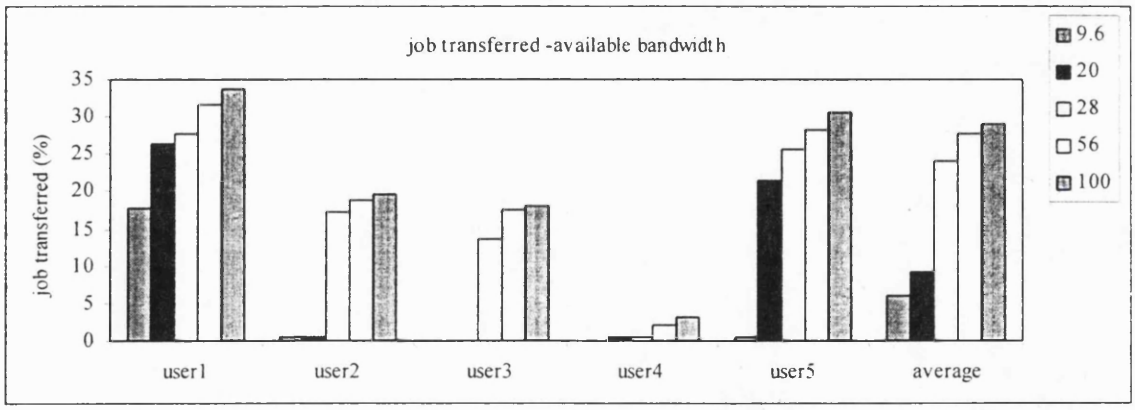
Table 5-2

A summary of battery lifetime improvement for each user for different bandwidths

bandwidth (kbps)	user1		user2		user3		user4		user5		avg. for 32 user
	%	hour	%	hour	%	hour	%	hour	%	hour	%
9.6	15	0.39	0	0	< 1	0.01	0	0	0	0	3
20	16	0.44	0	0	1	0.03	0	0	< 1	0.02	3
28	17	0.45	20	0.51	18	0.46	0	0	< 1	0.02	11
56	18	0.47	21	0.53	19	0.48	< 1	0.02	< 1	0.02	11
100	18	0.48	22	0.55	19	0.50	< 1	0.02	1	0.04	11

Figure 5-2

Graph showing percentage of jobs transferred as the available bandwidth increases



Contrary to the expectation that low bandwidth impedes load sharing, Figure 5-2 shows that user1 transferred a high percentage of jobs compared to other users at low bandwidth. The reason for this is that user1 executes smaller jobs compared to other users, as shown in Table 5-1. Consequently, user1 was able to transfer a high percentage of jobs even at low bandwidth. Interestingly, the graph shows that there is a sudden increase in the percentage of jobs transferred (and hence, battery lifetime improvement) when available bandwidth increased from 20 kbps to 28 kbps. However, when available bandwidth increased to 56 kbps and 100 kbps,

there was only a slight increase in the percentage of jobs transferred. This indicates that there is a limit on the number of jobs which can be transferred; an almost 2x improvement in available bandwidth did not bring a significantly higher increase in the percentage of jobs transferred.

Figure 5-3

Graph showing communication delays vs. available bandwidth

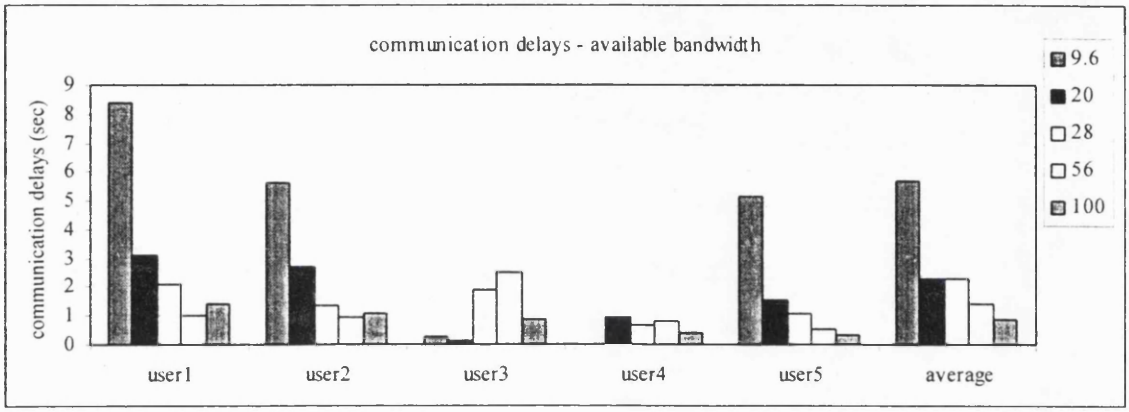


Figure 5-3 shows that communication delays decreased as the available bandwidth increased, as expected. For user1 and user2, the mean communication delays increased slightly when the available bandwidth increased from 56.0 kbps to 100.0 kbps. The same observation can be made for user3, when the available bandwidth increased from 28.0 kbps to 56.0 kbps. The reason for this observation is that, as the available bandwidth increases, jobs which were previously not transferred because it was not feasible to do so, were now selected for transfer. Since these were jobs with relatively large job size, that caused a slight increase in average communication delays.

Analysis of Results:

The results confirm the hypothesis that available bandwidth plays an important factor in determining how many jobs can be transferred. Although the results show that low bandwidth does impede load sharing, it also shows that for a user executing relatively small jobs, load sharing is still possible at low bandwidth. Therefore, to say that low bandwidth impedes load sharing is not necessarily true because, providing the job size is relatively small, it is still possible to transfer a high percentage of jobs, thus extending battery lifetime, as exhibited by user1. This finding is encouraging because it shows that it is still possible to perform load sharing even at low bandwidth. However, contrary to our expectation that a higher level of saving can be achieved as the percentage of jobs transferred increased with an increase in available bandwidth, the results show that this was not the case.

The next step is to analyse if the improvement obtained is statistically significant. Table 5-3 shows the result of z-test carried out at 95% confidence interval on users' battery lifetime with

no load sharing and with load sharing, and the improvement was found to be significant. The maximum battery lifetime improvement achieved is 33 minutes with an average of 20 minutes. In Chapter 1, we stated that we believe load sharing to be especially important in cases where users are using their mobile devices for CPU-intensive jobs, in which case it is imperative battery lifetime is extended to allow users to finish their tasks. Being able to extend battery lifetime for 33 minutes may be critical to allow users to finish the tasks they are performing.

Table 5-3

H_0: mean battery lifetime of NLS and LS are equal (for bandwidth=56 kbps)

z-test; $\alpha = 0.05$	NLS	LS
Mean	2.86	3.18
Known Variance	0.06	0.02
Observations	32	32
Hypothesised Mean Difference	0	
z	-6.39	
P(Z<=z) one-tail	0	
z Critical one-tail	1.64	
P(Z<=z) two-tail	0	
z Critical two-tail	1.96	

5.2 Processor Power of Mobile Computers

Motivation:

It is expected that slow mobile devices are more likely to benefit from load sharing as it gives users access to faster machines on the fixed network. If mobile devices are as powerful as fixed hosts, it is expected that few jobs will be transferred as the delays incurred by the transfer would cause degradation of job response time. In this experiment, the processor power of mobile hosts were varied relative to the fixed host to examine if this is true.

Hypothesis:

Mobile hosts with low processor power ought to gain more benefit from load sharing.

Methodology:

The speed at which mobile hosts processed jobs were varied by varying *param.proc-speed* to *1*, *1/2*, *1/3*, *1/4* and *1/5*; *param.bw* was 56 kbps. *param.proc-speed* of *1/n* means that mobile hosts are *n* time slower than fixed hosts. If *param.proc-speed=1/2*, jobs which previously took *t* seconds to execute, now require *2t* seconds to execute.

Results:

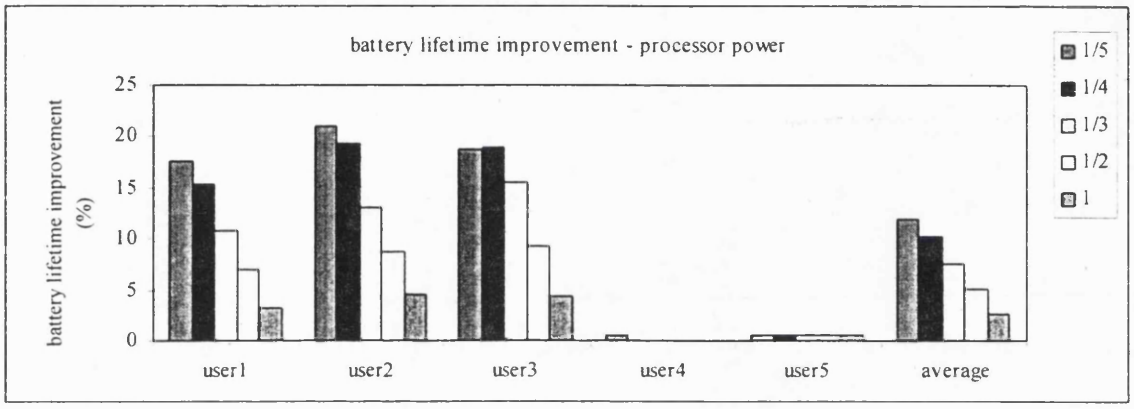
The result shows that when mobile hosts were as fast as the fixed hosts (*param.proc-speed=1*), there is little improvement in battery lifetime because few jobs were transferred. This is because the delay in transferring the jobs would cause increased response time. However, as

the value of *param.proc-speed* decreased, more jobs were transferred because remote execution resulted in better response time in addition to conserving power.

Figure 5-4 shows that for mobile hosts with lower processing power, greater power saving is achieved by transferring jobs. As slower machines require more CPU time to execute a job, the cost of transferring jobs becomes less than the cost of executing them locally. Therefore, on slow machines, more jobs were transferred and a significant amount of power was saved. Once again, user4 and user5 do not exhibit significant improvement in their battery lifetime.

Figure 5-4

Graph showing that slower mobile computers benefit more from job transfers



As transferring jobs gives users access to faster machines, mean response time was improved. The response time improvement was calculated as follows:

$$\text{RT improvement} = ((RT_a - RT_b) / RT_a)\,\%$$

where, RT_a = mean response time without load sharing,
RT_b = mean response time with load sharing.

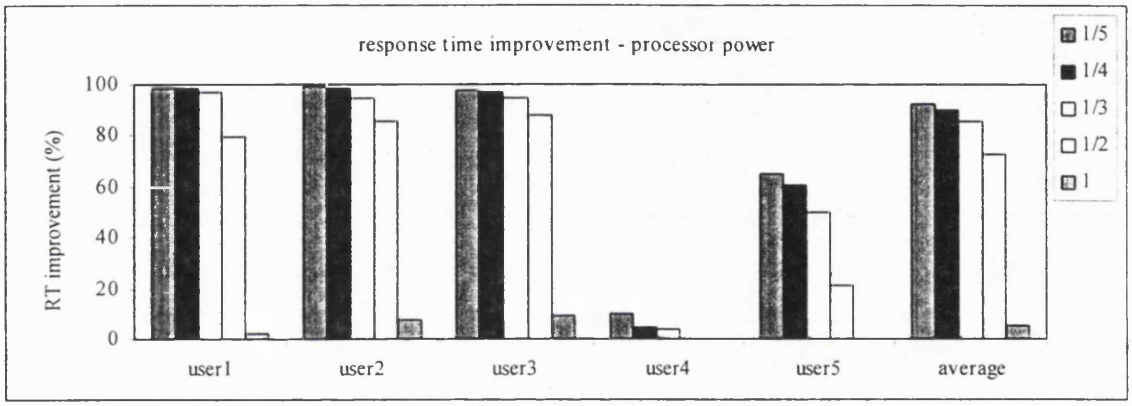
Figure 5-5 shows that there is a vast improvement in response time as the processor cycle decreases. What happens during the simulation is: the job inter-arrival time was calculated from the trace data and as this value remains the same as *param.proc-power* was decreased, jobs still arrived at the same rate even though they were serviced at a slower rate, causing them to wait longer in queue before being served. The long wait occurred especially during periods when there was a burst of activity where job arrival rates were more than 1 jobs/sec. When jobs were transferred to a fixed host, the waiting time was reduced drastically, resulting in the very high improvement in response time when jobs were transferred from the slow machines. The response time improvement from the simulation is very high and we are sceptical that this level of improvement is possible in reality.

Another possible reason for the high response time improvement is the simplification of the *Channel* entity, where a reliable communication medium was assumed, and possible

interference, packet loss and retransmissions were not simulated. In reality, these factors would lead to increased communication delays, and hence much lower response time improvement. The emulation results presented in Chapter 7 exhibit a much lower and more reasonable response time improvement, implying that the simplification of the *Channel* entity might have contributed to the high response time improvement. No attempt was made to simulate these conditions which would also involve economising power utilisation of the transceiver in the presence of such conditions, because the power management strategy used for the transceiver might also have an impact on the overall level of power saving achieved.

Figure 5-5

Graph showing response time improvement by giving mobile hosts access to faster machines on the fixed network



Analysis of Results:

One-way ANOVA was used to ascertain if the battery lifetime improvement is significant as *param.proc-speed* was varied. Table 5-4 shows that there is a significant difference in battery lifetime improvement, showing that mobile devices with low processing power is more likely to benefit from load sharing.

Table 5-4

H_0: mean battery lifetime is equal for different processor power

one-way ANOVA; α=0.05						
Source of Variation	*SS*[1]	*df*[2]	*MS*[3]	*F*	*P-value*	*F critical*
Processor Power	1.94	4	0.48	11.38	0	2.43
Within Groups	6.60	155	0.04			
Total	8.53	159				

[1] sum of square
[2] degree of freedom
[3] mean square

5.3 CPU Utilisation

Motivation:

The results presented in section 5.1 shows that user4 and user5 did not exhibit any improvement of battery lifetime at all. This experiment was carried out to determine if this has to do with CPU utilisation.

Hypothesis:

Users with high CPU utilisation are more likely to benefit from load sharing as that allows the CPU to operate in doze mode more frequently and for longer periods. Users with low CPU utilisation are unlikely to benefit as the CPUs are not heavily utilised in the first place.

Methodology:

The average CPU utilisation for each trace data was calculated and Table 5-1 shows that the CPU utilisation of user4 and user5 are low compared to other users. CPU utilisation for each trace was calculated as follows:

$$\text{CPU utilisation} = \frac{\sum Ji}{T},$$

where J_i = CPU time to execute job i (sec)
T = simulation period (sec)

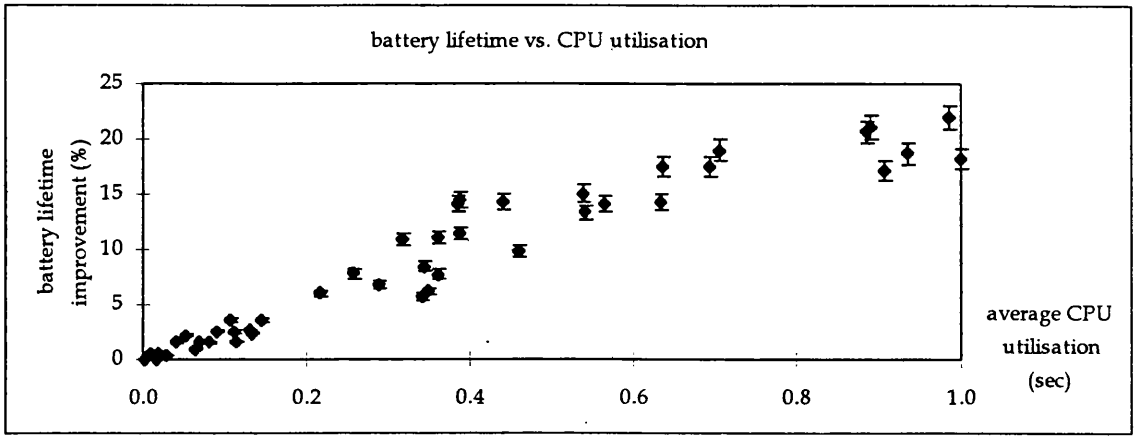
A graph was plotted to determine if there is any relationship between CPU utilisation and battery lifetime improvement.

