UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

AN ENERGY-EFFICIENT CONCURRENCY CONTROL ALGORITHM FOR MOBILE AD-HOC NETWORK DATABASES

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

ZHAOWEN XING
Norman, Oklahoma
2011

AN ENERGY-EFFICIENT CONCURRENCY CONTROL ALGORITHM FOR MOBILE AD-HOC NETWORK DATABASES

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Le Gruenwald, Chair

Dr. John Albert

Dr. Mohammed Atiquzzaman

Dr. Qi Cheng

Dr. Changwook Kim

ACKNOWLEDGEMENTS

I would like to take the opportunity to thank the people who guided and supported me during this PhD study. Without their contributions, this research would not have been possible.

First, I would like to thank my PhD advisor, Dr. Le Gruenwald. During my PhD study, Dr. Gruenwald taught me how to do research, and she was always patient and helpful whenever her guidance and assistance was needed. As an international student, I found technical writing was difficult. She contributed much of her valuable time to help me improve my writing skills. She also gave advice on how to communicate and cooperate with other researchers as a professional. Therefore, I do want to show my deep appreciation to her.

I am also grateful to Dr. Albert, Dr. Atiquzzaman, Dr. Cheng, and Dr. Kim for serving on my PhD committee, and thank them for spending time reviewing this work and giving valuable suggestions and comments on my work. I also appreciate valuable discussions with my former committee member, Dr. Yifei Dong, visiting professor Dr. Keat Keong Phang (Malaya University, Malaysia) and visiting professor Dr. Seokil Song (Chungju National University, Republic of Korea).

There was help from the research teams inside and outside of OU. I still remember the time when Jason Foss, Nickolas Hunter, Syed Maruful Huq, and Ying Su worked together with me. It was my great pleasure to work with them in our database research lab. Their suggestions and comments also encouraged me in these years. Besides this, I also want to thank Michele Mastrogiovanni (University of Roma, Italy), who provided valuable information about clustering MANETs.

Therefore, I want to thank all of them too.

I would like to show my deep appreciation to my parents, parents-in-law, brothers (above all, Zhaoyun Xing) and sisters for their continuous support during all these years. Last, but not least, my wife, Suyu Li, also contributed much of her time and efforts to support me during my study. Without any one of them, the work would not have been possible.

To my beloved wife Suyu, and to our beloved sons Jerry and Jasper: thank you for all the love, support, and happiness that you brought to me.

This work was supported in part by funding from the National Science Foundation (NSF) Grant No. IIS-0312746. This support is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the rapid growth of the wireless networking technology and mobile computing devices, there is an increasing demand for processing mobile database transactions in mission-critical applications such as disaster rescue and military operations that do not require a fixed infrastructure, so that mobile users can access and manipulate the database anytime and anywhere. A Mobile Ad-hoc Network (MANET) is a collection of mobile, wireless and battery-powered nodes without a fixed infrastructure; therefore it fits well in such applications. However, when a node runs out of energy or has insufficient energy to function, communication may fail, disconnections may happen, execution of transactions may be prolonged, and thus time-critical transactions may be aborted if they missed their deadlines. In order to guarantee timely and correct results for multiple concurrent transactions, energy-efficient database concurrency control (CC) techniques become critical. Due to the characteristics of MANET databases, existing CC algorithms cannot work effectively.

In this dissertation, an energy-efficient CC algorithm, called Sequential Order with Dynamic Adjustment (SODA), is developed for mission-critical MANET databases in a clustered network architecture where nodes are divided into clusters, each of which has a node, called a cluster head, responsible for the processing of all nodes in the cluster. The cluster structure is constructed using a novel weighted clustering algorithm, called MEW (Mobility, Energy, and Workload), that uses node mobility, remaining energy and workload to group nodes into clusters and select cluster heads. In SODA, in order to conserve energy and balance energy consumption among servers so that the lifetime of the network is prolonged, cluster heads are elected to work as coordinating servers. SODA is based on optimistic CC

to offer high transaction concurrency and avoid unbounded blocking time. It utilizes the sequential order of committed transactions to simplify the validation process and dynamically adjusts the sequential order of committed transactions to reduce transaction aborts and improve system throughput.

Besides correctness proof and theoretical analysis, comprehensive simulation experiments were conducted to study the performance of MEW and SODA. The simulation results confirm that MEW prolongs the lifetime of MANETs and has a lower cluster head change rate and re-affiliation rate than the existing algorithm MOBIC. The simulation results also show the superiority of SODA over the existing techniques, SESAMO and S2PL, in terms of transaction abort rate, system throughput, total energy consumption by all servers, and degree of balancing energy consumption among servers.

CHAPTER 1

INTRODUCTION

In this chapter, we first introduce MANETs and their applications. We then discuss concurrency control in transaction processing, why energy efficiency is necessary in transaction processing and research challenges due to MANETs. Finally we state our objectives and contributions of our research and the outline of this dissertation.

1.1 Mobile Ad-hoc Networks and Their Applications

A mobile ad hoc network (MANET) is a collection of battery-powered mobile nodes (or hosts) connected by relatively lower bandwidth wireless links. Each node has an area of influence called cell, only within which other nodes can receive its transmissions [Fife, 2003]. Due to no fixed infrastructures, all nodes can move freely, the network topology may change rapidly and unpredictably over time, and nodes have to form their own cooperative infrastructures. Thus, each node operates as an autonomous end system and a router for other nodes in the network. As no fixed infrastructure is required, MANET databases can be deployed in a short time and mobile users can access and manipulate data anytime and anywhere, and they become an attractive solution to handle mission-critical database applications, such as disaster response and recovery systems [Catarci, 2008; Lu, 2007; Alampalayam, 2009], mobile telemedicine systems [Viswacheda, 2007] and military operations like battlefields [Alampalayam, 2009].

1.1.1 Disaster responses and disaster discovery using hybrid MANETS

When a hurricane, earthquake or tsunami occurs, disaster response and recovery are usually hampered by communications failure because the incumbent communications infrastructures have most likely been damaged or destroyed during these disasters. A mobile ad-hoc communications infrastructure, with support for multimedia traffic such as voice over IP and video streaming, must be rapidly developed to support the command, control and communication needs of the rescue and recovery operations [Lu, 2007].

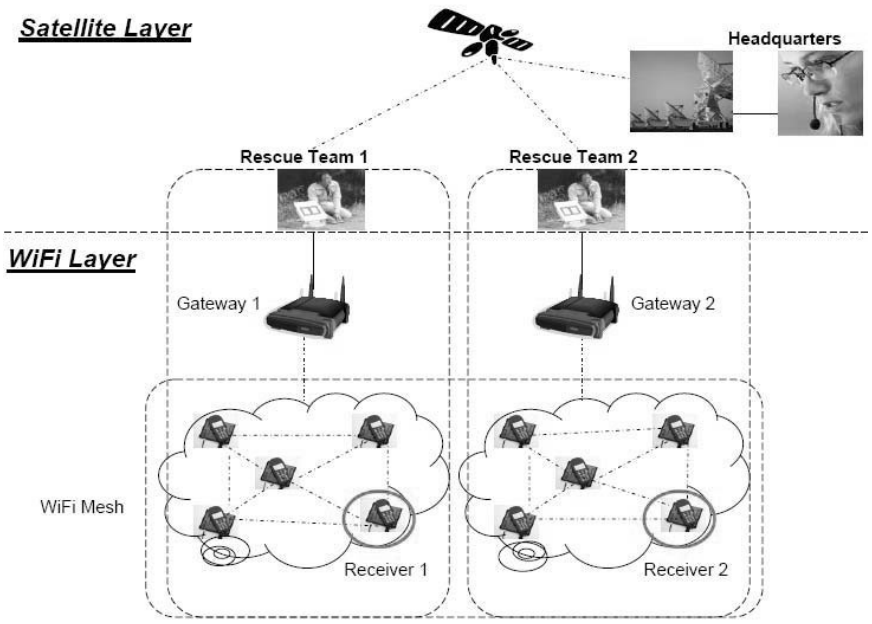

Figure 1.1 Disaster rescue application [Lu, 2007]

Figure 1.1 shows a system architecture that consists of two tiers: satellite layer and WiFi layer [Lu, 2007]. In the satellite layer, multiple rescue teams can communicate among themselves as well as with their headquarters via satellite links. The team

members in each team (group) communicate among themselves using a multi-hop MANET, but team members from different teams do not communicate with each other directly.

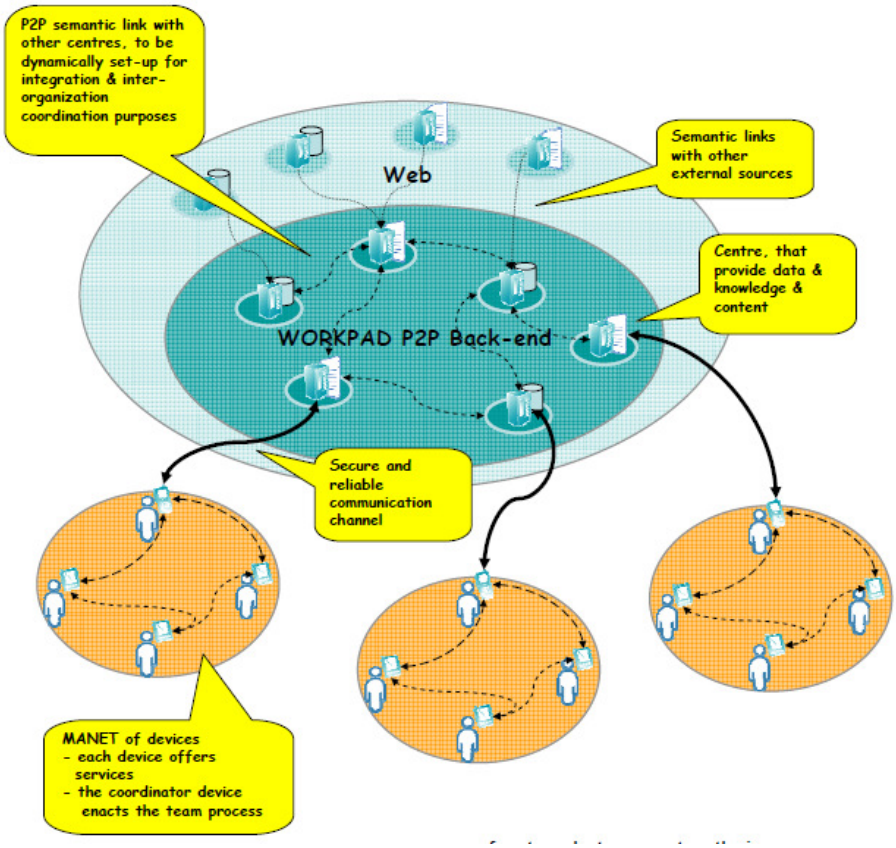

Figure 1.2 Workpad for emergency management [Catarci, 2008]

In order to design successful communication technology architectures for emergency management, Workpad employs user-centered techniques from human–computer interaction paradigms. User-centered design relies on continuous interaction with end users, so that designers understand how organizations are arranged during

disasters, what information is critical, and how teams exchange information among themselves and with their operational centers as shown in Figure 1.2. Workpad designers collaborated with the Civil Protection of Calabria, Italy, and involved interviewing officers and generic actors from the organizations most critical to emergency management in that region. They also studied the emergency management structures of different European countries and found that most of them had emergency management structures similar to Italy's [Catarci, 2008].