Analysis of Results:

Figure 5-6 shows that battery lifetime does indeed correspond to CPU utilisation with a correlation coefficient of 0.96, confirming the hypothesis, and explaining why user4 and user5 hardly show any improvement in battery lifetime. The result confirms that it is applications which use the CPU intensively that have much to gain from load sharing.

Figure 5-6

Graph showing the relationship between CPU utilisation and the benefit of job transfer (param.bw=56 kbps; param.proc-power=1/5)



Referring back to Figure 5-2, even though the percentage of jobs transferred for user5 is comparable to user1, user2 and user3, there is no significant improvement in battery lifetime because, since his CPU utilisation is low, transferring jobs did not extend the duration that the CPU remained idle significantly enough to result in substantial saving. The percentage of jobs transferred for user4 was low simply because most jobs were short-lived and, therefore, not worth transferring.

5.4 History and ALS

Motivations:

The experiments done so far were run using the optimal LS algorithm which assumes it has a priori knowledge of the CPU requirement of each job. In reality, it is very unlikely that such information is available in advance. Consequently, an algorithm which makes no assumption of a priori knowledge is required for a real life implementation. In this experiment, two adaptive algorithms which do not assume a priori knowledge of CPU requirements were tested and the results compared to the results of optimal LS. The adaptive algorithms, History and ALS, were described in Chapter 4.

Hypothesis:

As the adaptive algorithms make use of imperfect information, the level of savings achieved by these algorithms might not be as good as the optimal LS. It is expected that the amount of saving will be lower, but hopefully, still brings significant battery lifetime improvement.

Methodology:

Simulations were run with *param.alg* set to ALS and History mode; *param.bw*=56 kbps; *param.proc=1/5*.

Results:

Table 5-5 summarises the simulation results for users using History and ALS compared to LS. From the table, we can see that the average performance of History and ALS were almost as good as the upper bound LS.

Table 5-5

Table comparing the performance of LS, History and ALS

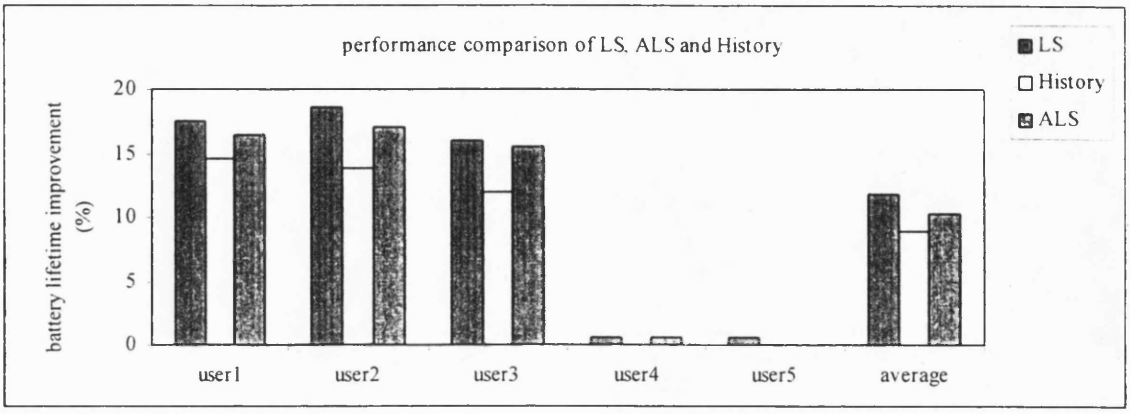
	battery life extended (%)			job transferred (%)			RT improvement (%)		
	LS	History	ALS	LS	History	ALS	LS	History	ALS
user1	15	14	16	31	16	31	98	95	98
user2	21	16	20	25	13	26	98	97	98
user3	19	16	18	27	15	27	97	94	97
average for 32 users	11	9	10	27	16	27	95	81	87

The trace data shows that the execution time of a job is not a constant value, but may fluctuate. Both History and ALS make use of an average CPU time in its calculation. Since an average value is used, it is possible that sometimes an incorrect estimate is made regarding power consumption. A wrong prediction may cause either an unnecessary job transfer, or cause a job to be executed locally when it should have been transferred. Incorrect predictions result in a waste of battery power.

Since History makes use of an average value from a previous simulation in its calculation, the same average value for job A is used in the calculation each time job A is executed. Therefore, an incorrect estimate will be used repeatedly, without consideration of recent behaviour. On the other hand, ALS uses current information, where the average value is updated after each job execution. Consequently, it is capable of adapting its behaviour over time and of improving its estimates. Even though ALS may still make mistakes, these are not as often or as serious as History. Consequently, ALS outperformed History. Figure 5-7 which compares the performance of History and ALS to LS shows that there is little difference in the performance of the algorithms.

Figure 5-7

Graphs showing the performance of History and ALS compared to LS



Analysis of Results:

Even though it was expected that History and ALS would not perform as well as the optimal LS, the results show that their performance is not much different from optimal LS. An ANOVA analysis was carried out to determine if there is a significant difference in the performance of the three algorithms, and the result in Table 5-6 shows that there is no significant difference between their performances, implying that History and ALS perform as well as the optimal LS.

The adaptive algorithms performed better than expected, which is very encouraging as it shows that the adaptive algorithms were able to make good predictions of CPU requirements. A good prediction is important for a real implementation where CPU requirements are not known in advance.

Table 5-6

H_0: mean battery lifetime of NLS, History and ALS are equal.

one-way ANOVA; $\alpha = 0.05$						
Source of Variation	*SS*	*Df*	*MS*	*F*	*P-value*	*F critical*
Algorithms	0.11	2	0.05	2.47	0.09	3.09
Within Groups	2.00	93	0.02			
Total	2.11	95				

5.5 Combining Load Sharing with a Disk Spin-Down Strategy

Motivation:

In Chapter 2, various power management strategies were discussed. It is only practical that a real life implementation would combine various strategies to prolong battery lifetime as much as possible. In this experiment, we would like to examine the level of saving possible by combining load sharing with a disk spin-down strategy.

Hypothesis:

Combining two power management strategies ought to further extend battery lifetime compared to using load sharing alone.

Methodology:

Disk trace data obtained from Hewlett-Packard were collected from three computer systems running the HP-UX operating systems. Cello was a time sharing system used by a group of researchers at HP, Snake was a file server which served nine clients at the University of California, Berkeley, and hplajw was a disk on a personal workstation which was mainly used for electronic mail and editing papers. Trace data from hplajw was selected for this simulation because since as it was obtained from a personal workstation, it probably better reflects the type of activities carried out by a user of a mobile computer. Details of the disk access patterns are discussed in [Ruem92].

A number of disk spin-down strategies are discussed in Chapter 2. Since hplajw has a buffer cache associated with it, the access frequency ought to be less than if there was no cache. As a result, it should be possible to spin down the disk more vigorously than when there is no cache. Consequently, the strategy proposed by Li et al [Li94] (discussed in section 2.3.1) was chosen, where a timeout value of 2 seconds and a disk critical rate, R_{CR}, of 6 seconds were assumed. However, unlike the study carried by Li et al, the access latency was not measured because the use of buffer cache is an established technique to improve disk access time. The study of Li et al also confirms this fact and, therefore, it is unnecessary to examine this issue any further.

A new entity, *Disk* entity, was introduced for this experiment. There is one *Disk* entity for each *MH* entity. Its function is to read a disk trace and simulate disk accesses, spin-downs, and spin-ups. It also keeps a record of how long a disk remains active and idle, and the number of spin-ups and spin-downs to calculate power consumption. Every 60 seconds, it sends a message to the *MH* entity informing it of the duration the disk is idle and active. This information is used in part of the calculation to determine the remaining power on the mobile host. The *Disk* entity is instantiated by the *MH* entity if *param.disk* is *1*. Once instantiated, the entity runs independently from the *MH* entity. The only communication between the two entities is the message sent by the *Disk* entity informing *MH* entity of the duration the disk is idle/active.

One flaw of the simulation combining disk spin-down and load sharing is due to the fact that the traces were obtained from two independent sources. Consequently, there is no way of associating disk accesses with user jobs, and simulation of disk accesses was run independently of user jobs. It is also acknowledged that it is not possible to ascertain if the trace accurately represent disk access pattern on a mobile device. However, like the argument presented in

Chapter 4 regarding the use of trace data, since people are not yet using mobile applications on a regular basis, it is not possible to obtain a disk trace for mobile applications. Using a disk trace from existing applications is the closest approximation to modelling disk access realistically. In spite of these shortcomings, we expect that the results obtained are indicative of the amount of saving possible when different power management strategies are combined.

Results:

Figure 5-8 compares battery life improvement using load sharing and load sharing with disk spin-down (LS+SD) policy, while Table 5-7 summarises the result obtained. As expected, combining load sharing and a disk spin-down policy further improved battery lifetime. In the best case, battery lifetime was extended by about 33% or 50 minutes. Combining a disk spin-down strategy further extended battery lifetime by approximately 3-11%, with an average improvement of about 8%. No strong claims can be made as a result of this experiment, however, there is some indication that extra benefit can be obtained. Table 5-8 shows the result of the z-test.

Figure 5-8

Graph showing battery lifetime improvement using load sharing and load sharing combined with a disk spin-down policy

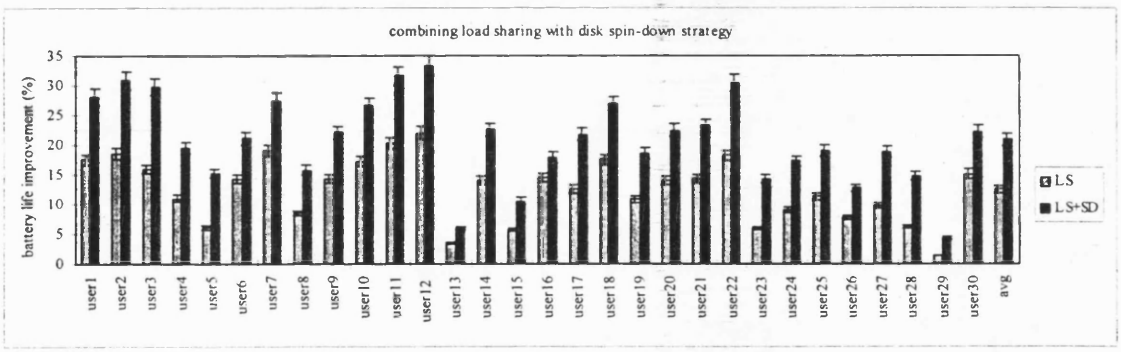


Table 5-7

Table summarising results of load sharing and load sharing with disk spin-down

	improvement (%)		improvement (hour)	
	LS	LS+SD	LS	LS+SD
Maximum	22	33	0.55	0.83
minimum	1	4	0.05	0.15
average	13	21	0.35	0.57
standard deviation	5	7	0.13	0.17

Table 5-8

H_0: mean battery lifetime of LS and LS+SD are equal.

z-test; α=0.05	LS	LS+SD
Mean	3.18	3.41
Known Variance	0.02	0.01
Observations	30	30
Hypothesized Mean Difference	0	
z	-7.46	
P(Z<=z) one-tail	0	
z Critical one-tail	1.64	
P(Z<=z) two-tail	0	
z Critical two-tail	1.96	

5.6 Conclusion

Results of experiments presented in this chapter indicate that load sharing is a potentially effective power management strategy, forming an orthogonal approach to those already in use, and enhancing their benefits. Transferring jobs from mobile hosts for remote execution has been shown to be successful in extending battery lifetime and improving job response time, subject to some constraints.

The benefits of remote execution depend on available bandwidth, CPU utilisation and processor power of the mobile relative to the fixed host. While generally, more jobs are transferred as available bandwidth increases, mobile hosts with small jobs were found to be able to transfer a considerable number of jobs even at low bandwidth. This finding is encouraging because it shows that load sharing is still possible at low bandwidth, subject to the constraint of job size. A slightly unexpected finding is that the percentage of jobs transferred does not increase linearly as available bandwidth increases. This results from the fact that the level of possible saving is limited by the number of jobs which can be transferred. On average, between 20% to 30% jobs were transferred, and this is enough to bring significant battery lifetime improvement. Previous studies have shown that less than 20% job movement is required to improve system performance.

CPU utilisation was established to be an important factor influencing the benefit of remote execution. Mobile hosts with high CPU utilisation benefit more from remote execution as doing so allows the CPU to operate in doze mode more often, thus extending battery lifetime significantly. This implies that mobile applications which are likely to benefit from load sharing are those which perform heavy computations and uses the CPU intensively, e.g. spreadsheet applications and program compilation.

Mobile hosts with low processing power were found to benefit from remote execution as this assured them access to a faster machine, resulting in lower response time, in addition to

reducing power consumption by the CPU. If mobile hosts have limited processor power, load sharing will have its role in giving access to fast machines, in addition to conserving power.

The two adaptive algorithms, History and ALS, were found to perform as well as LS in conserving power. Since they make no assumption of a priori knowledge of CPU requirements, History and ALS offer a more practical approach when load sharing is implemented in a real environment. Although it is surprising to find the adaptive algorithms performing as well as the optimal algorithm, this is encouraging as it indicates that a performance which is close to an optimal performance might be possible in a real life implementation.

As expected, combining load sharing with a disk spin down strategy further extends battery lifetime. It is only practical that power management strategy on mobile devices combines various power saving techniques. There is no reason why load sharing cannot be combined with existing power management techniques to further improve battery lifetime.

Having established that load sharing does extend battery lifetime, the next chapter examines the stability and scalability of the load sharing algorithm.

6. Stability and Scalability of the Load Sharing Algorithm

In Chapter 2, the importance of mobile applications having the ability to adapt its behaviour according to its environment in order to make the best use of available resources was discussed. In the context of this study, the adaptive ability is important not only to make the best use of available resources, but also to prevent instability.

In section 3.3 of Chapter 3, the issue of stability was discussed. Depending on the design of an algorithm, various factors have been identified to prevent unstable behaviour of the algorithm. The discussion also illustrated that the definition of stable behaviour depends on the environment in which an algorithm operates. In the context of this study, the load sharing algorithm is considered stable if, under unfavourable conditions, it does not cause power consumption to be worse than when there is no load sharing. Unfavourable condition, in this case, occurs when there is a high number of users competing for the spare capacity on the fixed network. In this case, it is crucial that mobile hosts do not continue to send transfer requests when they are unlikely to be accepted, as doing so would only waste precious battery power. A method of disabling the algorithm under such circumstances is required.

In order to test the stability of the LS algorithm, the contention at a fixed host was simulated by setting the parameter *param.mss=1* and increasing the value of *param.mh* gradually. It is increased in steps of *1* until *param.mh=10*, and then in steps of *5* until *param.mh=40*. By doing so, mobile hosts flood the MSS with transfer requests, creating a heavy demand for its processor cycles. Figure 6-1 shows that the performance of LS degrades steadily as the number of users increases and Figure 6-2 shows users for whom load sharing caused worse power consumption than no load sharing. Figure 6-3 shows that the percentage of jobs transferred decreased as the total number of users increased, while Figure 6-4 shows that the percentage of rejected requests increased.

As a considerable amount of power is consumed in transmitting/receiving messages, a rejected request is considered a performance penalty. The percentage of rejected requests is used as a parameter to measure the ability of the load sharing algorithm to adapt itself to current condition, where the number of rejected requests should be kept to a minimum. The existing LS algorithm is modified to prevent instability by disabling it when it is no longer feasible to transfer jobs.

Figure 6-1

Graph showing the performance of LS degrades as the number of users in a cell increases.

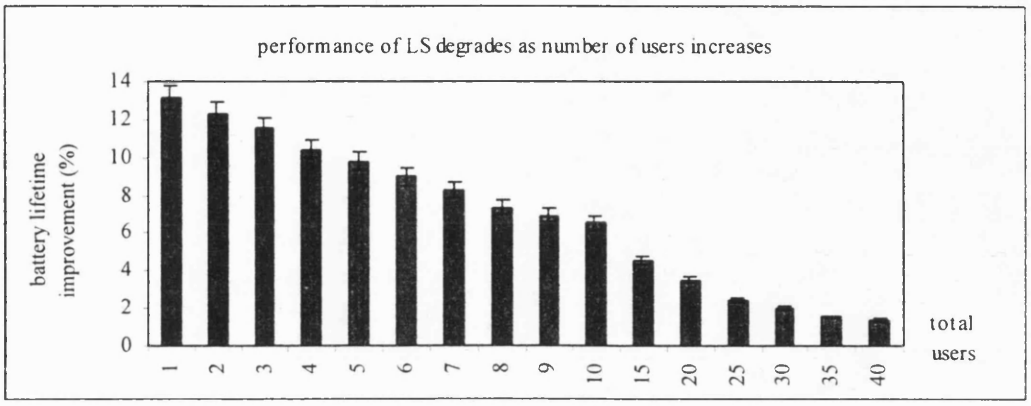


Figure 6-2

Graph showing users for whom performance of LS degrades beyond no load sharing.

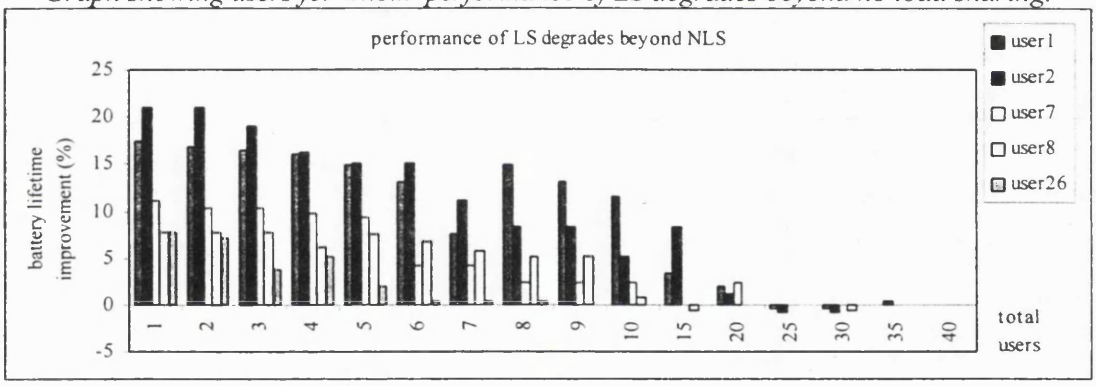


Figure 6-3

Graph showing the percentage of jobs transferred decreases as total users in a cell increases.

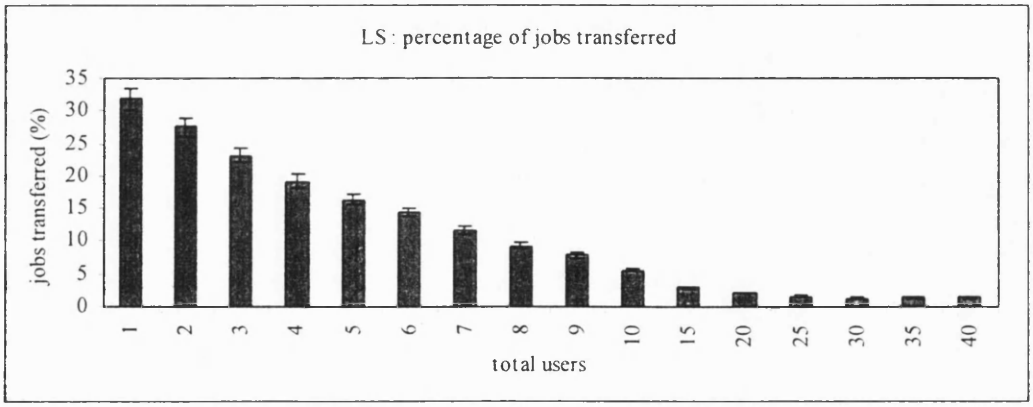
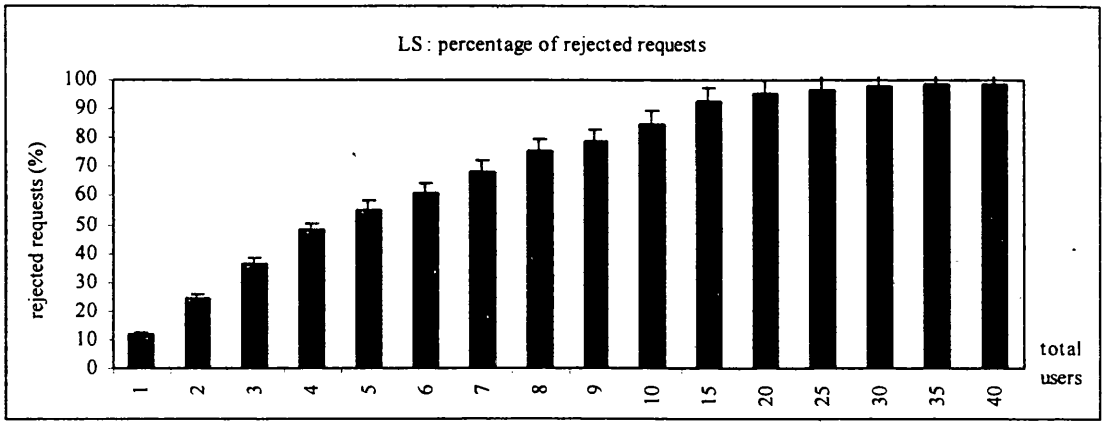


Figure 6-4

Graph showing the percentage of requests rejected increases as total users in a cell increases.



Note that even if the algorithm is successful in detecting when load sharing is no longer feasible and disables itself accordingly, that may not further improve battery lifetime. As more mobile hosts compete for the fixed host's processor cycle, spare capacity becomes more scarce, thus not many jobs can be transferred for remote execution. What is important is to prevent mobile hosts wasting precious battery power by sending fruitless transfer requests.

Another important issue which we attempt to address is scalability. Kremien and Kramer [Krem92] considered stability as a pre-condition to scalability. Once the issue of stability is addressed, how well the load sharing algorithm scales is examined.

The following two sections describe the modified algorithms, Backoff and Slotted LS, and examine their effectiveness in disabling themselves in order to avoid instability.

6.1 Backoff Algorithm

Motivation:

The load sharing algorithm should disable itself during periods when its transfer requests are unlikely to be accepted to avoid wasting power. A method is required to bar mobile hosts from sending requests when fixed hosts are busy. This experiment tests a Backoff algorithm where a busy MSS specifies a backoff period during which a mobile host must refrain from sending requests.

Hypothesis:

The Backoff algorithm is expected to reduce the percentage of rejected requests.

Methodology:

Each time a transfer request is rejected, a mobile host backoffs for a period of time known as a *backoff period* during which jobs are executed locally. The backoff period is specified

by the MSS, where each time a MSS rejects a transfer request, it specifies a backoff period in its reply. The backoff period is calculated based on the CPU requirement of jobs it has in its queue and the processor cycles reserved by transfer requests it has previously accepted. The underlying assumption is the MSS knows how much CPU is required to execute the jobs in its queue. In reality, this information might not be available, in which case an estimate must be used.

$$backoff\ period = \sum J_i + \sum R_j$$

where J_i = execution time of job i in queue
R_j = processor cycle reserved for job j

The mobile host abstains from sending transfer requests during this period and only starts sending requests after the period expires.

Results:

There appears to be very little difference in battery lifetime improvement of Backoff to LS, as shown in Figure 6-5, while Figure 6-6 shows that the percentage of rejected requests of Backoff was lower than LS. Figure 6-7 shows that there was only a small difference in response time improvement of LS and Backoff.

Figure 6-5
Graph comparing the performance of Backoff to LS.

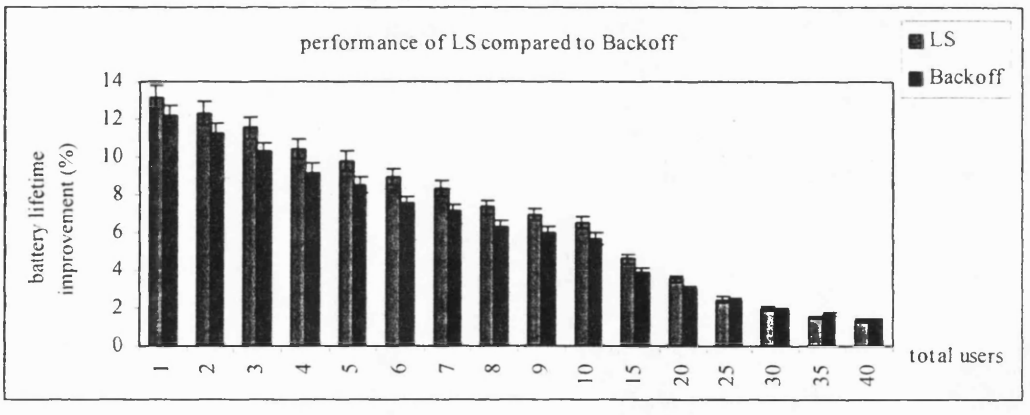


Figure 6-6

Graph comparing the percentage of rejected requests of LS and Backoff.

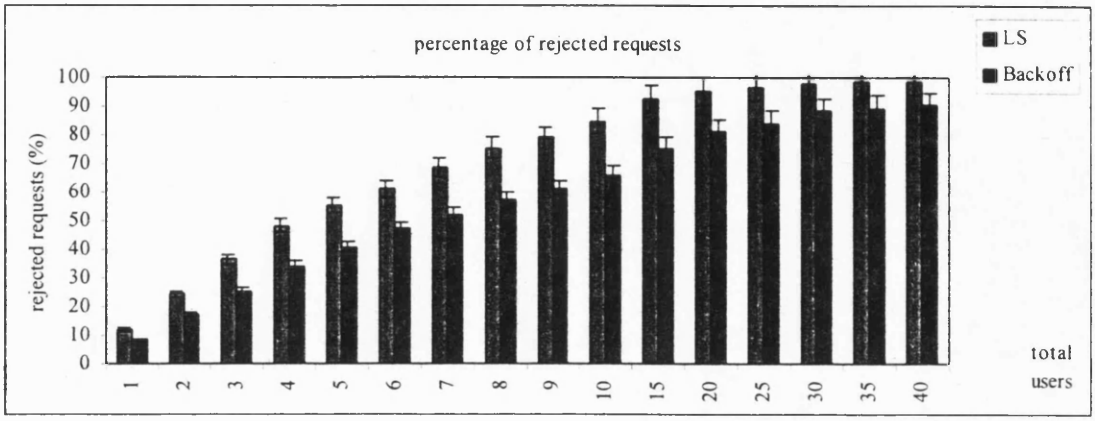
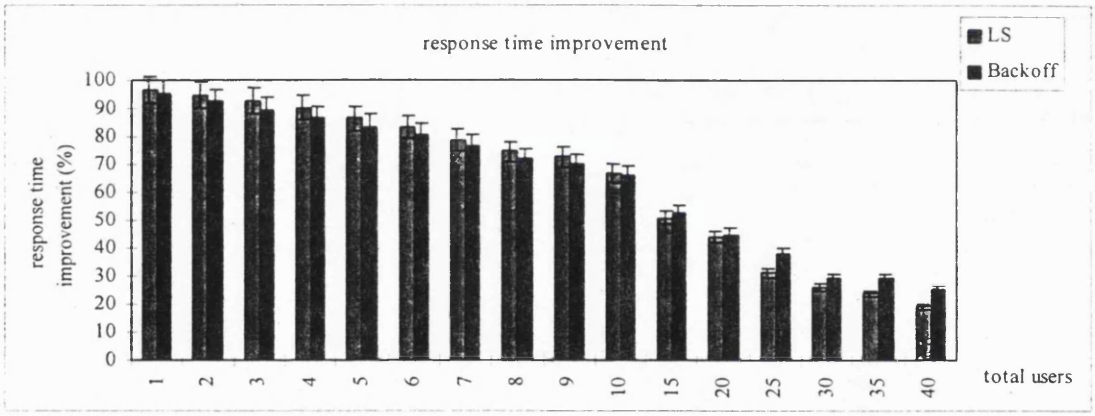


Figure 6-7

Graph comparing response time improvement of LS and Backoff



Analysis of Results:

Analysis on the results was carried out using two-way ANOVA with replication. The hypothesis tests determine if there is a significant difference in the percentage of rejected requests as the number of users increase, and if there is a significant difference in the percentage of rejected requests between the LS and Backoff algorithms. The result of the analysis in Table 6-1 shows that there is a significant difference in the percentage of rejected requests as total users increases. It also shows that there is a significant difference in the percentage of rejected requests of Backoff and LS, indicating that Backoff successfully disabled itself when load sharing was not feasible. Even though Backoff successfully disabled itself, the percentage of rejected requests was still high.

Table 6-1

H_0 : mean rejected requests as total users increases are equal;
H_0 : mean rejected requests of Backoff and LS are equal.

two-way ANOVA with replication; $\alpha = 0.05$						
Source of Variation	*SS*[1]	*df*[2]	*MS*[3]	*F*	*P-value*	*F critical*
Total Users	904191.73	15	60279.45	436.45	0	1.67
Algorithms	52986.82	1	52986.82	383.64	0	3.85
Interaction	5937.28	15	395.82	2.87	0	1.67
Within	172367.07	1248	138.11			
Total	1135482.89	1279				

The Slotted LS algorithm was designed in an attempt to examine if it is possible to:

- further reduce the percentage of rejected requests, and
- limit competition among mobile hosts so that spare capacity can be allocated more fairly.

The performance of Slotted LS is compared to Backoff and LS.

6.2 Slotted LS Algorithm

Motivation:

Even though Backoff reduced the percentage of rejected requests, the percentage was still high. Therefore, a more effective method is required to disable the load sharing algorithm. This experiment tests an algorithm which limits the number of users granted permission to send transfer requests, and they are only granted the permission for a limited period. Once that period expires, permission is granted to a different set of users.

Hypothesis:

By limiting the number of users granted permission to send transfer requests, competition for spare capacity is limited to a small set of users. Limiting the period selected users are allowed to send transfer requests ought to allocate spare capacity more fairly among mobile users. As a result, the chance that a transfer request has of being accepted is increased. When that period expires, users execute their jobs locally until they are once again granted permission. This approach is expected to result in better performance than Backoff.