1.1.2 Mobile telemedicine system

Remote areas lack telecommunication infrastructures, thus, it is difficult to provide medical services in time and quality manner. Telemedicine is defined as the delivery of medical healthcare and medical expertise using a combination of telecommunication technologies. Telemedicine systems can support applications ranging from video conference to providing diagnostics, high quality image and still-image, and medical database records. Applications in Telemedicine are classified into basic and extended services. Basic services applications are digital electrocardiogram, oxy-meter, patient database records, and location information based on GPS technologies, while extended services applications are complete multimedia services. Both services can be used in rural areas based on wireless communication despite the fact that hospitals have wired communication. [Viswacheda, 2007].

Figure 1.3 shows two MANET based sub networks (MANET1 and MANET2). In a sub network, mobile nodes (MN) communicate directly with each other in peer-to-peer connections, and each MN acts as a router for other nodes. The health practitioners use

MNs to request/transmit patient's data from/to the healthcare center or mobile ambulance. The server in the mobile ambulance functions as a local database if there is no connection to the health care center [Viswacheda, 2007].

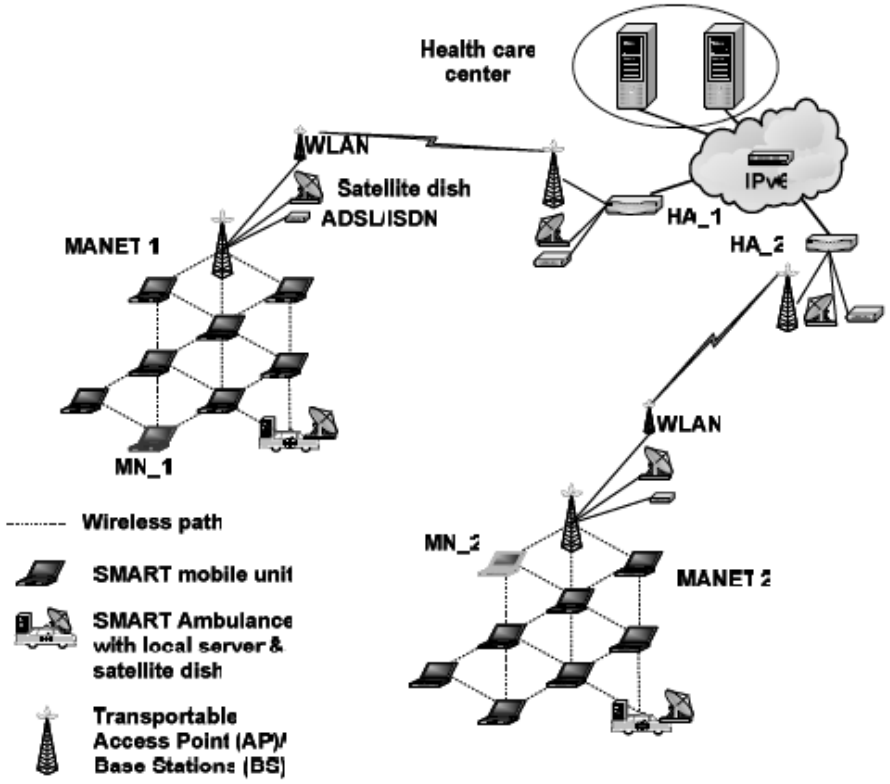

Figure 1.3 Mobile telemedicine system [Viswacheda, 2007]

1.1.3 Military operations in battlefields

In a battlefield, soldiers are organized into platoons, stay close to their tanks or humvees, and share information through them. In other words, there is a leader or a group of leaders who tells everybody where and how to move or in which area to work as shown in Figure 1.4 [ATAC, 2005]. In general, their movements are driven by tactical

reasons. Due to this, the units normally use the optimal path to a destination. The destinations depend on the work area that is based on tactical issues. The tactics as well as the scene are usually hierarchically organized. Typically, the site is divided into different tactical areas. Each unit belongs to one of these areas. Thus, the area in which a unit moves depends on tactical issues but is restricted to one specific area [Aschenbruck, 2008].

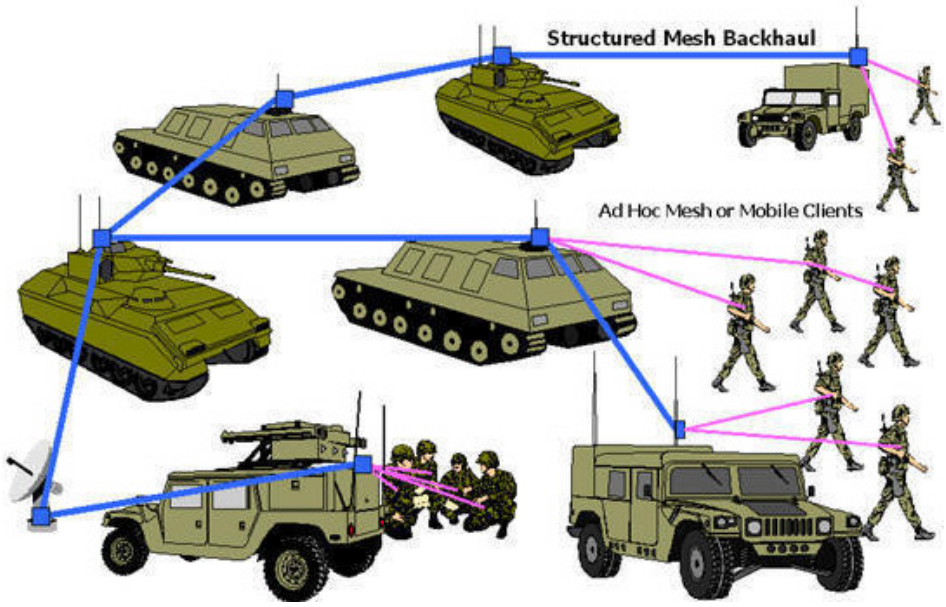

Figure 1.4 Military operations in a battlefield [ATAC, 2005]

1.1.4 Common characteristics of mission-critical MANET applications

The MANET applications described above have the following common characteristics:

- These applications are mission-critical as they are related to human lives, thus, transactions must be executed not only correctly but also within their deadlines.
- These applications are semantics-based and users are already organized into logic

groups.

- Groups share information among each other through the help of satellites, but satellites are not always available [Viswacheda, 2007].
- Users can move randomly within application area. However, their movement is within a finite range and they stay within the groups. In other words, the application area is divided into different tactical areas. Each group belongs to one of these areas, and is restricted to one specific area.

1.2 Concurrency Control in Transaction Processing and Why Energy Efficiency Is Necessary

Database management systems ensure convenient and efficient access of databases for their users, and transaction Manager (TM) is a vital component in the system. TM is responsible for ensuring that a database remains in a correct/consistent state even if system fails. Also TM applies concurrency control to ensure that concurrent transaction executions proceed without interleaving. CC is the activity of preventing transactions from destroying the consistency of the database while allowing them to run concurrently, so that the throughput and resource utilization of database systems are improved and the waiting time of concurrent transactions is reduced [Silberschatz, 2005]. A CC technique is pessimistic if it avoids conflicts at the beginning of transactions, or optimistic if it detects and resolves conflicts right before the commit time.

Because of the mobility and portability, mobile nodes have severe resource constraints in terms of capacity of battery, memory size and CPU speed [Fei, 2008]. As the battery capacity is limited, it compromises the ability of each mobile node to support

services and applications [Chlamtac, 2003]. Also the battery technology is not developed as rapidly as the mobile devices and wireless technologies, so that the limited battery lifetime is always a bottleneck for the development of improved mobile devices [Sklavos, 2007]. Therefore, a suitable CC algorithm for MANET databases should be energy-efficient. Here, energy efficiency refers to the amount of service work that a system can accomplish in the least amount of energy consumption when its battery capacity is limited [Fei, 2008].

1.3 Research Challenges due to MANETs

The flexibility and convenience in a MANET introduces a number of constraints/characteristics which impact transaction processing and are listed below. As a result of these constraints and of the fact that servers are also mobile, CC techniques for cellular mobile databases cannot be directly applied in MANET environments. In a cellular mobile database, servers are generally static nodes running on a wired network, while clients are mobile nodes communicating with servers through static mobile support stations to have their transactions processed.

- *Mobility*: When a node roams, its network and physical location change dynamically, and at the same time, the states of transactions and accessed data items have to move along with the node.
- *Low bandwidth*: Wireless network bandwidth is much lower than its wired counterpart. For example, the widely used 802.11b wireless card has a maximum data rate of 11 Mbit/s [Hofmann, 2006]; however, currently an affordable Gigabit

Ethernet card realizes a maximum data rate of 1000 Mbit/s [StarTech, 2011]. Thus, within the cell of a node, inside neighbors, which are defined as nodes within 1-hop communication of each other, have to share and compete for the same channel. If someone fails, it may keep sending requests until timeout. This low bandwidth can result in communication delays, a high risk of disconnections and long-lived transactions.

- *Multi-hop communication*: In a MANET, nodes can communicate with each other either directly or via other nodes that function as routers. When communication requires more hops, more energy and bandwidth are consumed, and more execution time is needed to complete transactions.

- *Limited energy*: Because of the mobility and portability, clients and servers have severe resource constraints in terms of capacity of battery. Once a node runs out of energy or has insufficient energy to function, communication fails, disconnections happen, execution of transactions is prolonged, and some time-critical transactions may be aborted if they missed their deadlines.