Methodology:

Since the instability is caused by a high number of users requesting transfers, one way to overcome this problem might be by limiting the number of users allowed to send transfer requests. The MSS selects N mobile hosts, out of a total of M mobile hosts, which have requested permission to transfer jobs in a round robin fashion and grants them permission to

[1] sum of squares.
[2] degrees of freedom

send transfer requests for period T. The period T is termed a slot. The MSS sends a message to the selected mobile hosts informing them that they have been granted access for the next T seconds. During that period, selected mobile hosts wishing to transfer a job send a transfer request as usual, and the request is accepted if the MSS is able to execute the job in the specified time. Any requests sent within that period that have been accepted are executed by the MSS even if the period has expired. When there are fewer than N mobile hosts in a cell, the algorithm behaves like the unmodified LS algorithm.

The following example demonstrates how the algorithm works. A mobile host, mh_i, sends a message to the MSS at time $t=0$ stating its interest to transfer jobs. If there are fewer than N mobile hosts requesting permission, Slotted LS behaves like the LS algorithm: mh_i is granted permission and no time limit is imposed on the duration mh_i is allowed to send requests. If there are M mobile hosts requesting permission, where $M>N$, N mobile hosts are selected to be granted permission on a first-come-first-served basis. The other $(M-N)$ mobile hosts are put in queue and are granted permission after the first slot expires.

Assuming that there are M mobile hosts requesting permission to send transfer requests and mh_i is one of N mobile hosts selected for the first slot, the MSS sends a message to mh_i to indicate that permission to send transfer requests has been granted. At time $t=1$, mh_i receives a reply informing it that it has been granted permission to send requests for the next T seconds. Let us assume that $T=30$. During that period, if mh_i has a job it wishes to transfer, it sends a transfer request as usual. If the request is accepted, processor cycles are reserved for that job and mh_i is informed that the request is accepted. Any requests accepted within period T will be executed. If a request is accepted at $t=30$ and the job arrives at $t=31$, the job is still executed by the MSS because processor cycles have been allocated for that job. If another mobile host, mh_j, also sends a message to the MSS at time $t=0$ stating its interest to transfer jobs, but N mobile hosts have already been selected for the current slot, mh_j will only be informed that it is granted permission to send transfer requests at time $t=31$. mh_j can then starts sending transfer requests between $t=31$ and $t=60$. mh_i might not always send transfer requests during the period T, but this is not a problem as that would increase the possibility of other selected users having their requests accepted. mh_i is not depriving other selected mobile hosts from being allocated the spare capacity on the fixed host.

The next step is identifying suitable values for N and T. The value of N was chosen to limit the number of users so that good performance can be achieved. Figure 6-1 shows that LS performs reasonably well when there were fewer than 15 users in a cell, thus the simulations were run for N values of 5 and 10. A suitable value for T must also be determined. The

---

[3] mean square

trace data shows that jobs arrival usually occurs in bursts, and the burst could last for a few or several seconds. The interval between bursts varies from less than 60 seconds to a few minutes. We would like the value T to be long enough to include the time span of a burst, but not so long so that spare capacity is allocated unfairly among users. Based on the trend shown in the trace data, values of $T=30$ and $T=60$ were chosen. Experiments were, therefore, run with the following parameter values:

- Experiment 1: $N=5$, $T=30$
- Experiment 2: $N=10$, $T=60$
- Experiment 3: $N=10$, $T=30$

Results:

When the total number of users is fewer than N, Slotted LS behaves in a similar way to the unmodified LS algorithm because the MSS starts limiting the number of users granted permission to request transfers only when total users > N. Figure 6-8 shows that there is little difference in the performance of Slotted LS (especially for Experiments 2 and 3) and LS when the total users $\leq N$, but there is a marked difference in the performance of Slotted LS (for Experiments 2 and 3) compared to LS once total users > N. In fact, Figure 6-8 shows that once the algorithm started operating in a Slotted LS mode, there is an increase in battery lifetime improvement. Limiting the number of users granted permission to request transfers resulted in less contention for the spare capacity on the fixed host, and increased the possibility of a request being accepted. As a result, the percentage of jobs transferred increased, leading to a higher battery lifetime improvement. Figure 6-8 shows that Slotted LS (for Experiments 2 and 3) performed better than LS and Backoff as the number of users competing for the fixed host's processor cycles increases. Slotted LS resulted in higher battery lifetime improvement because:

- more jobs were transferred during periods when a mobile host is granted permission to request transfers, and
- by reducing the percentage of rejected request, Slotted LS prevented mobile hosts from wasting battery power.

The reason why there is not much improvement for Experiment 1 is explained later.

Table 6-2 shows that once the total number of users exceeded N, and the algorithm started operating in a Slotted LS mode, the percentage of rejected requests was kept to a minimum, and dropped to less than 1%. The percentage was kept to a minimum by the following factors:

- users who were not granted permission to request transfers operate in a no load sharing mode, which means the overall number of requests sent was reduced,
- there was less contention for the fixed host's spare capacity, and the period T, during which mobile hosts were granted permission to request transfers, was short enough that very few requests were rejected during that period. This is different from LS and Backoff where the number of rejected requests is accumulated during the duration of the simulation.

Table 6-2

Table showing percentage of rejected requests for LS, Backoff and Slotted LS

total users	LS	Backoff	Slotted LS		
			Expt. 1	Expt. 2	Expt. 3
1	11.91	8.53	11.91	11.91	11.91
2	24.36	17.20	26.54	26.54	26.54
3	36.48	25.56	36.27	36.27	36.27
4	48.10	33.99	49.58	49.58	49.58
5	55.19	40.52	58.16	58.16	58.16
6	61.19	47.17	1.94	64.51	64.51
7	68.50	52.27	0.07	68.74	68.74
8	75.60	57.20	0.07	76.50	76.50
9	79.05	61.24	0.08	81.67	81.67
10	84.85	66.16	0.08	85.47	85.47
15	92.60	75.65	0.13	0.15	0.12
20	95.24	81.25	0.18	0.14	0.13
25	96.91	84.32	0.19	0.23	0.23
30	98.15	88.44	0.20	0.23	0.19
35	98.38	89.27	0.19	0.25	0.30
40	98.62	90.46	0.27	0.28	0.26

Figure 6-9 shows that the response time improvement of Slotted LS was much lower than the improvement shown by LS and Backoff when $N=5$, but was not much different from LS and Backoff when $N=10$.

Figure 6-8

Graph comparing the average performance of Slotted LS, Backoff and LS.

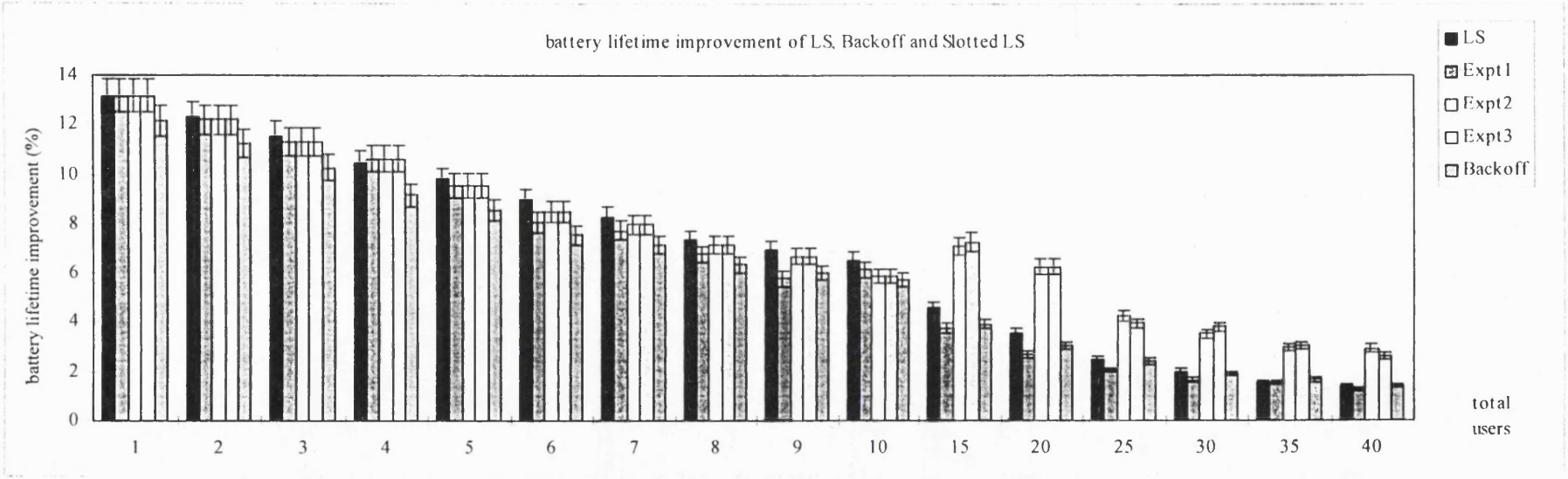
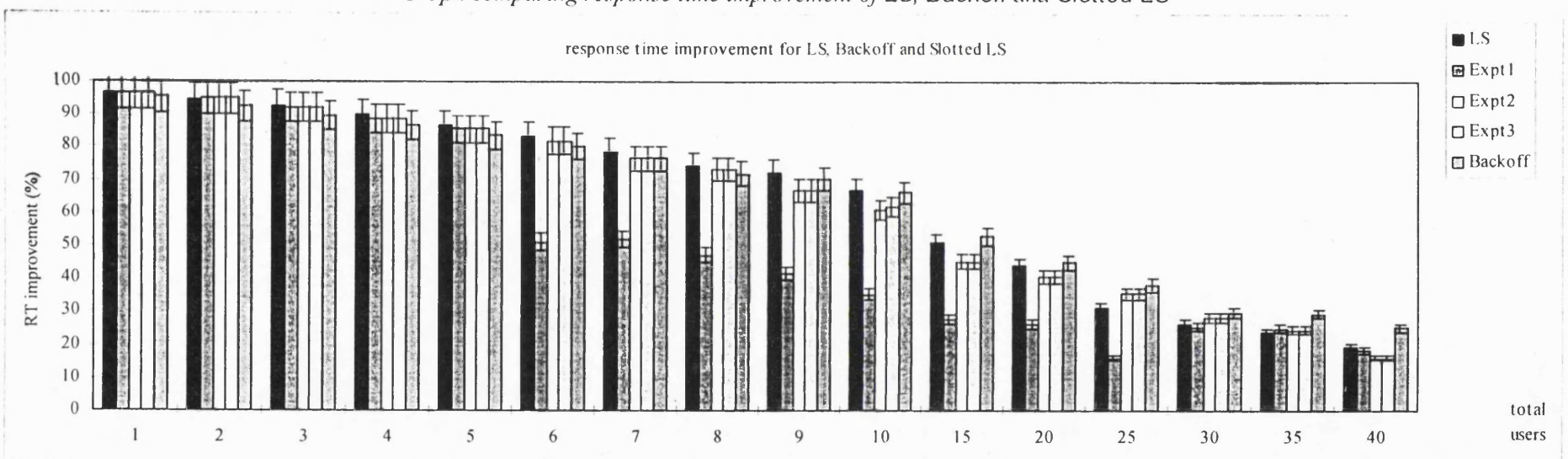


Figure 6-9

Graph comparing response time improvement of LS, Backoff and Slotted LS



Analysis of Results:

Analysis of the results were carried out to determine:

- if the drop in the percentage of rejected requests is significant,
- whether there is a significant improvement in battery lifetime between Backoff and Slotted LS,
- if the value of parameters N and T have a significant impact on the performance of Slotted LS.

Analysis was carried out using two-way ANOVA with replication, in a similar way as when testing Backoff and LS. As expected, Table 6-3 (a) and (b) confirms that there was a significant reduction in the percentage of rejected requests when using Slotted LS for Experiment 1 and Experiment 2, respectively. Analysis was not carried out for Experiment 3 as its performance was almost identical to Experiment 2.

Table 6-3

(a)

H_0: mean rejected requests as total users increases are equal;
H_0 :mean rejected requests of LS and Experiment 1 are equal.

two-way ANOVA with replication; $\alpha = 0.05$						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critical*
Total Users	152186	15	10145.76	88.23	0	1.67
Algorithms	1102779	1	1102779.01	9590.36	0	3.85
Interaction	560942	15	37396.15	325.22	0	1.67
Within	143505	1248	114.99			
Total	1959413	1279				

(b)

H_0: mean rejected requests as total users increases are equal;
H_0 :mean rejected requests of LS and Experiment 2 are equal.

two-way ANOVA with replication; $\alpha = 0.05$						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critical*
Total Users	443238	15	29549.22	211.98	0	1.67
Algorithms	398309	1	398309.10	2857.33	0	3.85
Interaction	718935	15	47929.04	343.83	0	1.67
Within	173970	1248	139.4			
Total	1734453	1279				

Referring to Figure 6-8, Slotted LS appears to perform better than Backoff as the number of users increases, especially in the case of Experiment 2 and Experiment 3. Further analysis shows that Experiment 2 really did perform better than Backoff (see Table 6-4).

Table 6-4

H_0 : mean battery lifetime as total users increases are equal;
H_0 : mean battery lifetime of Experiment 2 and Backoff are equal.

two-way ANOVA with replication; $\alpha = 0.05$						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critical*
Total Users	13545.94	15	903.06	68.94	0	1.67
Algorithms	574.75	1	574.75	43.88	0	3.85
Interaction	201.02	15	13.40	1.02	0.43	1.67
Within	16347.97	1248	13.10			
Total	30669.68	1279				

Having established that Slotted LS is more effective than Backoff in avoiding unnecessary transfer requests, we now proceed to establish if the parameter *N* and *T* have a significant impact on the performance of the algorithm. From the graph in Figure 6-8, Experiment 1 does not seem to perform as well as Experiments 2 and 3 in extending battery lifetime. Table 6-5 confirms that the value *N* does have a significant influence in determining how well Slotted LS performs: *N=10* extends battery lifetime more than *N=5*. The reason for this is that, if the value of *N* is too small, the load sharing algorithm is unable to make full use of the spare capacity at the MSS, resulting in poorer overall performance. Performance appears to be relatively insensitive to variation in *T* as shown by Figure 6-8, where there is little difference in the performance of Slotted LS for Experiment 2 and Experiment 3.

The results show that by choosing suitable values for *N* and *T*, Slotted LS was not only effective in reducing the percentage of rejected requests, but also performed better than Backoff as contention for the fixed host's processor cycles increased. Slotted LS was also able to distribute spare capacity at the MSS more fairly among mobile hosts, resulting in better battery lifetime improvement than that achieved by Backoff.

Table 6-5

H_0 : mean battery lifetime as total users increases are equal;
H_0 : mean battery lifetime of Experiment 1 and Experiment 2 are equal.

two-way ANOVA with replication; $\alpha = 0.05$						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critical*
Total Users	11.86	15	0.79	23.58	0	1.67
Algorithms	0.22	1	0.22	6.71	0.01	3.85
Interaction	0.34	15	0.02	0.67	0.82	1.67
Within	41.84	1248	0.03			
Total	54.27	1279				

6.3 Delegating Job Transfer Requests

Motivation:

In the experiments discussed in this chapter so far, it was assumed that there was only one fixed host catering for the transfer requests. We have shown that load sharing causes performance degradation if proper precautions are not taken when there is a high number of users competing for spare capacity. It is not practical to expect only one fixed host to cater for job transfer requests, as it might not be able to cope with the demand. Also, relying on one fixed host would not allow the algorithm to scale to cater for a high number of users. Slotted LS has been shown to be effective in preventing performance degradation and allocating spare capacity more fairly among users. Nonetheless, if there is only one fixed host catering for a high number of users, performance degrades steadily and load sharing will eventually cease to be effective. One way to alleviate this bottleneck is to have the MSS delegate transfer requests to other fixed hosts. Experiments were carried out with the MSS sending probes to other fixed hosts, to test if that would improve performance.

Hypothesis:

Delegating job transfers to other fixed hosts ought to lead to an increase in the percentage of jobs transferred and, hence, battery lifetime improvement.

Methodology:

P other fixed hosts were assumed to offer its services to mobile hosts. When the MSS receives a transfer request and is unable to execute the job within the specified time, it sends a probe to *P* other fixed hosts requesting if the host could execute the job on its behalf. If the probed host is able to execute the job within the specified time, it accepts the request and processor cycles are reserved for the job. The fixed hosts were assumed to be connected by high speed links and communication delays involved in sending the probes were assumed to be negligible, based on the approach taken by [Mirc89] and [Mirc90] which established that if the size of probes is significantly smaller than the size of jobs, communication delays incurred by probes are negligible. A mobile host requesting a job transfer is not aware of the probes sent by the MSS and has no knowledge of the actual location where its job is executed.

Simulations were run with the LS algorithm sending probes to $P=2$ and $P=3$ hosts. Simulations were also run combining probing and Slotted LS (Slotted Probe) with $P=2$, $N=25$ and $T=60$ to see if it results in a more stable behaviour.

Results:

Figure 6-10 shows that the performance of load sharing with probes increased tremendously. The battery lifetime improvement did not degrade as rapidly as when load sharing was run without probes. As expected, Figure 6-11 shows that percentage of rejected requests was far lower when transfer requests were delegated. Note than even when no request was rejected, battery lifetime did not improve beyond 20%, again confirming that the improvement is limited by the number of jobs which can be transferred. The graph also shows that the percentage of rejected requests increased rapidly when there were more than 25 users, indicating high contention for the fixed hosts' resources. Even though the percentage of rejected requests was much lower than LS in the beginning, it increased rapidly once the spare capacity at the fixed hosts were almost fully utilised. When total users were 30, the percentage of rejected requests were almost as high as LS.

Figure 6-10

Graph comparing the performance of load sharing with probes.

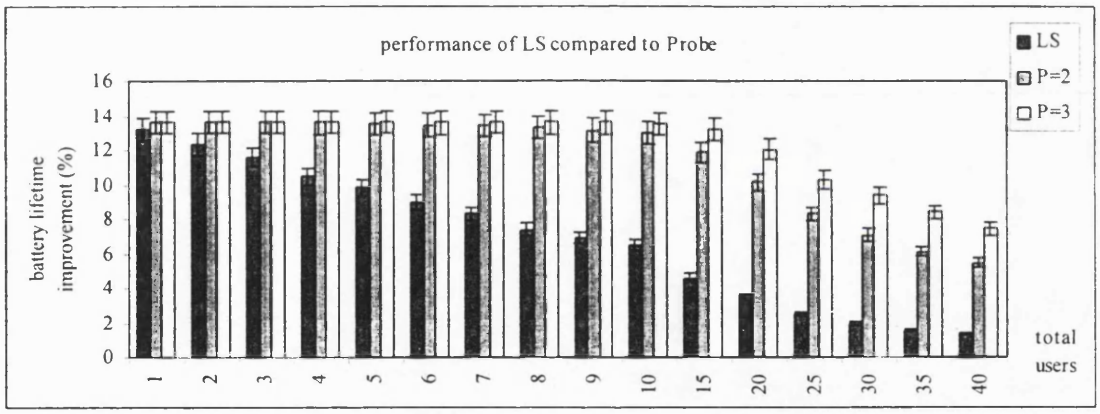


Figure 6-11

Graph comparing percentage of rejected requests of Probes to LS

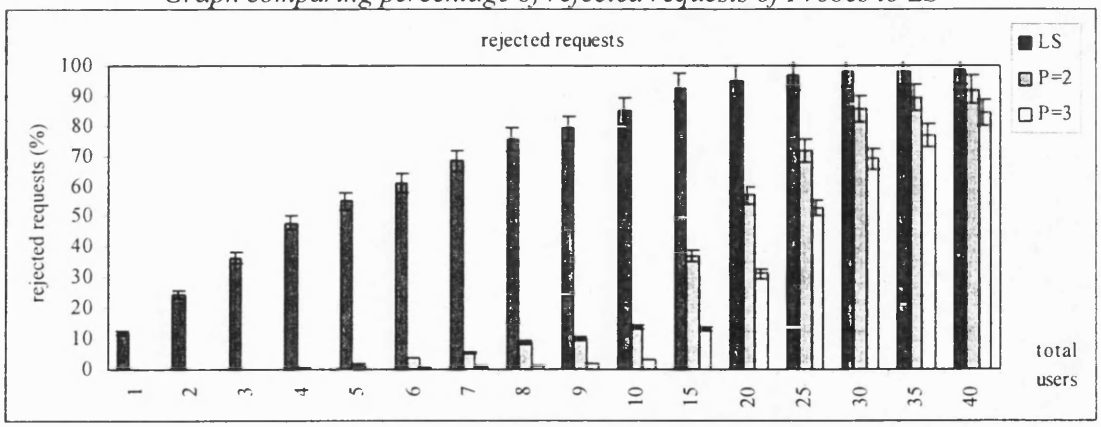


Figure 6-12 shows that the percentage of rejected requests with Slotted Probe was very low. Although Figure 6-13 shows that the battery lifetime improvement of Slotted Probe was lower than *P=2*, Figure 6-14 shows that its percentage of jobs transferred was higher

than *P=2*. Figure 6-15 shows that when the algorithm operates in Slotted Probe mode, the response time improvement was much lower than *P=2*, but slightly higher than LS.

Figure 6-12

Graph comparing the percentage of rejected requests of LS, P=2 and Slotted Probe.

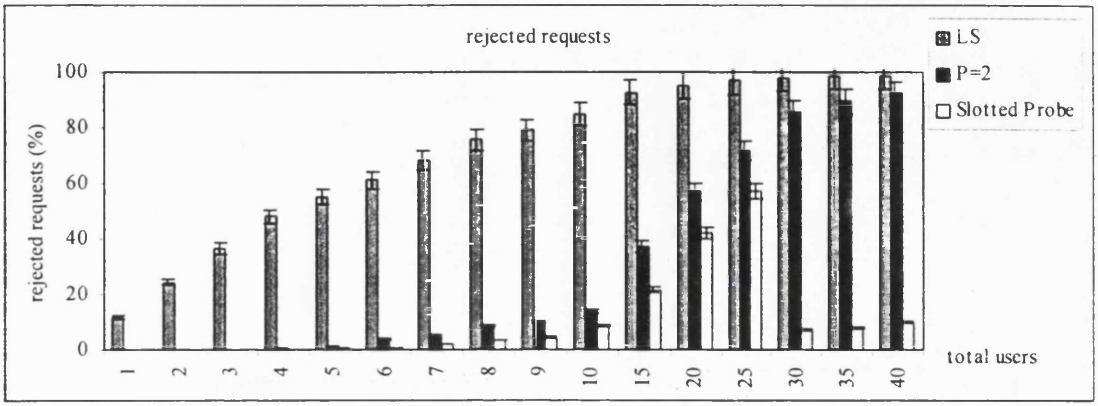


Figure 6-13

Graph comparing battery lifetime improvement of LS, P=2 and Slotted Probe.

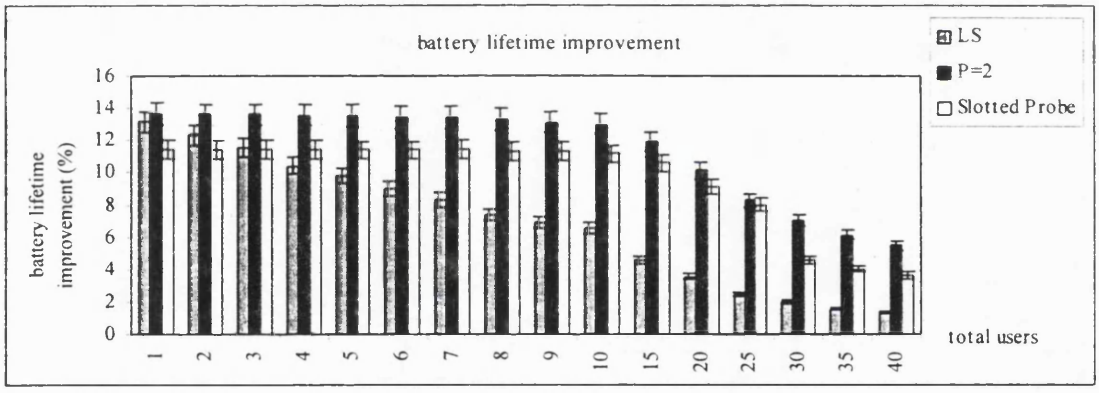


Figure 6-14

Graph comparing the percentage jobs transferred of LS, P=2 and Slotted Probe.

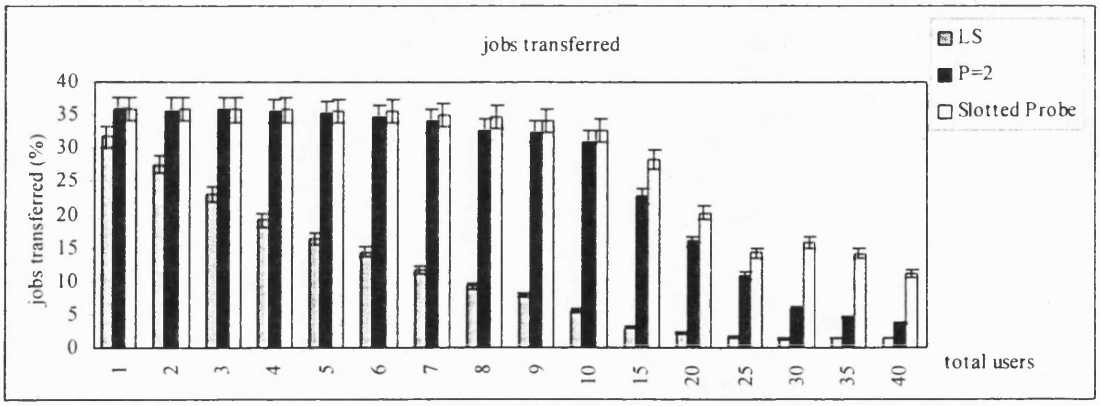
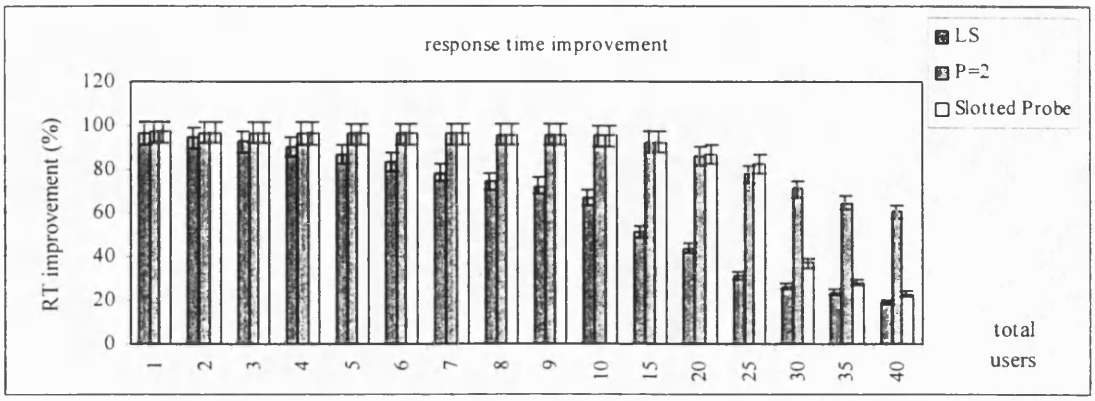


Figure 6-15

Graph comparing response time improvement of Slotted Probe to LS and P=2



Analysis of Results:

Two-way ANOVA with replication was performed to determine if there is a significant difference in performance when transfer requests were delegated to other fixed hosts. Table 6-6 shows that there is a significant difference in performance when transfer requests were delegated to 2 other hosts. Table 6-7 shows that there is a significant difference between *P=2* and *P=3*, indicating that performance improved as more fixed hosts offered their services to mobile hosts.

Table 6-6

H_0 : mean battery life as total users increases are equal;
H_0 : mean battery lifetime of LS and P=2 are equal.

two-way ANOVA with replication; α=0.05						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critical*
Total users	1109581	15	73972.05	641.87	0	1.67
Algorithms	523613	1	523613.10	4543.50	0	3.85
Interaction	158526	15	10568.46	91.70	0	1.67
Within	143825	1248	115.24			
Total	1935546	1279				

Table 6-7

H_0 : mean battery life as total users increases are equal;
H_0 : mean battery lifetime of P=2 and P=3 are equal.

two-way ANOVA with replication; α=0.05						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critical*
Total Users	1354082	15	90272.14	1182.06	0	1.67
Algorithms	25231	1	25231.45	330.39	0	3.85
Interaction	21952	15	1463.49	19.16	0	1.67
Within	95307	1248	76.37			
Total	1496574	1279				

A possible explanation for the reason Slotted Probe did not improve battery lifetime more than $P=2$ even though it resulted in more job transfers is as follows. As is the case with Slotted LS, Slotted Probe successfully reduced the percentage of rejected requests by limiting the number of users contending for the fixed hosts' spare capacity and distributing them more fairly among the mobile hosts, thus more jobs were transferred. However, in limiting the number of users in each slot, there is a possibility that Slotted Probe did not fully utilise the spare capacity at the fixed hosts. We expect that the performance of Slotted Probe can be optimised by tuning the parameter value N.

6.4 Conclusion

Load sharing could cause worse power utilisation than no load sharing as the number of users competing for spare capacity increased. If load sharing is to be implemented in a real environment, it is imperative that appropriate measures were taken to prevent instability. Any algorithm should be equipped with a mechanism to detect when load sharing is no longer feasible and to react accordingly. This feature is important not only to prevent instability, but also so that the algorithm is scalable.

Two modified algorithms were introduced, i.e. Backoff and Slotted LS. The Slotted LS algorithm was found to be effective not only in avoiding instability, but also in allocating spare capacity more efficiently among users. As a result, Slotted LS performed better than LS and Backoff algorithms in conserving power.

The results suggest that in order for load sharing to scale to cater for a high number of users, spare capacity from a few fixed hosts must be utilised. Otherwise, even with Slotted LS performance degrades rapidly, and load sharing will eventually cease to be effective. The number of fixed hosts offering their service is determined by the workload of the fixed hosts; if fixed hosts are busy, they will be unable to offer their services to mobile hosts. Consequently, like the finding in previous studies on load sharing, under high system load, the best course of action is to disable load sharing.

7. Emulation of Load Sharing on a Wireless LAN

The simulation results indicate that load sharing is a potentially good power management strategy. In this chapter, the emulation carried out to verify the simulation results and to ascertain the effectiveness of the strategy in a real environment is discussed.

Motivations:

There are two motivations for emulating the load sharing algorithm. Firstly, simulation results need to be verified to ascertain the extent to which the simulation represents a real environment. Secondly, there is a discrepancy between our and Rudenko et al's results, which we would like to investigate.