- *Frequent disconnections*: A node is disconnected when it roams freely and is out of the transmission range of all its neighbors; or it fails to compete for the channels of popular neighbors; or its battery runs out; or it runs into some failures. It is normal for a node to become disconnected in a MANET and the disconnected nodes may reconnect after some time. When disconnections happen, more transactions may be delayed or blocked, and even aborted if they are real-time and miss their deadlines.

- *Long-lived transactions*: Due to wireless communication delay, less processing power, frequent disconnections and unbounded disconnection time, transactions in MANET databases tend to be long-lived. When the execution is prolonged, the probability of conflicts with other executing transactions becomes higher and, consequently, transactions are likely blocked if a pessimistic CC method applies, or aborted if an optimistic CC method is in use.

1.4 Objectives and Contributions of this Research

CC research in MANET databases is still in an early stage. To the best of our knowledge, only one MANET CC algorithm, called Semantic Serializability Applied to Mobility (SESAMO) [Brayner, 2005], has been proposed. SESAMO does not take energy efficiency into account, and is based on semantic serializability, which requires that not only databases on mobile nodes be disjoint but also updates on a database depend only on the values of the data in the same database. Therefore, in SESAMO, transaction atomicity and global serializability can be relaxed. Transaction atomicity ensures that a transaction either terminates normally to make all of its effects permanent or is aborted to have no effect at all [Bernstein, 1987]. Global serializability requires that all global transactions be serialized in the same order at all the participating servers at which they execute [Dirckze, 2000]. However, in MANET databases for mission-critical applications, the assumption for semantic serializability does not hold because each database depends on each other due to the organizational structure of the applications. For example, in a disaster rescue scenario, before sending firefighters out to pursue some actions, the status of their equipment has to be checked to ensure atomic decisions

[Obermeier, 2009], where the firefighter database may be stored on one mobile server, and the equipment database may be stored on another mobile server. In a battlefield scene, before a tank fires its cannon, the locations of their own soldiers have to be checked to ensure their safety, where the tank database may be stored on one mobile server, and the soldiers' information may be located on another mobile server.

Except for SESAMO, all other techniques are designed for cellular mobile databases, in which powerful servers are static and broadcast is heavily used to disseminate data from servers to clients, thus, they are not suitable for MANET databases either. With broadcasting, static servers transmit latest data items to clients periodically regardless of their demands, and then the clients read the data items of interest from the broadcast channel [Choi, 2006]. In this dissertation, our objective is to design a CC algorithm that is energy-efficient and suitable for mission-critical MANET databases that require global serializability. In other words, our new algorithm should minimize energy consumption of mobile nodes, clients as well as servers, and balance energy consumption among servers, so that servers with low energy do not run out of energy quickly, and thus, the number of disconnections and transaction aborts due to low energy or energy exhaustion can be reduced and system throughput can be improved as well.

In this dissertation, an energy-efficient CC algorithm, called Sequential Order with Dynamic Adjustment (SODA), is developed for mission-critical MANET databases in a clustered network architecture where nodes are divided into clusters, each of which has a node, called a cluster head, responsible for the processing of all nodes in the cluster. The cluster structure is constructed using a novel weighted clustering algorithm, called MEW (Mobility, Energy, and Workload), that uses node mobility, remaining

energy and workload to group nodes into clusters and select cluster heads. In SODA, in order to conserve energy and balance energy consumption among servers so that the lifetime of the network is prolonged, cluster heads are elected to work as coordinating servers. SODA is based on optimistic CC to offer high transaction concurrency and avoid unbounded amount of blocking time. It utilizes the sequential order of committed transactions to simplify the validation process and dynamically adjusts the sequential order of committed transactions to reduce transaction aborts and improve system throughput.

Besides correctness proof and theoretical analysis, comprehensive simulation experiments were conducted to study the performance of MEW and SODA. The simulation results confirm that MEW prolongs the lifetime of MANETs and has a lower cluster head change rate and re-affiliation rate than the existing algorithm MOBIC. The simulation results also show the superiority of SODA over the existing techniques in terms of transaction abort rate, system throughput, total energy consumption by all servers, and degree of balancing energy consumption among servers.

1.5 The Outline of Dissertation

The rest of the dissertation is organized as follows. Chapter 2 first reviews concurrency control techniques for MANET databases and cellular mobile databases, and then reviews clustering algorithms to elect cluster heads and form clusters in MANETs. Chapter 3 describes the MANET database architecture used in the proposed algorithm, and how to construct this architecture using our clustering algorithm MEW (Mobility, Energy, and Workload). Chapter 4 presents our concurrency control algorithm, SODA.

Chapter 5 presents the performance evaluation of MEW compared with MOBIC using the NS-2 simulator [Basagni, 2006]. Chapter 6 discusses the performance evaluation of SODA compared with SESAMO and S2PL (Strict 2-Phase Locking) [Bernstein, 1987] using the AweSim simulation language [Pritsker, 1999]. Finally Chapter 7 concludes the dissertation with future research.

CHAPTER 2

LITERATURE REVIEW

The contributions of this research are a concurrency control (CC) algorithm for MANET databases and a clustering algorithm for MANETs, thus in this chapter, we review not only CC design issues and CC techniques for mobile databases but also clustering algorithms for MANETs.

2.1 Issues in Designing Concurrency Control Algorithms for Mobile Databases

In this section, we discuss the general issues and application-dependent issues that need to be addressed in the design of CC algorithms for mobile databases.

2.1.1 General issues

General issues are those that every CC algorithm for mobile databases needs to address.

2.1.1.1 Types of concurrency control algorithms

To guarantee the correct results and consistency of databases, the conflicts between transactions can be either avoided, or detected and then resolved. Most of the existing mobile database CC techniques use the (conflict) serializability as the correctness criterion, where serializability requires that the effects of executing a set of

transactions concurrently be equivalent to the effects of executing the same set transactions in some serial order [Bernstein, 1987]. They are either pessimistic if they avoid conflicts at the beginning of transactions, or optimistic if they detect and resolve conflicts right before the commit time, or hybrid if they are mixed. To fulfill this goal, locking, timestamp ordering (TO) and serialization graph testing can be used as either a pessimistic or optimistic algorithm. An improper type of CC algorithm may waste limited system resources like bandwidth and battery power in MANET databases, and cause more transactions aborted.

2.1.1.2 Rules of producing serializability

There are three general rules to produce serializability: locking, timestamp ordering and serialization graph testing [Bernstein, 1987]. In a locking scheme, each data item has a lock associated with it. Before a transaction can access a data item, it must obtain the lock of this data item first; otherwise, it has to wait until other transactions release the lock. In timestamp ordering, a unique timestamp is assigned to each transaction, and transactions are executed based on the order of their timestamps. In serialization graph testing, each transaction is added to the graph as a node, and there is an edge between two nodes if there is a conflicting operation between these two transactions. If there is a cycle in the graph after adding a new node, the newly added transaction is aborted to maintain the serializability. However, locking is not suitable for MANET databases because it is an unnecessary overhead when transactions are read-only. In addition, because of the early prevention, available limited system resources cannot be fully utilized. Due to frequent disconnections in MANETs, timestamp

ordering scheme may abort lots of transactions with smaller timestamps if severs disconnect, reconnect after a while and cannot execute them based on their timestamps. Serialization graph testing is time-consuming because it always requires quadratic time to check serializability [Hwang, 2000].

2.1.1.3 Concurrency control granularity

The granularity of a data item is the size of the data contained in the data item [Bernstein, 1987]. The CC granularity, which is the size of data items used to prevent/detect transaction conflicts, can be a database row, a page, a table or a database. If a typical transaction accesses a small number of rows, then it is advantageous to have row granularity for higher concurrency and fewer aborts. If a transaction typically accesses many rows of the same table, then it is better to have table granularity so that the resources, which are required to prevent conflicts in a pessimistic method or detect conflicts in an optimistic method, can be saved. In other words, higher concurrency and fewer aborts but more resources are required for fine granularity. In contrast, lower concurrency and more aborts but fewer resources are required for coarse granularity; however, more aborts consequently wastes the limited system resources like wireless bandwidth and battery power in MANETs.

2.1.1.4 Mobile system architecture

A mobile system architecture can be classified as either a cellular mobile network architecture or a MANET architecture. A cellular mobile network architecture consists

of fixed nodes and mobile nodes, where only mobile nodes are mobile and battery-powered. Mobile nodes retain their network connections through a wireless interface supported by some fixed nodes known as mobile support station (MSS) [Serrano-Alvarado, 2004]. In a MANET architecture, every node is mobile, wireless and battery-powered, and can communicate with each other directly either through one hop or multiple hops. Unlike cellular mobile databases where servers are static, servers can move freely in MANET databases, so that it is hard to check serializability among mobile servers.

2.1.1.5 Location of concurrency control manager(s)

A CC manager (or scheduler) is the heart of a CC algorithm because every data request or final transaction validation before the commit time in an optimistic algorithm must go through it. Depending on the architecture of a database system (centralized or distributed) and the design of a CC algorithm, a CC manager can be either centralized or distributed as well. For instance, in a distributed database system, if there is only one CC manager that is located at one node to schedule all transactions, then this CC manager is centralized. However, in MANET databases, besides the bottleneck problem, a centralized CC manager may not always be available due to frequent disconnections. Also the computation overhead gets worse because of limited battery power, memory and disk space. To resolve these problems and process transactions in a timely manner, there should be more than one CC manager located at different sites, where each CC manager has autonomous processing capability on local transactions that access or update data in only one server, and may coordinate with each other to execute global

transactions that access or update data in more than one servers. However, the tradeoff of distributed CC managers is the communication overhead in terms of handshaking among them for the cooperation or broadcasting data and invalidation information among them.

2.1.1.6 Improving system performance

In order to improve transaction throughput and response time and to effectively utilize system resources, it is natural to allow multiple concurrent transactions to be executed simultaneously. However, when a transaction does not complete its execution successfully because of transaction failure or database inconsistency, it is aborted or restarted and, consequently, the transaction execution time and system resources are wasted. It is expensive to abort or restart transactions in MANET databases because this would consume the limited bandwidth, battery power and storage and, consequently, more transactions are aborted due to the MANET database characteristics discussed in Chapter 1. Therefore, a good CC algorithm for MANET databases should offer high transaction concurrency and avoid unnecessary aborts, so that more transactions will have chances to complete and commit in a timely manner and nodes will not waste their scarce resources, especially power, which may subsequently cause disconnections.

2.1.1.7 Cascading abort

When a transaction aborts, the recovery scheme must restore the database to the consistent state that existed before the transaction started. It is necessary to ensure that

any transaction that has read data written by the aborted transaction is also aborted. This phenomenon is called cascading rollback [Bernstein, 1987]. Cascading abort usually can be avoided by not allowing transactions to read un-committed data items in the CC design; otherwise the consistency property is not preserved and it also results in a significant amount of transaction undo work, which is expensive in MANET databases because once limited battery power is consumed, it cannot be replaced until the battery is recharged. When battery power is low or runs out, communication may fail or transactions cannot be processed.

2.1.1.8 Insert and delete operations

Besides read and write (update) operations, an insert operation inserts a new data item with an initial value into the database; and a delete operation deletes a data item from the database. Read, write, insert and delete can be conflict operations and result in the phantom phenomena or logic errors when any two of them execute in different orders [Silberschatz, 2005]. For instance, a logic error will occur when a data item is read after it is deleted or before it is inserted. To avoid logic errors, insert and delete operations have to be treated like write operations; otherwise, corresponding transactions have to be aborted or restarted, and aborts are expensive in MANET because they waste limited system resources like bandwidth and battery power, which were discussed in Chapter 1.

2.1.1.9 Level of consistency

In some applications (e.g. statistical analysis or traffic information), in order to

increase the degree of concurrency and reduce transaction abort rate, it is acceptable to relax the transaction consistency requirement by reading stale data; however, this loose consistency may compromise the accuracy of the database. In mission-critical applications like disaster rescue and battle field of military operation, consistency cannot be relaxed and accurate data are required; consequently, it becomes a challenging task in the MANET environment because every node is mobile, wireless and battery-powered.

2.1.1.10 Transaction model

Depending on applications, a transaction can be flat or nested [Moss, 1985]. The flat model is simpler to implement, but rolling back the entire transaction and starting from the very beginning is the only option when some part of the transaction fails. This would definitely waste lot of the limited system resources in MANETs because every mobile node has limited bandwidth and battery power. In other words, when some nodes occupy the wireless bandwidth to process those transactions that are aborted later, other nodes have to wait for some time period and then try to connect again; once the limited battery power is consumed, it cannot be replaced until the battery is recharged. In a nested model, a transaction can be dynamically decomposed into a hierarchy of sub-transactions (child transactions), and this decomposition grants the opportunity to rollback/restart only the failed sub-transaction rather than the entire transaction. This of course reduces the amount of completed work that is wasted by a flat model, but the tradeoff is that a nested model complicates the ACID (Atomicity, Consistency, Isolation and Durability) properties of transactions [Özsu, 1999] in MANETs. For instance, some sub-transactions committed early to improve concurrency, but their parent transactions

cannot complete and cannot rollback those already committed sub-transactions due to frequents disconnections. Then how to guarantee transaction atomicity and consistency has to be addressed.

2.1.2 Application-dependent issues

Not all mobile applications have the same requirements for CC algorithm design. In this section, we discuss the issues that are application-dependent.

2.1.2.1 Global serializability

The consistency of a global transaction may not be guaranteed although its sub-transactions are serializable at each participating server because the serialization orders may be different at different participating servers. Thus, in a distributed client-server database system, serializability has to be maintained at not only the local level but also the global level. To achieve global serializability, due to MANET characteristics, it is crucial to address the following:

- How to require less communication because every node is wireless and more communication consumes more battery power
- How to utilize battery power efficiently since every node has limited battery power
- How to allow nodes to effectively cooperate with each other because frequent disconnections are normal in MANET environments.

2.1.2.2 Degree of local autonomy

In a heterogeneous database (or multi-database) system that allows local autonomy, each local database system has the right to share internal data or not, and choose its own mechanisms for data and transaction management. Local autonomy is preserved if the local site does not need to be modified in order to coordinate global serializability. When the local site has more autonomy, it can effectively utilize its local system resources. However, in MANET mission-critical applications like disaster rescue and military operations, global serializability is required; so the local autonomy cannot be preserved; but each system still needs to share as few data items as possible to achieve the global serializability. This would allow limited wireless bandwidth and battery power to be saved because fewer shared data would require less communication.

2.1.2.3 Cached/Replicated data

In order to improve data access time and availability, caching/replication is a process that creates several duplications of the same data and stores one copy at a different site. Caching slightly differs from replication in that cached data is only available to the site where the data is cached. However, in MANETs, since every node is battery-powered, to save and balance the power consumption of primary copy servers, which maintain the primary copies of data, must be addressed; otherwise, no service would be provided when those servers run out of power. Servers with frequent disconnections (due to mobility or power failure) should not be considered as primary

copy servers or caching/replication servers because it is difficult to maintain cached/replicated data consistency and guarantee data availability.

2.1.2.4 Real-time applications

A real-time transaction is defined as a transaction that must be executed within its deadline. In some cases, data items associated with real-time transactions have temporal constraints, that is, they remain valid only within a certain time interval. These data are called temporal data. Due to frequent disconnections, unbounded disconnection time and long-lived transactions in MANETs, how to meet transaction deadlines (and how to process data within their valid time intervals if temporal data exist) is critical for real-time applications in addition to guaranteeing database consistency.

2.2 Concurrency Control Techniques for MANET Databases and Cellular Mobile Databases

In this section, most recently published CC algorithms for mobile databases are reviewed according to MANET database characteristics. These algorithms are classified into three categories based on their types: pessimistic, optimistic and hybrid.

2.2.1 Pessimistic CC methods

In a pessimistic CC method, many transactions are assumed to conflict with each other. Each data access (read or write) by a transaction is checked for conflicts, and conflicting transactions are blocked, restarted or aborted. A lot of recent CC research in

mobile databases use this method and apply a locking scheme to produce transaction serializability. However, this method is not suitable for high volumes of transactions, and it is an unnecessary overhead when transactions are read-only. Also because of this early prevention, available limited system resources cannot be fully utilized. As transaction execution is prolonged and disconnection time is unbounded in MANET databases, the possibility of conflicts among concurrent transactions increases as well. When there are more conflicts, more transactions will be blocked, restarted, or directly aborted.

2.2.1.1 Semantic Serializability Applied to Mobility

Semantic Serializability Applied to Mobility (SESAMO) [Brayner, 2005] is proposed for MANET databases. SESAMO is based on the semantic serializability, which assumes that databases are disjoint and updates on a database only depend on values of data in the same database; therefore, transaction atomicity and global serializability can be relaxed. However, in SESAMO, global transactions still need be serialized at each site using strict 2PL, while at the same time, each site must maintain the consistency of its own local database. The limited bandwidth is saved and transaction execution time is reduced because global serializability is automatically guaranteed without coordination among servers, and the locks held by the sub-transactions of a global transaction can be released once they finish at the local sites. However, SESAMO may fail in MANET applications in which global transactions conflict with each other because it assumes "*Any two given mobile multi-database transaction schedulers do not schedule any transaction in common*" [Brayner, 2005]. Due to no coordination among

servers, database consistency is not preserved when some sub-transactions cannot commit along with others due to disconnections.

2.2.1.2 Look-Ahead Protocol

To maintain data consistency of broadcast data in mobile environments and overcome repeatedly restarting update transactions, Look-Ahead Protocol (LAP) is proposed in [Lam, 2005]. In LAP, update transactions are classified into hopeful transactions and hopeless ones. Hopeless transactions are those that can not commit before their deadlines, and are aborted as earlier as possible to save system resources and reduce data locks, while hopeful transactions can continue to execute their read and write operations using the 2PL algorithm. Unfortunately, in MANETs, because of node disconnections, locked data may be unavailable for an unbounded amount of time.

2.2.1.3 Multi-Version Transaction

A Multi-Version Transaction (MV-T) processing model and a deadlock-free concurrency control algorithm based on the multi-version 2-phase locking scheme are introduced in [Madria, 2007] for mobile database systems. A successful mobile transaction (MT) goes through three states (start, commit that is different from the commit in database systems, and terminate). A MT can start and commit at a mobile node but it terminates only at one of the database servers. A read operation is never blocked because it always gets the last committed or terminated version of data. In addition to read-lock and write-lock, a verified-lock is introduced to achieve isolation. A

write-lock is converted to a verified-lock after a MT commits locally. When requesting a write-lock, a MT with a higher timestamp may be blocked by one with a lower timestamp that holds a write-lock or a verified-lock, but just like applying the timestamp ordering, the requesting MT is rejected or restarted if it has a lower timestamp, therefore, there is no deadlock. Unfortunately, in MANETs, because of frequent disconnections, locked data may be unavailable for an unbounded amount of time.

2.2.1.4 Single Lock Manager Approach

Single Lock Manager Approach (SLMA) [Moiz, 2007] is proposed for cellular mobile networks, in which a single lock manager resides at a fixed server and handles all lock and unlock requests from mobile clients (or nodes). Transactions are initiated and executed at mobile clients, but required data items are read from the fixed server and final updates are done at the fixed server. To increase the system throughput and save the limited uplink bandwidth, SLMA applies a dynamic timer to roll back transactions if they lock data items too long, and puts them in a round robin queue for the next execution. To recover the failure of the single fixed lock manager, the final updates of transactions are also replicated at other fixed hosts. Unfortunately, in MANET, every node is mobile; so it will be challenging to elect a stable node as the single lock manager.

2.2.1.5 Absolute Validation Interval

A concurrency control approach using Absolute Validation Interval (AVI) is proposed in [Moiz, 2008]. The AVI is the life span of a data item in which it is said to be

valid. Mobile nodes copy data items from fixed hosts to their private memory and execute transactions locally. Fixed nodes maintain the AVIs of all data items, commit update transactions requested by mobile nodes, and provide the invalidation reports to mobile nodes. During the local execution of a mobile transaction *T*, if the current time for accessing data is greater than the total of read timestamp of data item plus AVI, this transaction is restarted. However, in MANETs, since every node is mobile, wireless and battery-powered, it is challenging to elect which nodes to work as the fixed nodes to maintain the AVIs of all data items and provide invalidation reports.

2.2.2 Optimistic CC methods

In contrast with pessimistic CC techniques, an optimistic CC method assumes that not too many transactions conflict with each other and, thus, allows transactions to be executed simultaneously, and delays the serializability check of these transactions until their commit time. This delay provides CC mangers with more information to determine the fate of committing transactions. However, the tradeoff is the overhead of late transaction restart and waste of limited resources if a committing transaction must be aborted. This tradeoff becomes worse when the probability of conflicts among concurrently executing transactions is high due to the prolonged execution of transactions and unbounded amount of disconnection time in MANET databases.