Methodology:

The mobile host was a Mobile 3000 Scenic laptop from Siemens Nixdorf, employing an Intel Pentium 133 MHz CPU with 40 Mb RAM and running Windows NT 4.0. The CPU consumes 4.3 Watts when active and 1.7 Watts in idle mode. The transceiver was a Lucent Wavelan 2 GHz system consuming 3.00 Watts when active and 1.48 Watts when operating in doze mode. An Ultra Sparc 1 workstation running Solaris with a 143 MHz processor, 160 Mb of RAM and a 9 Gb hard disk was used as a fixed host.

Code for the emulation was written in C using Microsoft Visual C++, where a connection between the laptop and the workstation was established using a socket() call and messages were exchanged using the sendto() and recvfrom() functions. Use of the CPU executing a job was emulated with a while loop performing a simple mathematical computation.

A simulation was also run using the same traces and environmental parameters as the live environment so that the results could be compared to the emulation results. Both emulation and simulation were run for a set of 30 users, with no load sharing and with load sharing. Each emulation was run until the battery was flat, and the battery was fully recharged. The optimal LS algorithm was used in order to obtain an upper bound on the amount of possible saving which may be achieved. As it has been established that it is mobile hosts with high CPU utilisation that benefit most from load sharing, both the emulation and simulation were run assuming *param.proc-speed=1/5*. The simulation was run with *param.bw* set to 2 Mbps, the same as the amount of bandwidth available on the wireless LAN. As the laptop implements power saving mechanisms, the simulation was run combining load sharing with a disk spin-down strategy.

The results obtained, which are presented below, have also been discussed in [Othm99].

7.1 Results of Emulation and Simulation

Figure 7-1 compares the battery lifetime improvement of the emulation and simulation, while Table 7-2 summarises the results obtained. On average, the emulation result shows that battery lifetime was extended by about 21%, while the simulation result shows an average improvement of about 25%. Even though in a few cases there was more than 10% difference in the improvement achieved, generally the difference was small.

There was little difference in the percentage of jobs transferred, as depicted by Figure 7-2. Both emulation and simulation results shows significant improvement in response time, but the response time improvement of the emulation was found to be more reasonable than that of the simulation (see Figure 7-3), where the average response time improvements were 65% and 96% for the emulation and simulation, respectively. As discussed in Chapter 5, the high response time improvement for the simulation might have to do with the simplification of the *Channel* entity, where a reliable communication medium was assumed, and possible interference and packet loss were not simulated. When running the emulation experiments, it was observed that the quality of the communication channel sometimes varied over time, during which the communication delays were higher. Since these conditions were not simulated, the simulation results exhibit a much higher response time. Table 7-1 shows the percentage of packet loss and delays when the Unix ping command was executed between a fixed workstation and the laptop during one of the experiments.

Table 7-1

Table showing the percentage of packet loss and delays during one of the emulation experiments.

```
----sisley.cs.ucl.ac.uk PING Statistics----
48 packets transmitted, 30 packets received, 37% packet loss
round-trip min/avg/max = 8975.603/9375.171/10645.447 ms
```

Figure 7-1

Graph comparing the results of emulation and simulation.

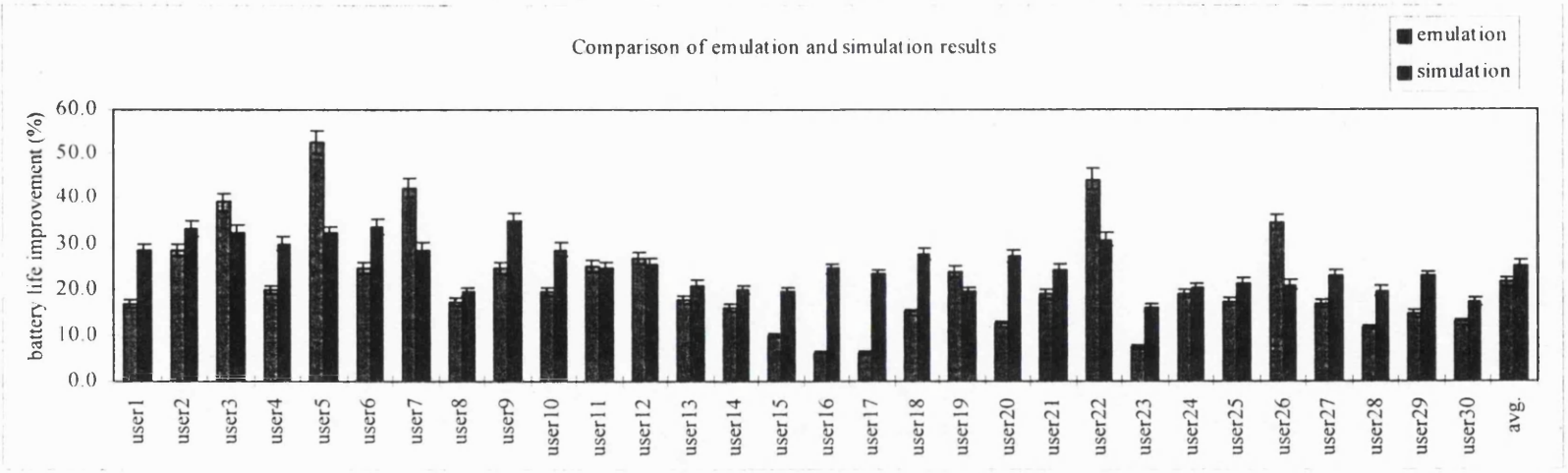


Table 7-2

Table comparing the results of emulation and simulation.

	maximum		minimum		average		std. deviation	
	emulation	simulation	emulation	simulation	emulation	simulation	emulation	simulation
battery lifetime NLS (hour)	3.39	3.00	2.34	2.57	3.00	2.79	0.26	0.13
battery lifetime LS (hour)	3.97	3.60	3.07	3.4	3.62	3.48	0.18	0.04
battery lifetime improvement (hour)	1.24	0.90	0.19	0.48	0.62	0.69	0.27	0.12
battery lifetime improvement (%)	52.81	34.84	6.12	16.11	21.39	24.97	11.45	5.35
response time improvement (%)	81.06	97.75	33.16	56.34	64.71	96.41	9.58	7.75
jobs transferred (%)	76.67	75.53	60.04	51.29	66.49	62.95	3.92	6.19

average difference in battery lifetime improvement =8 %; standard deviation=5

Figure 7-2

Graph comparing the amount of jobs transferred for the emulation and simulation.

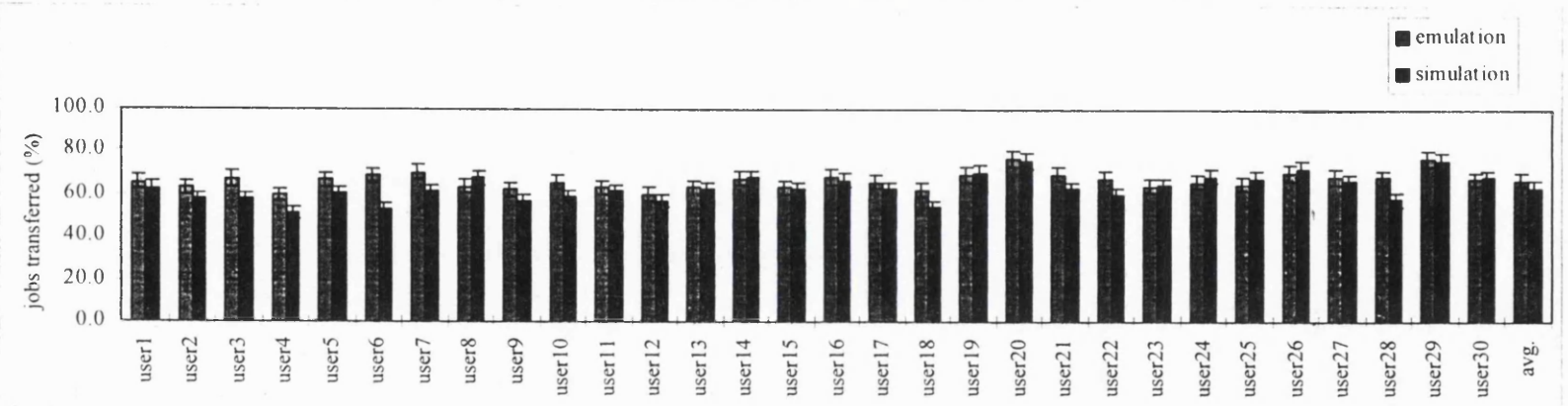


Figure 7-3

Graph comparing response time improvement of emulation and simulation.

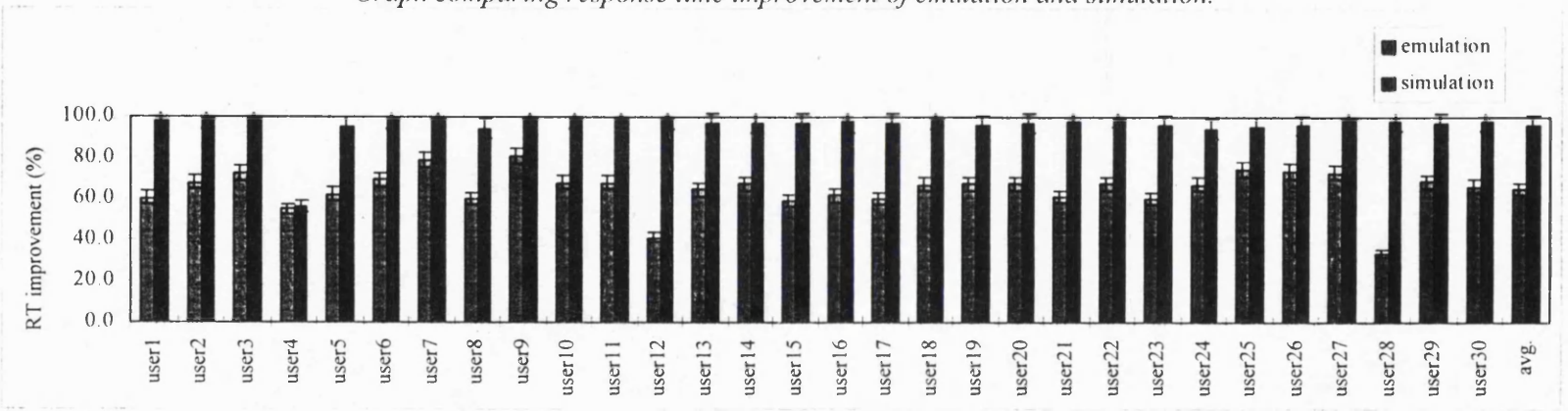


Figure 7-4 and Figure 7-5 show that the relationship between CPU utilisation and battery lifetime improvement of the emulation and simulation, respectively. Both graphs show that there is a linear relationship between CPU utilisation and battery lifetime improvement, but that relationship is more evident in the emulation result. The correlation coefficient is 0.78 and 0.72 for the emulation and simulation, respectively. This result verifies the result discussed in Chapter 5 when the influence of CPU utilisation on the benefit of remote execution was examined.

Figure 7-4

Graph showing the relationship between CPU utilisation and battery lifetime improvement (emulation).

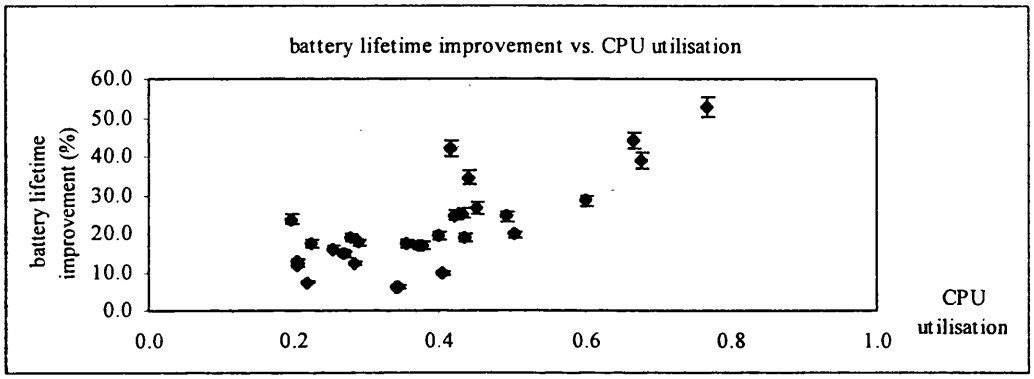
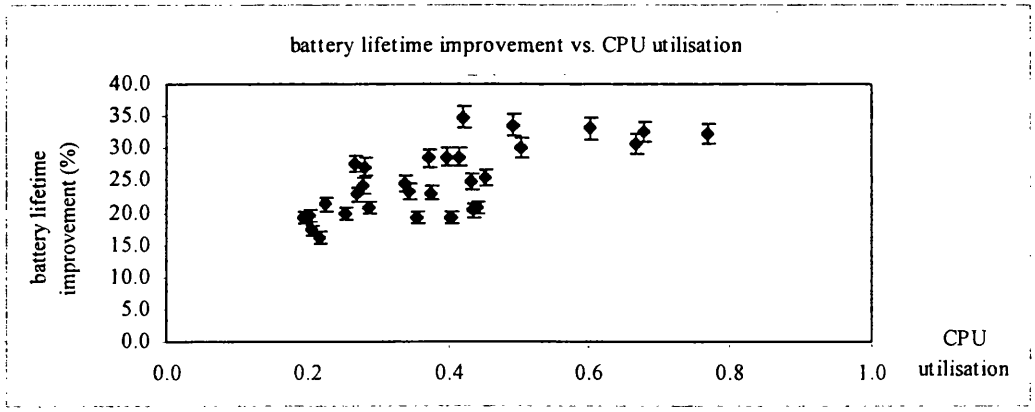


Figure 7-5

Graph showing the relationship between CPU utilisation and battery lifetime improvement (simulation).



Analysis of Results:

z-tests were carried out to determine:

- if there is a significant difference in battery lifetime when load sharing is performed, for both simulation and emulation,
- if there is a significant difference in the emulation and simulation results.

Both the emulation and simulation results exhibit a significant improvement from no load sharing, as shown in Table 7-3 and Table 7-4, respectively. An analysis of the improvement achieved by the emulation and simulation shows that the difference is not significant (see Table 7-5), implying that the simulation does indeed model a real environment to a good degree of accuracy, given the particular parameters chosen.

Table 7-3

Emulation : H_0: mean battery lifetime of NLS and LS are equal.

z-test; α=0.05	*NLS*	*LS*
Mean	3.00	3.62
Known Variance	0.07	0.03
Observations	30	30
Hypothesized Mean Difference	0	
z	-10.72	
P(Z<=z) one-tail	0	
z Critical one-tail	1.64	
P(Z<=z) two-tail	0	
z Critical two-tail	1.96	

Table 7-4

Simulation : H_0: mean battery lifetime of NLS and LS are equal.

z-test; α=0.05	*NLS*	*LS*
Mean	2.79	3.48
Known Variance	0.02	0.001
Observations	30	30
Hypothesized Mean Difference	0	
z	-27.79	
P(Z<=z) one-tail	0	
z Critical one-tail	1.64	
P(Z<=z) two-tail	0	
z Critical two-tail	1.96	

Table 7-5

H_0: mean emulation improvement and mean simulation improvement are equal.

z-test; α=0.05	*emulation*	*simulation*
Mean	21.39	24.97
Known Variance	131.13	28.60
Observations	30	30
Hypothesized Mean Difference	0	
z	-1.55	
P(Z<=z) one-tail	0.06	
z Critical one-tail	1.64	
P(Z<=z) two-tail	0.03	
z Critical two-tail	1.96	

The results in Chapter 5 showed that a 2x improvement in available bandwidth did not bring a significantly higher increase in the percentage of jobs transferred. The average percentage of jobs transferred was approximately 30% at available bandwidth of 100 kbps. The emulation results show that the average percentage of jobs transferred was 77%, indicating when high bandwidth is available, more jobs were transferred. This, however, did not lead to a higher battery lifetime improvement, due to a high overhead cost of transferring large jobs.