2.2.2.1 Optimistic Concurrency Control with Dynamic Timestamp Adjustment

Based on the same technique – the timestamp interval with dynamic adjustment

[Lee, 1993] like PVTO [Lee, 2002b], Optimistic Concurrency Control with Dynamic Timestamp Adjustment (OCC/DTA) [Choi, 2006] is proposed to process transactions in a mobile centralized broadcast environment. Since less information is transmitted between mobile clients and the centralized server, and the timestamp intervals of validating transactions are adjusted only when they read/write data items, OCC/DTA works like a lightweight version of the PVTO protocol. Thus, OCC/DTA has the same drawbacks as those of PVTO. In addition, one of the validation rules need be justified: *"if a committed transaction T_c already read some data and a validating/active transaction T tries to write the same data, the serialization order between them is T precedes T_c*" [Choi, 2006]. This is because the authors adopt this rule partially from [Lee, 2002a], change it slightly but do not provide their own correctness proof. Unfortunately, in MANETs, since every node is mobile, wireless and battery-powered, how to elect a node to work as the centralized server need be addressed.

2.2.2.2 Multi-Version Optimistic Concurrency Control for Nested Transactions

Multi-Version Optimistic Concurrency Control for Nested Transactions (MVOCC-NT) [Lei, 2008] is proposed to process mobile real-time nested transactions using multi-version of data in mobile broadcast environments. MVOCC-NT also adopts the timestamp interval with dynamic adjustment like OCC/DTA [Choi, 2006] to avoid unnecessary aborts. At mobile clients, all active transactions perform backward pre-validation against transactions committed in the last broadcast cycle at the fixed server. Read-only transactions can commit locally if they pass the pre-validation, but survived update transactions have to be transferred to the fixed server for the final validation.

Since data conflicts are detected early, processing and communication resources are saved. However, in MANETs, since every node is mobile, wireless and battery-powered, it is challenging to elect which node to work as the fixed server to do final validation and periodically broadcast data.

2.2.2.3 2-Phase Optimistic Concurrency Control

Choi et al. propose a 2-Phase Optimistic Concurrency Control (2POCC) [Choi, 2009] to process mobile transactions in wireless broadcast environments. Transaction validation is done in two phases: partial backward validation at mobile clients and final validation (forward validation first and then backward validation) at the static server. To guarantee transaction serializability in both phases, 2POCC validates mobile transactions using the following two rules. If a transaction T_i is serialized before transaction T_j:

1. No overwriting, that is, the writes of T_i should not overwrite the writes of T_j;
2. No read dependency, that is, the writes of T_i should not affect the reads of T_j.

To avoid too many restarts of mobile transactions with the read phase lasting several broadcast cycles, the end of a mobile transaction's read phase and validation phase can be shifted to before the beginning of the next broadcast cycle if the unread data objects were not updated during the current broadcast cycle. Unfortunately, in MANETs, since every node is mobile and battery-powered, electing which node to work as the static server to do final validation will be a challenge.

2.2.3 Hybrid CC methods

A hybrid method is a combination of optimistic and pessimistic CC methods. For instance, an optimistic method may be used to validate global transactions and pessimistic one is applied to verify local transactions in a distributed database system. Thus, all the problems existing in both methods have to be addressed.

2.2.3.1 Partial Global Serialization Graph

Partial Global Serialization Graph (PGSG) [Dirckze, 2000] is introduced to enforce a range of consistency and isolation in the cellular mobile multi-databases environment. In PGSG, before committing a global transaction, a partial global serialization graph is built to check for cycles. The local sites serialize transactions by applying the ticket method proposed in [Georgakopoulos, 1991]. A global data structure moves along with a mobile node when it migrates from one cell to another. When a mobile node disconnects, its status is marked as disconnected, and its Mobile Support Station (MSS) saves all the responses in a structure called ResponseList, and delivers them to this mobile node upon reconnection. If a catastrophic failure occurs during the disconnection, then the status is changed to suspended. In order to minimize erroneous aborts, suspended transactions are not aborted until they obstruct other executing transactions. In order to release resources in a timely manner and tolerate long-lived transactions, compensable sub-transactions can commit before the global ones commit. Unfortunately, in MANETs, since every node is mobile and battery-powered, during disconnections/mobility, electing which neighboring node as the MSS to backup/maintain the information will be a challenge. Also, in case of disconnections,

some predecessor graphs may not be collected, and then the PGSG algorithm has to stop and wait. Also some of MANET applications may not have the compensable transactions or are mission-critical, thus, it is impossible to allow sub-transactions to commit early and roll back later. For instance, to query the location of enemies before firing a missile, the accurate data has to be returned.

2.2.3.2 Mobile Transaction Commit using Serialization Graph/Sequential Order

In [Hwang, 2000], to speed up transaction processing and to reduce wireless communication, mobile clients execute transactions against the local cache and use strict 2PL to serialize transactions. To ensure that cached data are up-to-date, an invalidation report is periodically broadcast by the centralized static database server. Before transactions commit, the commit request must be validated at the centralized database server by applying one of the three algorithms: Mobile Transaction Commit using Serialization Graph (MTC-SG), SeQuential order (MTC-SQ) or MTC-Hybrid. MTC-SG maintains only committed transactions, and builds the serialization graph to guarantee no cycle is involved. In MTC-SQ, every validating transaction is checked if it can be inserted somewhere into the maintained sequential order of committed transactions, and the final order must comply with the serialization order. In MTC-Hybrid, MTC-SQ is applied first, and if it fails, then MTC-SG is employed. When mobile clients migrate from one cell to another, their last invalidation report must be checked to ensure that they receive the latest report. Unfortunately, in MANETs, because of limited energy and bandwidth, no node can process all commit requests and, at the same time, keep broadcasting periodically like the static database server. In MTC-SG/Hybrid, building

the serialization graph and checking for cycles require more time and energy to accomplish. When the centralized database server disconnects, not only mobile clients cannot receive broadcast data and invalidation report, but also commit requests cannot be processed at the server.

2.2.4 Summaries of reviewed CC techniques

In this section, we summarize each discussed design issue and its possible solutions from the existing techniques along with the identified MANET database characteristics (the environment issues), which are reviewed above. For every issue, the following specified terms/answers are expected to distinguish the reviewed techniques.

- *Types of CC Algorithms*: Which type does the CC technique adopt to guarantee the isolation property: pessimistic, optimistic, or hybrid?
- *Rules of Producing Serializability*: Which rule does the CC technique utilize to produce serializability: locking, timestamp ordering, serialization graph testing or serial execution?
- *CC Granularity*: What kind of granularity does the CC technique apply: row level, page level, table level, database level or unspecified?
- *Mobile System Architecture*: Which architecture does the CC technique adopt: cellular mobile network, MANET or mobile hybrid network?
- *Location of CC Manager(s)*: Where are the CC managers located: centralized or distributed?

- *Improving System Performance*: How does the CC technique improve the system performance: response time or abort rate (or restart rate)?
- *Cascading Rollback*: Does the CC technique handle cascading rollback?
- *Insert and Delete Operations*: Does the CC technique handle insert and delete operations?
- *Level of Consistency*: Which level of consistency does the CC technique support: relaxed or strict?
- *Transaction Model*: Which transaction model is utilized in the CC technique: flat or nested?
- *Global Serializability*: Does the CC technique guarantee the isolation property of global transactions?
- *Degree of Local Autonomy*: In order to guarantee the isolation property of global transactions, what does the CC technique do to the autonomy of local database system: preserved, violated or unspecified?
- *Cached/Replicated Data*: Does the CC technique apply any caching/replication scheme?
- *Real-Time Applications*: Does the CC technique support real-time transactions?
- *Mobility*: Does the CC technique address mobility of nodes?
- *Low Bandwidth*: Does the CC technique address low bandwidth?
- *Multi-hop Communication*: Does the CC technique address multi-hop communication?

- *Limited Battery Power*: Does the CC technique address limited battery power?
- *Limited Storage*: Does the CC technique address limited storage?
- *Frequent Disconnections*: Does the CC technique address frequent disconnections?
- *Long-lived Transactions*: Does the CC technique address long-lived transactions?

As shown in Table 2.1, none of the reviewed techniques addresses all the identified MANET database characteristics. In addition, we can observe the following from the table:

- Most techniques are pessimistic and guarantee serializability by using locking. They do not fit well in MANETs databases because blocking time may be unbounded and abort rate may be high as a consequence.
- All techniques adopt cellular mobile networks, except for SESAMO which is designed for MANETs.
- CC managers are distributed in all techniques except for LAP [Lam, 2005] and SLMA [Moiz, 2007].
- All techniques support strict database consistency except for PGSG [Dirckze, 2000], which relaxes database consistency.
- Only PGSG [Dirckze, 2000] and MVOCC-NT [Lei, 2008] are proposed for processing nested transactions, while the other techniques process flat transactions.

- Only PGSG [Dirckze, 2000] and SESAMO [Brayner, 2005] address global serializability, while the other techniques only support local serializability
- Most techniques apply caching/replication schemes to overcome the limited wireless bandwidth, but these schemes will not work without the data broadcast by MSSs, which are static and have unlimited energy and high bandwidth.
- Only LAP [Lam, 2005] and MVOCC-NT [Lei, 2008] support real-time applications.
- No technique addresses multi-hop communication.
- Some techniques which are proposed for cellular mobile networks take into account mobility, low bandwidth, limited energy or frequent disconnections. However, these cannot be done without applying caching/replication and heavily relying on MSSs to either broadcast data periodically for mobile nodes or process transactions on behalf of mobile nodes. In MANETs, it is impossible for any node to play the same role like a MSS.
- SESAMO [Brayner, 2005] addresses low bandwidth and long-lived transactions, but both of them are accomplished by assuming that global transactions do not conflict with each other.
- All techniques support long-lived transactions by either using cached/replicated data or partial validation at mobile nodes.

Table 2.1 Summary of the reviewed CC techniques and issues

Techniques/ Issues		SESAMO [Brayner, 2005]	LAP [Lam, 2005]	MV-T [Madria, 2007]	SLMA [Moiz, 2007]	AVI [Moiz, 2008]	OCC/DTA [Choi, 2006]	MVOCC-NT [Lei, 2008]	2POCC [Choi, 2009]	PGSG [Dirckze, 2000]	MTC-SG/SQ [Hwang, 2000]
General Issues	Type of CC Algorithm	Pessimistic	Pessimistic	Pessimistic	Pessimistic	Pessimistic	Optimistic	Optimistic	Optimistic	Hybrid	Hybrid
	Rule of Producing Serializability	Locking	Locking	Timestamp Ordering and locking	Locking	Timestamp ordering	Timestamp Ordering	Timestamp Ordering	Timestamp ordering	Timestamp ordering, and serialization graph testing	Locking, timestamp ordering and serialization graph testing
	Mobile System Architecture	MANET	Cellular mobile network	Cellular mobile network	Cellular mobile network	Cellular mobile network	Cellular mobile network	Cellular mobile network	Cellular mobile network	Cellular mobile network	Cellular mobile network
	Location of CC Manager	Distributed	Centralized	Distributed	Centralized	Distributed	Distributed	Distributed	Distributed	Distributed	Distributed
	Level of Consistency	Strict	Strict	Strict	Strict	Strict	Strict	Strict	Strict	Relaxed or Strict	Strict
	Transaction Model	Flat	Flat	Flat	Flat	Flat	Flat	Nested	Flat	Nested	Flat
Application Issues	Global Serializability	Yes	No	No	No	No	No	No	No	Yes	No
	Cached/ Replicated data	No	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes
	Real-time Application	No	Yes	No	No	No	No	Yes	No	No	No
MANET Database Characteristics	Mobility	No	No	No	No	No	No	No	No	Yes	Yes
	Low Bandwidth	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes
	Multi-hop Communication	No	No	No	No	No	No	No	No	No	No
	Limited Battery Power	No	Yes	No	No	Yes	No	Yes	Yes	No	Yes
	Frequent Disconnections	No	No	Yes	Yes	No	No	No	No	Yes	No
	Long-lived Transactions	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.3 Clustering Algorithms to Elect Cluster Heads and Form Clusters in MANETs

Several weighted clustering algorithms have been proposed and surveyed in [Yu, 2005] to group mobile nodes into clusters and to elect a node called a cluster head for each cluster. Here we briefly review some of the newly published approaches and other approaches which are already reviewed in [Yu, 2005], but are strongly related to our algorithm MEW. By considering the system parameters that are utilized to calculate the weight of each node, these approaches are categorized as mobility-only-based [Basu, 2001; Kim, 2006], energy-only-based [Sheu, 2006] and combination-based [Basagni, 1999; Chatterjee, 2002; Liu, 2005].

2.3.1 Mobility-only-based

Mobility of nodes triggers re-clustering and makes networks unstable, thus, it becomes the key attribute in the weight computation in the mobility-only-based clustering algorithms.

In MOBIC [Basu, 2001], in order to form stable clusters that have low cluster head change rate, the Relative Mobility (*RM*) metric is introduced and calculated as the logarithm of ratio of Received Signal Strengths (RSS): $10*\log_{10}\frac{RSS_1}{RSS_2}$, where RSS_1 and RSS_2 are read from the RSS indicator when two successive HELLO messages, which are sent by the same neighbor, are received. For each node, the variance of RMs among its neighbors with respect to 0 (not the exact mean) is calculated as the aggregate local mobility metric. If a node has the lowest aggregate local mobility among all its neighbors, it declares itself as a cluster head; otherwise it joins its neighboring cluster

head that has the lowest aggregate local mobility as a cluster member. If a node is a neighbor of two cluster heads, then it becomes a gateway node of those two cluster heads. Unfortunately, it is possible that some elected cluster heads may almost run out of energy, thus, the re-election has to be invoked soon.

In [Kim, 2006], to overcome the negative effects caused by nodes moving fast or moving back and forth, the average connection time (ACT) of each node with its neighbors during a time period is introduced as the major parameter to form and maintain the clusters. Those nodes having the largest ACT values become cluster heads. For each of the remaining nodes, if it has one or more neighboring cluster heads, it joins the one with the largest ACT value as a cluster member; otherwise, it declares itself as a cluster head. However, similar to the cumulative time in WCA [Chatterjee, 2002] or elapsed time [Liu, 2005], this technique has a problem in that it may elect a node as a cluster head because it has the largest ACT even though it almost runs out of energy; such election wastes energy and time.

2.3.2 Energy-only-based

A node with a higher remaining energy level is, of course, a better candidate for the cluster head; so energy is the only system parameter applied to calculate the weight of each node in energy-only-based clustering algorithms.

Because nodes with higher remaining energy have a higher priority to become cluster heads, it is possible that nodes with the least energy being left out and claiming themselves as cluster heads. In Sheu's Stable Cluster Algorithm (SCA) [Sheu, 2006],

Sheu et al. set up an energy level threshold, define nodes whose battery level is below the threshold as bottlenecks, count the number of neighbors that are bottlenecks for each node, and elect nodes with the largest number of bottlenecks as cluster heads. For each of the remaining nodes, if it has one or more neighboring cluster heads, it joins the one with the highest remaining energy as a cluster member; otherwise, it declares itself as a cluster head. By taking the detour in the election, nodes with the least energy are kept from becoming cluster heads, thus, the clusters become more stable. Unfortunately, because the mobility of nodes is not considered in the election, the possibility of re-clustering is still high when elected cluster heads have high mobility.

2.3.3 Combination-based

Each node is assigned with a weight, which is calculated by considering more than one system parameters like node degree, remaining energy, roaming speed, and so on [Yu, 2005]. In DCA (Distributed Clustering Algorithm) [Basagni, 1999], each node is assumed to have a different weight, nodes with the highest weights are elected as cluster heads, and neighbors of elected cluster heads join the cluster as cluster members. However, the calculation of nodes' weights is not discussed [Basu, 2001].

In WCA (Weighted Clustering Algorithm) [Chatterjee, 2002], to determine whether a node v is suited for being a cluster head, the weight of each node (W_v) is calculated by a formula as shown below that consists of four system parameters: sum of distance with all neighbors (D_v), average running speed (M_v), cumulative time of serving as a cluster head (P_v) and degree difference of nodes (Δ_v), where $\Delta_v = |d_v - \delta|$, in which d_v is the number of neighbors and δ is the ideal number of neighbors that a cluster head can

handle. Unfortunately, how to choose δ is not discussed [Yu, 2005].

$$W_v = f_1 * \Delta_v + f_2 * D_v + f_3 * M_v + f_4 * P_v,$$
$$where, f_1, f_2, f_3 \text{ and } f_4 \text{ are weighting factors}$$
$$and \sum_{i=1}^{4} f_i = 1$$

The values of f_1, f_2, f_3 and f_4 are varied based on different applications. Those nodes having the lowest weights are elected as cluster heads. For each of the remaining nodes, if it has one or more neighboring cluster heads, it joins the one with the lowest weight as a cluster member; otherwise, it declares itself as a cluster head. However, how to normalize these parameters is not addressed explicitly. The global positioning system (GPS), the accuracy of which is not ideal for fine computing and the operations of which would drain the limited energy of the node quickly, has to be applied to obtain the coordinates of each node for computing the running speed. The cumulative time of a node already serving as a cluster head cannot accurately reflect the current energy level because a busy node may almost run out of energy even though it may have never been a cluster head.

In Liu's Group Management (GM) [Liu, 2005], the resource (R) richness and elapsed time of a node being a cluster head (leader) are integrated to evaluate a node's suitability for being a cluster head. The resources are CPU load (L), memory (M), battery (B) and bandwidth (BW). The elapsed time (ET) is the time between now and the last time a node is a cluster head. The weight of a node v is calculated by the following formulas, and nodes with the highest weights are elected as cluster heads. For each of the remaining nodes, if it has one or more neighboring cluster heads, it joins the one with the highest weight as a cluster member; otherwise, it declares itself as a cluster head.

$$W_v = f_1 * R_v + f_2 * ET_v, \text{ where } f_1 + f_2 = 1, \text{and}$$
$$f_1 * R_v = f_{11} * L_v + f_{12} * M_v + f_{13} * B_v + f_{14} * BW_v,$$
$$\text{in which, } f_1 = f_{11} + f_{12} + f_{13} + f_{14}$$

However, how to normalize these parameters is not addressed explicitly either. The use of elapsed time in cluster head determination has the same disadvantage as the use of cumulative time in WCA [Chatterjee, 2002]. In addition, re-clustering may be frequently triggered since the mobility of nodes is not considered in the election.

2.3.4 Summaries of reviewed clustering algorithms

Table 2.2 summarizes the characteristics of some weighted clustering algorithms, which have been proposed to elect cluster heads, form clusters and maintain clusters in a MANET.

In those algorithms, when calculating the weight utilized to determine whether a node is eligible to be a cluster head, they either consider only one metric (like mobility or energy of nodes) [Basu, 2001; Kim, 2006; Sheu, 2006] or rely on some metrics collected from extra devices (such as locations of nodes read from Global Positioning Systems) [Chatterjee, 2002]. This often leads to a higher possibility of re-clustering and, consequently, quality of service cannot be provided.

Table 2.2 Summary of the reviewed clustering algorithms

Techniques/Issues	System Parameters Used in the Weight Calculation	Relying on Any Metrics Collected from Extra Devices
ACT [Kim, 2006]	Mobility-only: average connection time	No
DCA [Basagni, 1999]	Each node is assumed to have a different weight, but how to compute it is unspecified.	Unspecified
GM [Liu, 2005]	Combination: elapsed time of being a cluster head and system resources (CPU load, memory, energy and bandwidth)	No
MOBIC [Basu, 2001]	Mobility-only: aggregate local mobility	No
SCA [Sheu, 2006]	Energy-only: number of neighbors with low energy level	No
WCA [Chatterjee, 2002]	Combination: sum of distance with all neighbors, average running speed, cumulative time of serving as a cluster head and degree difference of nodes	Yes. Global positioning system (GPS)

2.4 Conclusions

In this chapter, we discussed the general issues and application-dependent issues that need to be addressed in the design of CC algorithms for mobile databases, and then we reviewed the CC techniques proposed for cellular mobile databases and MANET databases according to these issues. The review shows that SESAMO is the only algorithm proposed for MANET databases, but it does not take energy efficiency into account. All other techniques designed for cellular mobile databases are not suitable for MANET databases either because in those algorithms, servers are static and broadcast is heavily used to disseminate data from servers to clients.