With regards to job transfer, the questions which must be answered are: what type of jobs will be transferred, and how big are the jobs? Considering that applications are becoming increasingly large, job size might become a factor which impedes load sharing. However, it is reasonable to assume that if a job is to be transferred, the applications required to execute the job should be available on the fixed host. For example, in order to transfer a code compilation, the fixed host accepting the transfer request should have the compiler required to compile the code. It is not necessary to transfer the compiler to the fixed host, only the code file needs to be transferred. If only data files have to be transferred, the job size might not be very large, hence incurring lower overhead cost (in Chapter 4, when discussing the assumptions for this study, the job size is the size of the executable files because data file information is not available).

The use of a distributed file system which provides support for mobile users, such as Coda[1], might help reduce the overhead cost of transferring jobs. Its file replication facility makes it possible for a user to send only the changes made to a file since the previous transfer, instead of transferring a complete document, thus reducing the overhead cost of job transfers.

7.2 Conclusion

The results obtained exhibit little difference between the simulation and emulation. Further analysis shows that there is no significant difference between the two, implying that the simulation environment built for the study closely reflects the real environment to a good degree given the particular parameters chosen. In most cases, simulation results are close to the results obtained for the emulation. Given this result, it is not reasonable to argue that simulation using different parameters are also likely to represent a real environment, since there was nothing special about the way that parameters were chosen in this case. The results not only verify that load sharing is effective in extending battery lifetime, but also confirm the influence of CPU utilisation on the benefit of remote execution.

[1] Information about Coda can be obtained from http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

8. Future Vision and Future Work

In this chapter, the future scenario for wireless and mobile computing technologies and the applicability of this work are discussed. This chapter ends with a proposal for future work.

8.1 Future Systems

Mobile devices are no longer used solely for the purpose of communications, but now are also used by roaming users to allow greater flexibility over working practices, effectively extending the office into the wider environment. In the next decade, it has been forecast that 20% to 30% of GSM revenue might come from mobile data. If the rapidity of increase in the amount of data carried by terrestrial wired telecom networks in an indicator, this may well be a conservative estimate.

By the nature of the way that the radio medium is licensed, the future of mobile data depends largely on the development of appropriate standards. Currently, the bandwidth available over GSM for mobile data is low (typically in the range of 9.6 kbps to 14.4 kbps). However, there are a number of initiatives to develop high speed, wide area, data services. Thus trials of High Speed Circuit Switched Data (HSCSD) are currently underway, and it is projected that GPRS will roll out in the near future. These services will offer data rates up to about 130 kbps. Enhanced Data for GSM Evolution (EDGE), which is a successor of GPRS, will provide data rates of up to 348 kbps.

In a wider context, Iridium is planning a second generation satellite system offering a bandwidth of up to 120 kbps, which it plans to launch in 2003. It is also targeting 64 kbps / 125 kbps throughput for handheld devices. Ericsson and Telia are testing a wireless interconnection to IP-based enterprise networks, effectively providing enterprise LANs for fixed and mobile terminals. The commitment shown by the industry to provide support for mobile data is evident from the Wireless Application Protocol (WAP) Forum, which is joined by 60 companies, with a goal of designing and establishing standards for mobile data.

Perhaps the most significant development for the future will centre around ETSI's work on the Universal Mobile Telecommunications Standard (UMTS) or the ITU's work on IMT-2000. UMTS is a third generation mobile (3G) systems. Its goal is to cater for future market demands for low cost, high quality mobile personal communications. UMTS will deliver pictures, graphics, video communications and other wide-band information as well as voice and data to mobile users. It extends the existing capabilities of mobile, cordless and satellite technologies to provide increased capacity, data capability and a wider range of services, blurring the distinction between the use of telephones for voice communications and the use

of PCs for data transmissions. Since UMTS provides a wide range of services, it also provides support for a wide variety of terminals, from PDAs to palmtops to laptops.

UMTS offers a wide range of quality of service (QoS) by optimising the communications channel for each individual service. By tailoring the communications channel QoS (bandwidth, error rate and latency) to a range of services, traffic can be packed into the channel more efficiently, potentially reducing costs to service providers, and price for users. Although UMTS offers connectionless services, users will get the impression of being connected for the whole duration of a session. In reality, communication channels and other resources will only be occupied while data is being sent. As soon as the transmission is completed, the resources are automatically released to another user. 3G systems will provide adaptive services where if a requested QoS cannot be met, the service is adapted to provide a similar, but lower quality version, taking into account the dynamic nature of radio channel and traffic characteristics. A mechanism will be introduced to price services according to the QoS provided.

Initially, UMTS will provide transmission rates of up to 2 Mbps. Later on, UMTS will be integrated with picocellular systems based on Wireless LAN (W-LAN) and Broadband Radio Access Networks (BRANs) to provide a bandwidth of up to 155 Mbps. The UMTS networks will be comprised of terrestrial networks and Satellite UMTS (S-UMTS) networks. S-UMTS is required to serve rural and remote areas, which is essential to achieve the goal of providing a ubiquitous and universal services. In addition to multimedia services, UMTS will also provide interactive services and the Virtual Home Environment (VHE) services.

Bluetooth[1] is another project which will make ubiquitous computing a reality. Bluetooth aims to replace proprietary cables connecting devices with one universal short-range radio link to facilitate access to both LANs and WANs to provide a universal bridge to existing data networks. It will also provide a mechanism to form small private ad hoc groupings of connected devices away from network infrastructures. Any digital device can be part of the Bluetooth system: laptops, handheld computers and digital cellular phone can be used together seamlessly. For example, emails can be received on the laptop via the digital cellular phone the user is carrying, and a presentation can be downloaded from a user's laptop to a data projector without connecting cables.

The proliferation of wireless technology applications will not be limited to supporting roaming users. In the future, wireless and mobile computing technologies will also be widely deployed at home. For example, Shared Wireless Access Protocol (SWAP) aims to provide a new, common air interface which supports wireless voice and LAN data services in the home

[1] Bluetooth web page is at http://www.bluetooth.com/.

environment. The specification ensures interoperability among PCs, communication and consumer electronics devices equipped with wireless communication capabilities, and is aimed for the home market. Among the applications which will be made possible by SWAP are:

- Information delivered via the Internet to anywhere in the home, e.g. a mobile display pad connected to the Internet can access recipe information in the kitchen, or it can be taken into the garage for the latest automobile mechanical updates.
- Control of electrical systems and appliances, e.g. to turn on/off lighting in the house while users are away to give the impression that someone is at home.
- Effective utilisation of communication channels by dynamically allocating phone line for incoming and outgoing calls, fax and Internet access.

These are only a few examples of possible applications, more information can be obtained from http://www.homerf.org.

In conclusion, the ongoing work in this field suggests that a ubiquitous computing environment where users truly have anytime-anywhere-access will one day be achieved, and the services will be provided at affordable prices. The use of wireless technologies will become widespread, encompassing every aspect of people's life, and giving users access to information whenever it is required. Access to the network will no longer be limited to downloading information or exchanges of emails and short messages, but will cover all aspects of applications currently available only to users on fixed networks, as well as a range of applications yet to be developed for the consumer electronics market.

In summary, we are currently seeing the conception of systems in which computers and networks will be transparently integrated into users' everyday lives. The goal of ubiquity will be reached and the demand for the services which run over such systems will be driven by the usefulness of the services rather than lower level technical specifications.

8.2 Applicability of this Work

The previous section and Chapter 2 show that there will be a wide range of mobile/wireless applications available to users. The question we would like to answer now is how our work will be applicable in such a diverse environment. Obviously, not all mobile applications will benefit from load sharing, but we believe that it will be particularly useful for applications where mobile devices are used as an extension of the workplace. One particular driver for the current expansion of the wired network into the home is precisely this, though secondary development involves the wider use of systems in leisure activity. The fact that there is an increasing demand for mobile data services indicates that users expect to be able to access

and manipulate data on the fixed network while roaming. In the near future, this data is likely to be work associated, though new applications will be developed as the number of mobile devices increase, and the type diversifies to include consumer electronic devices.

In the case of work-based use, it is imperative that the necessary support is available so that users are able to work whenever, and wherever, they wish to do so. In the case of wider applications, this is doubly important, since these will be predicated on embedding computer and network systems in such a way that the user should not explicitly be aware of their existence. In both scenarios, intelligent dynamic reconfiguration will be essential in providing the type of services that are required over extended periods in highly heterogeneous environments that are changing. One important part of this is the ability to extend battery lifetime, since this is currently (and will be for the foreseeable future) a limiting factor to the full exploitation of the potential of such environments. This thesis has demonstrated that load sharing affords an important weapon in achieving this goal for certain sorts of tasks. Indeed, since the cost of running the load sharing algorithm is minimal, it will be sensible to run it just in case jobs are suitable for transfer. If a job is not CPU-intensive and not suitable for transfer, the adaptive algorithm will identify it as such.

Undoubtedly, there are limitations to the scope of the work described in this thesis. We have argued that environments may become massively more heterogeneous than is currently the case. In this case, load sharing may be impracticable, simply because of the range of different types and capabilities of system. However, if people see significant benefit can be brought by load sharing, and there is a demand for this service to be provided, the industry will eventually have to cater for this demand. One way of addressing this issue is by providing a generic platform where jobs can be ported between different machines to avoid heterogeneity from becoming a hindrance to load sharing. This may be required in any case in order to allow for the full range of dynamic behaviours to be exploited and for cross platform applications to be developed.

8.3 Future Work

In this study, it was assumed that mobile hosts compete amongst themselves for a fixed host's processor cycles without considering the monetary cost that may be involved. If users are being charged for using the wireless link, or for utilising the fixed host's resources, this will be another constraint which must be considered when implementing load sharing as a power management strategy. An additional question to address is how much users are willing to pay in order to execute jobs remotely, and if the cost is justified.

This problem may be undertaken using the ***microeconomic approach***, previously explored by Ferguson et al [Ferg88] and Waldspurger et al [Wald92] as another approach to distributing load in a distributed network. In the approach proposed by Ferguson et al, the computer system was modelled as resources, consumers and suppliers. ***Resources*** are processors, memory and communication bandwidth, while the ***consumers*** are jobs that have to be executed and ***suppliers*** are the processors and their operating systems. Processors and jobs are modelled as independent economic agents that selfishly attempt to optimise their satisfaction without attempting to improve or optimise system performance. If this approach is applied in the context of this study, transfer requests will no longer be served on a fist-come-fist-served basis, but based on who is willing to pay for the resources. This not only enforces competition among users, but also restricts access to those who can afford to pay for it.

When the stability of the load sharing algorithm is discussed in Chapter 6, we showed that as users competing for the MSS's processor cycle increases, load sharing ceases to be effective. In order for it to continue to be feasible, spare capacity at other fixed hosts has to be utilised, where the MSS probes other fixed hosts to delegate transfer requests. Another approach to this problem is by using an intelligent mobile agent. Starner et al [Star97] describes an ***intelligent mobile agent*** as a butler that anticipates the requirements of a user and tries to find the required resources to execute a job. In the context of load sharing, instead of having a mobile host sends transfer requests the MSS, its intelligent agent may forecast what the user requirement in the near future might be and hence reserve the required resources. It may facilitate remote execution by reserving communication bandwidth or processor cycle on fixed hosts. The agent may even use the microeconomic approach in deciding the best way to utilise available resources according to a user's budget.

In the longer term, the exploration of new applications for wireless systems of the type we have envisaged above will lead to the identification of different constraints. Although it is hard to predict what these might be at present, we can certainly foresee a very interesting future for research in the area of intelligent dynamic adaptation in mobile systems.

9. Conclusion

Mobile devices operate on battery power and, given the fact that there is unlikely to be much improvement in battery capacity, it is vital that power utilisation be managed efficiently and economically. This study explores the possibility of using the concept of load sharing to utilise resources on the fixed network in order to conserve battery power on mobile hosts. It is built on the premise of making use of available resources on the fixed network to execute jobs on behalf of mobile hosts. The load sharing algorithm selects suitable jobs for remote execution on a fixed host in such a way that the power consumed for transferring the job is less than the power consumed if the job is executed locally. Transferring jobs reduces power consumption as the CPU is able to operate in doze mode more often.

The extent to which previous work on load sharing is relevant to our work was explained in Chapter 3, and the differences in the assumptions and mechanism of load sharing in wireless networks were presented in Chapter 4. These differences are inevitable due to the nature of the mobile computing environment; consequently, factors which are non-issues in fixed networks must now be taken into consideration. As the primary concern is to examine the effectiveness of load sharing in conserving power, the design of the algorithm is centred around trying to utilise power economically. Modifications to the existing load sharing strategy took into considerations the mobile computing environment and the objective of conserving battery power.

In Chapter 1, the hypotheses for this study were listed. It was expected that low bandwidth would impede load sharing. This hypothesis was tested by running simulations where the available bandwidth was varied to determine its effect on battery lifetime improvement. Although the results show that low bandwidth does impede load sharing, there is an exception. Providing that job size is small, it is possible to transfer a substantial number of jobs, resulting in a significant battery lifetime improvement. This indicates that load sharing might still be effective even at low bandwidth.

Load sharing is expected to improve performance by giving mobile hosts access to faster machines on the fixed network. Results obtained from simulations which varied the processor power of the mobile hosts show that this is indeed the case. It was found that if the mobile hosts have the same processing power as fixed hosts, it is unlikely that any jobs would be transferred, as communication delays cause increased response time. It is mobile hosts with low processing power which benefit most from load sharing, not only in terms of battery lifetime, but also in terms of better response time.

A finding of this study, which is similar to previous studies, is that it is CPU-intensive jobs that benefit from load sharing. It was found that the transfer of CPU-intensive jobs extends the period the CPU operates in doze mode significantly, resulting in reduced power consumption. In Chapter 1, we stated that we believe the proposed strategy to be particularly important when users are executing CPU-intensive jobs, in which case it is crucial that the battery power is not suddenly depleted. This finding shows that load sharing will indeed have a role to play in conserving power if users are using applications which are CPU-intensive.

In addition to establishing that load sharing does bring significant battery lifetime improvement and identifying factors influencing the benefits gained from load sharing, the possibility of combining load sharing with another power management strategy was also examined. Simulation combining load sharing with a disk spin-down strategy indicates that further power saving is possible when the strategies are combined.

Two adaptive load sharing algorithms, History and ALS, were proposed to offer a more practical approach to implementing load sharing in a real environment. Unlike the optimal algorithms, the adaptive algorithms assume no a priori knowledge of CPU requirements of jobs, and perform almost as well as the optimal algorithm, indicating that a performance which is near the optimal performance might be possible.

When addressing the stability and scalability issues, it was found that as the number of users competing for spare capacity on the fixed network increases, more than one fixed host is required to cater for the demand. If only one fixed host caters for the mobile hosts, load sharing does not scale well, and performance degrades rapidly as the number of users increases. The number of fixed hosts available to offer their services to mobile hosts depends on the current workload on the fixed network. If the fixed hosts are currently under heavy load, it is best to disable load sharing. An appropriate measure was introduced to prevent performance degradation under heavy load, and the Slotted LS algorithm appears to have succeeded in doing so. In addition to preventing performance degradation, Slotted LS also allocates spare capacity more fairly among mobile hosts.