We also reviewed clustering algorithms developed to elect cluster heads and form

clusters in MANETs. In those clustering algorithms, when calculating nodes' weights utilized to elect cluster heads, they either consider only one metric like mobility or energy of nodes, or rely on some metrics collected from extra devices such as GPS. This often leads to a higher possibility of cluster head change and re-affiliation. New CC techniques and node clustering algorithms need to be developed to address MANET characteristics including energy limitation, mobility, and frequent disconnection.

CHAPTER 3

MANET DATABASE ARCHITECTURE USED IN THE PROPOSED ALGORITHM

In this chapter, we introduce our clustered MANET architecture which is built by applying our robust weighted clustering algorithm called MEW (Mobility, Energy, and Workload) [Xing, 2010] to group nodes into groups (or clusters). MEW takes mobile nodes' mobility, energy and workload into consideration when clustering mobile nodes in a MANET. In this architecture as shown in Figure 3.1, mobile nodes are divided into clusters, each of which has one cluster head working as the coordinating server responsible for the transaction processing of the mobile nodes, called cluster members, within the cluster. Cluster heads can communicate with each other through some mobile nodes that work as gateways. Similarly, mobile nodes within the same cluster as well as from different clusters can also communicate with each other, but they have to go through their cluster heads to get the destination addresses first. In order to guarantee global serializability and reduce communication overhead, among cluster heads the one with the highest remaining energy is further elected as the primary cluster head to maintain the information of committed global transactions and validate transactions globally.

We choose this architecture for three reasons. First, users are logically grouped in many MANET applications in the literature, such as disaster response and recovery systems [Catarci, 2008; Lu, 2007] and military operations [Viswacheda, 2007]. Second, because every node is mobile in a MANET, the network topology may change rapidly and unpredictably over time. According to [Chatterjee, 2002], clustered architectures are

proper to keep the network topology stable as long as possible, so that the performance of routing and resource relocation protocols is not compromised. Third, in order to accommodate our optimistic CC algorithm SODA to guarantee the global serializability, the information of committed global transactions is maintained by the cluster head with the highest remaining energy.

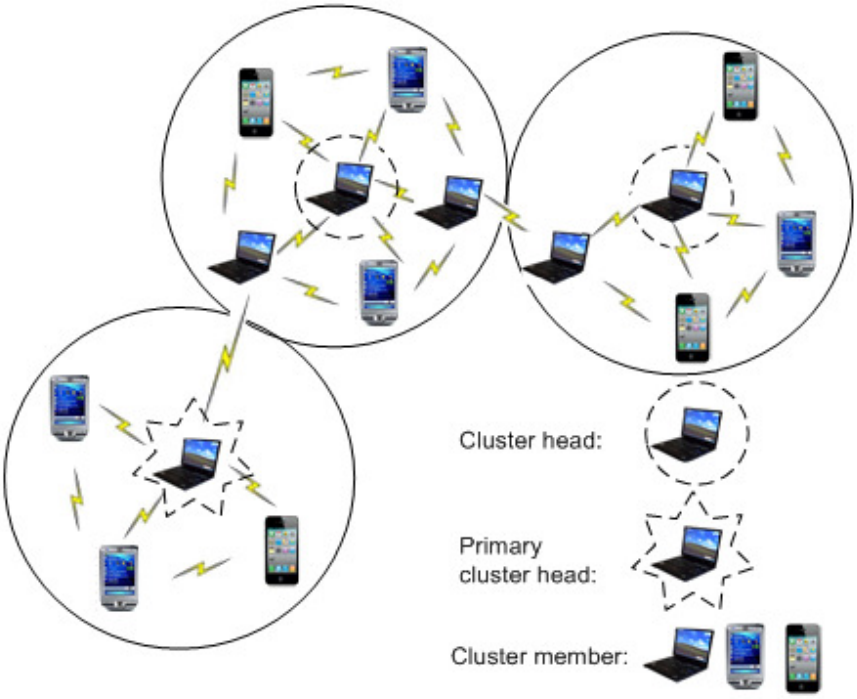

Figure 3.1 Architecture of a clustered MANET database

Figure 3.1 shows an example of a clustered MANET database architecture with three clusters, each of which is represented by a large solid circle with mobile clients and servers shown as PDA/iphone and laptop icons, respectively. The arrows between the devices show the communication between them. In the rest of this chapter, we describe

the functionality of mobile nodes (Section 3.1), the MEW algorithm (Section 3.2), the cluster formation (Section 3.3), the cluster maintenance (Section 3.4), and the analysis of clustering overhead (Section 3.5).

3.1 Mobile Node Functionality

Depending on the communication strength, computing power and storage size, mobile nodes are classified into clients and servers [Pabmanabhan, 2006]. On clients, only the query processing modules that allow them to submit transactions and receive results are installed, while on servers, the complete database management systems are installed to provide transaction processing services. Servers are further classified into coordinating servers or participating servers. Coordinating servers are the ones which receive global transactions from clients, divide them into sub-transactions, forward these sub-transactions to appropriate participating servers, and maintain the ACID (Atomicity, Consistency, Isolation, and Durability) properties of global transactions. Participating servers are the ones that process sub-transactions transmitted from coordinating servers, and preserve their ACID properties.

The entire data in the database are partitioned into fragments and distributed to different servers, and there is no caching or replication technique involved for simplicity. Transactions are based on the simple flat model, which contains a set of read, write, insert, and delete operations. Any subset of operations of a global transaction that access the same server is submitted and executed as a single sub-transaction [Dirckze, 2000].

With respect to the clustered MANET architecture shown in Figure 3.1, a mobile node is either a cluster head when it is a coordinating server or is a cluster member when

it is either a participating server or a client. In order to guarantee global serializability and reduce communication overhead, among cluster heads the one with the highest remaining energy is further elected as the primary cluster head, which maintains the information of committed global transactions and validates transactions globally.

3.2 The Basis of Our Clustering Algorithm - MEW

Being inspired by MOBIC [Basu, 2001] and WCA [Chatterjee, 2002] and considering a new system parameter, called "Energy Decreasing Rate (*EDR*)", we propose a weighted clustering algorithm, called MEW (Mobility, Energy, and Workload), to build a stable backbone in MANETs. Although our proposed clustering algorithm is also combination-based (based on a combination of parameters), it can become mobility-only-based if we tune the weighting factors accordingly. In other words, our approach can build a more stable backbone for MANETs by forming clusters.

To capture the mobility of nodes, we do not consider their absolute roaming speed, which is actually applied in WCA [Chatterjee, 2002]. This is because it is easy to calculate the speed's quantity but it is hard to predict the direction of movement. Without the direction, the speed's quantity alone is not appropriate to justify whether or not a node is a good candidate for cluster head. For instance, given two nodes that have small speeds, but both move in the opposite directions, as time goes, they will be out of each other's transmission range and get disconnected from each other. Also the utilization of GPS is opted out due to the following reasons:

- When a GPS is utilized, every node must be equipped with one GPS, which

incurs high hardware costs. In addition, these GPSs consume the limited energy of nodes.

- GPS does not work indoor because buildings shield the satellite signals [Basu, 2001; Bruning, 2007].
- The accuracy of a typical civilian GPS is in the range of 6-12 meters [Berrabah, 2009]; thus, the returned results could be the same when two GPSs are located within 10 meters.
- If data cannot be read from the GPS and no substitution of the missing data can be found, then the whole system has to wait or fail.

Instead, in our algorithm, two mobility metrics, Relative Mobility (*RM*) [Basu, 2001] and Mobility Prediction (*MP*), are introduced to monitor the mobility of nodes and applied to determine whether a node is suitable to be a cluster head as follows:

- For each node j ($1 \leq j \leq N$ for N nodes in the network), after receiving two successive HELLO messages from every 1- hop neighbor i ($1 \leq i \leq n$ if there are n neighbors), the RM_{ij} is calculated by the formula (3.1). RSS_{ij1} and RSS_{ij2} are the received signal strength (RSS) that are read from the RSS indicator when the first and second HELLO messages from the same neighbor are received, respectively. Based on the value of RM_{ij}, we can say that if RM_{ij} is equal to 1, then the node j and its neighbor i either do not move at all or move with the same speed in the same direction; if RM_{ij} is less than 1, then they move close to each other; otherwise, they move away from each other.

$$RM_{ij} = \frac{RSS_{ij1}}{RSS_{ij2}} \tag{3.1}$$

- For each node j, to take into account the mobility of all n neighbors, MP_j is calculated as the standard deviation of $RM_{1j}, RM_{2j}, \ldots, RM_{nj}$ shown in the formula (3.2). However, for the stability of elected clusters, we prefer RM_{ij} to be equal to or less than 1 because we want cluster heads not to move away from their members. Thus, in the MP_j calculation, the mean of RM_{ij} ($1 \le i \le n$) is 1 instead of the actual mean. A node j with a lower MP_j means that it stays closer to its neighbors, thus, it is a better candidate for the cluster head among its neighbors.

$$MP_j = \sqrt{\frac{\sum_{i=1}^{n} (RM_{ij} - \overline{RM_{ij}})^2}{n}}, \text{ where } \overline{RM_{ij}} = 1 \tag{3.2}$$

When dealing with the limited energy, we consider not only the Remaining Energy (*RE*) of each node but also its Energy Decrease Rate (*EDR*) as the workload because nodes with heavier workload consume more energy, so that we can balance the energy usage and prevent cluster heads from running out of energy quickly. In other words, for each node j, EDR_j is considered because RE_j represents only the current state of energy level and the energy will run out soon if this node usually has a heavy workload (for instance, it provides service as a server and relays packets for many neighbors). The EDR_j at time interval [t_1, t_2] is calculated by using the formula (3.3), where RE_{j1} and RE_{j2} are the remaining energy at time t_1 and t_2, respectively.

$$EDR_j = \frac{RE_{j_1} - RE_{j_2}}{t_2 - t_1} \tag{3.3}$$

A node with a lower *EDR* indicates that it was not busy at least during the

interval [t_1, t_2]. However, when a node had a busy work history, it most likely would be busy in the future as well. Since the larger the time interval is, the more accurate the *EDR* is in indicating a node's workload history, during the initial election, each node saves a copy of its initial remaining energy and initial time as RE_{j1} and t_1, such that a more accurate *EDR* can be calculated in the future cluster head re-election.