There are other factors which may cause instability of the load sharing algorithm. As discussed in Chapter 4, to simplify the simulation, we made an assumption that a reliable communication link exists between the mobile hosts and the MSS, and did not simulate possible interference or packet loss. In reality, high interference and packet loss/error may become factors which leads to instability if the load sharing algorithm attempts to transfer jobs under such unfavourable circumstances, possibly wasting battery power due to retransmissions. Hence, the algorithm needs to be intelligent enough to detect these

conditions and react accordingly. We did not attempt to address this issue as we believe an approach such as the one proposed by [Ruln96], discussed in Chapter 2, may be adopted.

The findings of this study also show that the proposed load sharing algorithm has been successful in making use of otherwise unutilised resource on the fixed network. While previous studies proposed this adaptive ability to utilise resources/services merely for the purpose of executing applications on behalf on mobile hosts, we exploit them to conserve battery power. In the context of the load sharing algorithm, this requirement is important not only to make the best use of available resources, but also to avoid instability. When load sharing is no longer feasible, appropriate measures are required to avoid load sharing from causing worse power utilisation than when there is no load sharing.

In conclusion, the proposed load sharing algorithm was shown to be a potentially effective power management strategy. The contributions and achievements of this study are summarised below:

1. It successfully adapted a well-known concept in distributed system to address a problem in mobile computing.
2. It identified the necessary modifications required to implement load sharing as a power management strategy in a mobile computing environment.
3. It identified factors conducive to load sharing and factors which impede load sharing.
4. It identified the type of applications which are likely to benefit from load sharing.
5. This study shows that it is possible to make use of otherwise unutilised resources on fixed hosts to conserve battery power on mobile computers.

References

[Badr93] B.R. Badrinath, A. Acharya, T. Imielinski, *Impact of Mobility on Distributed Computing*, Operating Systems Review, Vol. 27, No. 2, April 1993.

[Badr95] B.R. Badrinath, G. Welling, *Event Delivery Abstraction for Mobile Computing*, Technical Report, LCSR-TR-242, Rutgers University, 1995.

[Badr96] B.R. Badrinath, A. Acharya, T. Imielinski, *Designing Distributed Algorithms for Mobile Computing Networks,* Computers and Communications, Vol. 19, April 1996.

[Bagr95] R. Bagrodia. W.W. Chu, L. Kleinrock. G. Popek, *Vision, Issues, and Architecture for Nomadic Computing*, IEEE Personal Communications, December 1995.

[Chev94] K. Cheverst, G.S. Blair, N. Davies A. Friday, A.D. Cross, P.F. Raven, *Moving 'Desktop' into the Field*, IEE Colloquium on Integrating Telecommunications and IT on the Desktop, London, March 1994.

[Cox95] D.C. Cox, *Wireless Personal Communications : What Is It ?*, IEEE Personal Communications, April 1995.

[Davi93] N. Davies, G.S. Blair, A. Friday, A.D. Cross, P.F Raven, *Mobile Open Systems Technologies for the Utilities Industries*, IEE Colloquium on CSCW Issues for Mobile and Remote Workers, London, March 1993.

[Davi94a] N. Davies, S. Pink, G.S. Blair, *Services to Support Distributed Applications in a Mobile Environment*, Proc. of the 1st International Workshop on Services in Distributed and Networked Environments, Prague, June 1994.

[Davi94b] N. Davies, G.S. Blair, K. Cheverst, A. Friday, *Supporting Adaptive Services in a Heterogeneous Mobile Environment*, Proc. of IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December, 1994.

[Deme94] A. Demers, K. Peterson, M. Spreitzer, D. Terry, M. Theimer, B. Welch, *The Bayou Architecture : Support for Data Sharing Among Mobile Users*, Proc. of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December, 1994.

[Doug94a] F. Douglis, P. Krishnan, B. Marsh, *Thwarting the Power-Hungry Disk*, Proc. of the 1994 USENIX Winter Conference, San Francisco, January 1994.

[Doug94b] F. Douglis, R. Caceres, B. Marsh, *Storage Alternatives for Mobile Computers*, Matsushita Information Technology Laboratory Technical Report MITL-TR-93-93, March 21, 1994.

[Doug95] F. Douglis, B. Bershad, *Adaptive Disk Spin-down Policies for Mobile Computers*, Proc. of the 2nd USENIX Symposium on Mobile and Location-Independent Computing, Michigan, USA, April 1995.

[Duch91] D. Durhamp, S.K. Feiner, G.Q. Maguire, *Software Technology for Wireless Mobile Computing*, IEEE Network Magazine, November 1991.

[Duch92] D. Duchamp, *Issues in Wireless Mobile Computing*, Proc. Of the 3rd Workshop on Workstation Operating Systems, IEEE, April 1992.

[Eage86a] D.L Eager, E.D. Lazowska, J. Zahorjan, *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 5, May 1986

[Eage86b] D.L. Eager, E.D. Lazowska, J. Zahorjan, *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing*, Performance Evaluation, Vol. 6, 1986.

[Ebli94] M.R. Ebling, L.B. Mummert, D.C. Steere, *Overcoming the Network Bottleneck in Mobile Computing*, Proc. of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December, 1994.

[Ezza86] A.K. Ezzart, R.D. Bergeron, J.L. Pokoski, *Task Allocation Heuristics for Distributed Computing Systems*, Proc. of the 6th International Conference on Distributed Computing Systems, 1986.

[Ferg88] D. Ferguson, Y.Yemeni, C. Nikolaou, *Microeconomics Algorithms for Load Balancing in Distributed Computer Systems*, Proc. of the 8th International Conference on Distributed Computing Systems, 1988.

[Form94] G.H. Forman, J. Zahorjan, *The Challenges of Mobile Computing*, Technical Report, UW CSE #93-11-03, University of Washington, March 1994.

[Frid96] A.J. Friday, G.S. Blair, K.W.J Cheverst, N. Davies, *Extensions to ANSAware for Advanced Mobile Applications*, Proc. of the International Conference on Distributed Platform, Dresden, Germany, February 1996.

[Gran93] W.C. Grant, *Wireless Coyote : A Computer-Supported Field Trip*, Communications of the ACM, Vol. 36, No. 5, May 1993.

[Gree94] P.M. Greenwalt, Modeling *Power Management for Hard Disks*, Proc. On Modelling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE, January 1994.

[Hac89] A. Hac, *A Distributed Algorithm for Performance Improvement Through File Replication, File Migration, and Process Migration*, IEEE Transactions on Software Engineering, Vol. 15, No. 11, November 1989.

B.

[Helm96] D.P. Helmbold, D.D.E. Long, B. Sherrod, *A Dynamic Disk Spin-Down Technique for Mobile Computing*, Proc. of Mobicom '96, 1996.

[Hous96] B.C. Housel, D.B. Lindquist, *WebExpress : A System for Optimising Web Browsing in a Wireless Environment*, Proc. of Mobicom '96, 1996.

[Hsu86] C.H. Hsu, J.W. Liu, *Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems*, Proc. of the 6th International Conference on Distributed Computing Systems, 1986.

[Imie94a] T. Imielinski, S. Viswanathan, B.R. Badrinath, *Energy Efficient Indexing on Air*, Proc. of the International Conference on Management of Data, ACM SIGMOD, May 1994.

[Imie94b] T. Imielinski, B.R. Badrinath, *Mobile Wireless Computing : Solutions and Challenges in Data management*, Communications of the ACM, Vol. 37, No. 10,

October 1994.

[Imie95] T. Imielinski, M. Gupta, S. Peyyeti, *Energy Efficient Data Filtering and Communication in Mobile Wireless Computing*, Proc. of the 2nd USENIX Symposium on Mobile and Location-Independent Computing, Michigan USA, April 1995.

[Imie96] T. Imielinski (ed.), H.F. Korth, *Mobile Computing*, Kluwer Academic Publishers, 1996.

[Inte94] Intel Application Note, *Pentium® Processor (610\75) Power Consumption*, Rev. 1.1, October 1994.

[Jacq93] C. Jacqmot, E. Milgrom, *A Systematic Approach to Load Distribution Strategies for Distributed Systems*, IFIP Transaction A - Computer Science and Technology, Vol. 39, 1993.

[John97] D.B. Johnson, *Mobility Support in IPv6*, IETF Internet Draft, 21st November 1997.

[Katz94] R.H. Katz, *Adaptation and Mobility in Wireless Information Systems*, IEEE Personal Communications Magazine, Vol. 1, No. 1, 1994.

[Koya93] K. Koyama, K. Shimizu, H. Ashihara, Y Zhang, H. Kameda, *Performance Evaluation of Adaptive Load Balancing Policies in Distributed Systems*, Singapore International Conference on Networks Information Engineering, Vol.2, 1993.

[Krem92] O. Kremien, J. Kramer, *Methodical Analysis of Adaptive Load Sharing Algorithms*, IEEE Transaction on Parallel and Distributed Systems, Vol. 3, No. 6, November 1992.

[Kris95] P. Krishnan, P.M. Long, J.S. Vitter, *Learning to Make Rent-to-Buy Decisions with Systems Applications*, Proc. of the 12th International Machine Learning Conference, 1995.

[Krue88] P. Krueger, M. Livny, *A Comparison of Preemptive and Non-Preemptive Load Distributing*, Proc. of the 8th International Conference on Distributed Computing systems, June 1988.

[Krue91] P. Krueger, R. Chawla, *The Stealth Distributed Scheduler*, Proc. of the 11th International Conference on Distributed Computing Systems, 1991.

[Krue94] P. Krueger, N.G. Shivaratri, *Adaptive Location Policies for Global Scheduling*, IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994.

[Lett97] P. Lettieri, C. Fragouli, M.B. Srivasta, *Low Power Error Control for Wireless Links*, Proc. of the Mobicom '97, Budapest, Hungary, 1997.

[Li94] K. Li, F. Kumpf, P. Horton. T. Anderson, *A Quantitative Analysis of Disk Drive Power Management in Portable Computers*, Proc. of 1994 Winter USENIX Conference, San Francisco, January 17-21, 1994.

[Long96] S.L. Long, R. Kooper, G.D. Abowd, C.G. Atkeson, *Rapid Prototyping of Mobile Context-Aware Applications : The Cyberguide Case Study*, Proc. of Mobicom '96,

Rye NY, USA, 1996.

[Lorc96] J.R. Lorch, A.J. Smith, *Reducing Processor Power Consumption by Improving Processor Time Management in a Single-User Operating System*, Proc. of Mobicom '96, 1996.

[Mars94] B. Marsh, B. Zenel, *Power Measurements of Typical Notebook Computers*, Matsushita Information Technology Laboratory Technical Report MITL-TR-110-94, May 1994.

[Mirc89] R. Mirchandaney, D. Towsley, J.A. Stankovic, *Analysis of the Effects of Delays on Load Sharing*, IEEE Transactions on Computers, Vol. 38, No. 11, November 1989.

[Mirc90] R. Mirchandaney, D. Towsley, J.A. Stankovic, *Adaptive Load Sharing in Heterogeneous Distributed Systems*, Journal of Parallel and Distributed Computing, Vol. 9, No. 4, August 1990.

[Othm97] M. Othman, S. Hailes, *Load Balancing As a Power Management Strategy for Mobile Computers*, Proc of Malaysia International Conference on Communications, November 1997.

[Othm98] M. Othman, S. Hailes, *Power Management Strategy for Mobile Computers Using Load Sharing*, Mobile Computing and Communications Review, Vol. 2, No. 1, January 1998.

[Othm99] M. Othman, S. Hailes, *Emulation of Load Sharing for Power Management on a Wireless LAN*, Submitted for Review.

[Perk97a] C. Perkin, *IP Mobility Support Version 2*, IETF Internet Draft, 12th November 1997.

[Perk97b] C. Perkin, *Route Optimisation in Mobile IP*, IETF Internet Draft, 20th November, 1997.

[Rudn98] A. Rudenko, P. Reiher, G.J. Popek, G.H. Kuenning, *Saving Portable Computer Battery Power through Remote Process Execution*, Mobile Computing and Communications Review, Vol.2, No. 1, January 1998.

[Ruem92] C. Ruemmler, J. Wilkes, *UNIX Disk Access Patterns*, HP Laboratories Technical Report HPL-92-152, December 1992.

[Ruln96] J.M. Rulnick, N. Bambos, *Mobile Power Management for Maximum Battery Life in Wireless Communication Networks*, Proc. of the Infocom '96, 1996.

[Shek94] S. Shekar, D.R. Lin, *Genesis and Advanced Traveller Information systems (ATIS) : Killer Applications for Mobile Computing ?*, MOBIDATA : An Interactive Journal of Mobile Computing, Vol.1, No. 1, November 1994. Available at http://www.cs.rutgers.edu/~badri/journal.

[Shen92] S.Sheng, A. Chandrakasan, R.W. Brodersen, *A Portable Multimedia Terminal*, IEEE Communications Magazine, December 1992.

[Shin89] K.G. Shin, Y.C. Chang, *Load Sharing in Distributed Real-Time Systems with*

State-Change Broadcasts, IEEE Transactions on Computers, Vol. 38 No. 8, August 1989.

[Shor95] J. Short, R. Bagrodia, L. Kleinrock, *Mobile Wireless Network System Simulation*, Mobicom '95, November 1995.

[Stan84] J.A. Stankovic, *Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms*, Computer Networks, Vol. 8 No. 3, June 1984.

[Stan85] J.A. Stankovic, *Stability and Distributed Scheduling Algorithms*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 10, October 1985.

[Star97] T. Starner, S. Mann, B, Rhodes, J. Levine, J. Healey, D. Kirsch, R.W. Picard, A. Pentland, *Augmented Reality Through Wearable Computing*, Presence - Teleoperators and Virtual Environment, Vol.6, No. 4, 1997.

[Sven90] A. Svensson, *History, an Intelligent Load Sharing Filter*, Proceedings of the 10th International Conference on Distributed Computing Systems, 1990.

[Wald92] C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, W.S. Stronetta, *Spawn : A Distributed Computational Economy*, IEEE Transactions on Software Engineering, Vol. 18, No. 2, February 1992.

[Wang85] Y.T. Wang, R.J.T. Morris, *Load Sharing in Distributed Systems*, IEEE Transactions on Computers, Vol. C-34, No. 3, March 1985.

[Wang93] C.J. Wang, P. Krueger, M.T. Liu, *Intelligent Job Selection for Distributed Scheduling*, Proc. of the 13th International Conference on Distributed Computing Systems, 1993.

[Wats94] T. Watson, *Application Design for Wireless Computing*, Proc. of IEEE Workshop on Mobile Computing Systems and Applications, December 1994.

[Watt96] J. Watts, M. Rieffel, S. Taylor, *Practical Load Balancing for Irregular Problems*, Proceedings of IRREGULAR '96, Parallel Algorithms for Irregularly Structured Problems, 1996.

[Weis93] M. Weiser, *Some Computer Science Issues in Ubiquitous Computing*, Communications of the ACM, Vol. 36, No. 7, July 1993.

[Weis94] M. Weiser, B. Welch, A. Demers, S. Shenker, *Scheduling for Reduced CPU Energy*, Proc. of USENIX 1994 Operating Systems Design and Implementation Symposium, Monterey, CA, November 1994.

[Zhan95] Y. Zhang, K. Hakozaki, H. Kameda, *Effectiveness of Adaptive and Static Load Balancing Strategies*, International Workshop on Computer Performance Measurement and Analysis, 1995.

[Zhou88] S. Zhou, *A Trace-Driven Simulation of Study of Dynamic Load Balancing*, IEEE Transaction on Software Engineering, Vol. 14, No. 9, September, 1988.