Based on the above analysis about energy, mobility and workload, it is obvious that a node *j* is the best candidate for a cluster head among all its neighbors if its RE_j is the highest, its MP_j is the lowest and its EDR_j is the lowest. In other words, a node with the highest weight is the best candidate for a cluster head when we combine these three metrics together as the weight, which is calculated in formula (3.4). Since these metrics have different units, we apply the inversed exponential function to normalize MP_j and EDR_j and bound their values between 0 and 1. RE_j is left out because it is the remaining energy level in percentage and its value is already between 0 and 1.

$$W_j = f_1 * e^{-MP_j} + f_2 * RE_j + f_3 * e^{-EDR_j} \qquad (3.4)$$

In formula (3.4), $RE_j = RE_{j2}$, the weighting factors f_1, f_2 and f_3 are set according to the different scenarios in the applications, and $f_1 + f_2 + f_3 = 1$. When we let $f_2 = f_3 = 0$, that is, we take away the effect of energy and workload, our algorithm turns into a mobility-only-based approach just like MOBIC [Basu, 2001].

3.3 Cluster Formation

Cluster formation involves the following four steps, where the messages used in cluster formation and maintenance are summarized in Table 3.1.

Step 1: Each node j periodically broadcasts (this broadcast interval is predefined [Basu, 2001]) a HELLO message with the same transmission power. In the mean time, the remaining energy level RE_{j1} is recorded at the initial election time t_1; for each HELLO message received from neighbor i, RSS_{ij} is recorded. If only one HELLO message from a neighbor is received after j broadcasts three successive HELLO messages, then this neighbor is excluded from the weight calculation [Basu, 2001].

Table 3.1 Messages used in MEW

Message	***Description***
HELLO(my_ID, my_W, CH_ID, my_RE, other_CH)	To notify neighbors about my ID, my weight, my cluster head's ID, my remaining energy and any other neighboring cluster heads.
WEIGHT(my_ID, my_W)	To notify neighbors about the value of my weight.
CLUSTERHEAD(my_ID, CH_ID)	To notify neighbors about my role: I am a cluster head, that is, my ID is the same as my cluster head's ID.
JOIN(my_ID, CH_ID)	To notify neighbors that I am going to join the cluster whose cluster head's ID is CH_ID. If a cluster head broadcasts a JOIN message, then it informs its members about its resignation and joins a cluster at the same time.

Step 2: Immediately after each node j receives two successive HELLO messages from all the neighbors, it records the remaining energy level RE_{j2} and the time t_2, and then node j calculates the values of RM_j, MP_j, EDR_j and W_j using the formulas 3.1, 3.2, 3.3 and 3.4 defined above, respectively. When the weight is re-calculated, the saved remaining energy RE_{j1} at time t_1 is applied in the calculation of EDR_j.

Step 3: Each node j broadcasts the value of W_j to all its neighbors in a WEIGHT message, and waits for their WEIGHT messages.

Step 4: Upon receiving the weights from all neighbors, the nodes with the highest

weight declare themselves as cluster heads among their neighbors by broadcasting a CLUSTERHEAD message, but, no two cluster heads should be neighbors. All neighbors of elected cluster heads just join them as their members by broadcasting JOIN messages. If two nodes have the same weight, then the node with a smaller ID becomes the cluster head [Basu, 2001]. If it happens that a node is neighbor of two or more cluster heads, then it joins the cluster head with the highest weight and works as a gateway for these cluster heads.

Note that because clients have less communication strength, less computing power and smaller storage size than servers, clients cannot be elected as cluster heads and cannot work as coordinating servers.

3.4 Cluster Maintenance

Because every node can roam and has limited energy in a MANET, the links between members and cluster heads can be broken, and the links between two cluster heads can be generated [Xue, 2006]. Consequently, clusters need be re-clustered. In other words, leaving clusters, joining clusters, merging clusters, and re-electing cluster heads are normal re-clustering operations in a clustered MANET. However, these operations should be performed only on demand to reduce the overhead of computation and communication, and to provide consistent quality of service.

In order to detect the link breaks and new link establishments, each node periodically broadcasts a HELLO message, which contains the ID and weight of itself, its cluster head's ID, its remaining energy and any other neighbor which is a cluster head (a

Boolean variable). Being a cluster head, it has to periodically monitor its remaining energy level so that it will resign when the remaining energy drops below a predefined threshold. Also each node keeps recording the values of *RSS* from the last two HELLO messages and re-calculating its weight in case of future re-elections.

Relying on these two periodical operations, cluster maintenance can be done by the following recovery:

- *From the link break between a member and its cluster head*: after three successive broadcast intervals (*BI*) [Wang, 2007a], if no HELLO message is received from a member, the cluster head will just remove this member from its neighbor and member lists. On the other hand, if a member does not receive a HELLO message from the cluster head after three successive *BI*s, it removes the cluster head from its neighbor list, and joins another cluster head with the highest weight if any. If no other cluster head is available from its neighbor list, this member declares itself as a cluster head.

- *From the link establishment when two cluster heads become neighbors*: if a cluster head has become a neighbor of another cluster head for a predefined Cluster Contention Interval (*CCI*) [Basu, 2001], then the cluster head with the smaller weight resigns and joins the cluster head with the larger weight. The members of the resigned cluster head cannot join the new cluster head because they are not neighbors. They have to join other cluster heads with the highest weight or declare themselves as cluster heads instead.

- *From the link break when a cluster head resigns*: if the current remaining energy of a cluster head becomes less than the Low Energy Threshold (*LET*), and if there

exists a member whose energy is higher than *LPT*, and this member has no other neighbor that is a cluster head, then the cluster head resigns and triggers a cluster head re-election; but this re-election is limited to inside the old cluster instead of the whole network, that is, the resigned cluster head goes through each member's profile, which is periodically updated after receiving a HELLO message, and finds a replacement that has the highest weight. After the new cluster head is elected, the resigned cluster head joins its cluster. If a member cannot join the new cluster head because they are not neighbors, it has to join other cluster heads or declare itself as a cluster head.

3.5 Analysis of Clustering Overhead

In this section, we analyze MEW with respect to the message overhead per time step per node and time complexity per network topology change. These terms are defined below. The approach used is inspired by the theoretical analysis in [ER, 2005].

The price of clustering is that extra time is consumed and additional messages are incurred to form and maintain clusters. The consequence of these additional messages is called message overhead (the more messages are transmitted, the more traffic is in the network and the more energy of nodes is consumed). Since bandwidth and energy of each node are limited in MANETs, message overhead is an important metric for evaluating the performance of a clustering algorithm. We analyze the message overhead by analyzing the overhead due to the HELLO protocol and the overheads due to cluster formation and maintenance. In the mean time, the time complexity per network topology change is also computed.

To simplify the analysis, the continuous runtime is divided into discrete time steps, which are the duration between the time when a message is sent and the time when the message is received and processed by a receiver [Bettstetter, 2002]. The random waypoint mobility model with zero pause time is assumed. The following definitions are used in the analysis [ER, 2005]:

- N: the number of nodes in the MANET;
- m: the average number of cluster members in a cluster; $m = \Theta(1)$ because all clusters should have a maximum size constrain to avoid overburdening cluster heads [Banerjee, 2001].
- f_{hello}: the number of HELLO messages broadcast by a node per time step; $f_{hello} = \Theta(1)$ because f_{hello} is proportional to average node speed s and inversely proportional to the transmission radius R, and both s and R are less than or equal to some constants [Sucec, 2004].
- f_{link}: the average frequency of network topology changes occurred per time step; $f_{link} = \Theta(N)$ [Sucec, 2004].
- T: the number of time steps taken by the algorithm after a network topology change to re-establish a valid cluster structure (or called re-clustering);
- M: the number of messages (or called packets) exchanged between nodes after a network topology change to re-establish a valid cluster structure;
- L: the total evaluation time in terms of time steps.

In terms of the average number of messages transmitted by MEW per time step per node, and the number of time steps needed to re-establish a valid cluster structure

after a topology change, the following claims are made:

Claim 1: the *message overhead of MEW is O(1) packet transmissions per time step per node.*

Claim 2: *the time complexity of MEW is $T \leq 2$ per network topology change.*

Both claims are proved in the following subsections.

3.5.1 Hello protocol overhead

In order to discover its neighborhood and compute its weight, each node broadcasts HELLO messages periodically. Thus, the HELLO protocol introduces an overhead of $f_{hello}*N$ packets per time step for all nodes.

3.5.2 Cluster formation overhead

Immediately after each node calculates its weight, it broadcasts a WEIGHT message in one time step. After receiving WEIGHT messages from all its neighbours, each node either becomes a cluster head by broadcasting a CLUSTERHEAD message or joins some cluster by broadcasting a JOIN message in one time step. Thus, for all nodes, they broadcast $2N$ messages in 2 time steps, that is, cluster formation overhead is N messages per time step.

3.5.3 Cluster maintenance overhead

From the discussion of cluster maintenance, it is obvious that every network

topology change is detected by relying on the periodical HELLO messages, and each cluster head resignation is verified by periodically checking the remaining energy level. Once a network topology change or a cluster head resignation occurs, relating nodes have to take respective actions to re-establish a valid cluster structure. Because of these actions, cluster formation and maintenance overheads incur, which are investigated in the following four subsections.

3.5.3.1 Link break between a member and its cluster head

Since the cluster structure is still valid when a link break occurs between nodes from different clusters or nodes that are members from the same cluster, there is no action. Only a link break between a member and its cluster head triggers the re-clustering.

The cluster head removes this member from its neighbor and member lists, so no message is necessarily transmitted. On the other hand, this member removes this cluster head from its neighbor list as well, and joins another cluster head with the highest weight if any. This case is done by broadcasting a JOIN message in one time step. If no other cluster head is available from its neighbor list, this member declares itself as a cluster head, and broadcasts a CLUSTERHEAD message in one time step. Thus, we have: $T = 1$ and $M = 1$ for one of this kind of link breaks.

3.5.3.2 Link establishment because two cluster heads become neighbours

When a cluster head becomes a neighbor of another cluster head for a predefined

duration of *CCI* interval, the cluster head with the smaller weight resigns and joins the cluster head with the larger weight. The resigned cluster head has to broadcast a JOIN message to inform all its members in one time step. After receiving the JOIN message from their cluster head, each member of this resigned cluster head has to do re-clustering, which is the same as the case in subsection "Link Break between a Member and Its Cluster Head". To summarize, $T = 2$ and $M = m + 1$ for one of this kind of link establishments.

3.5.3.3 Link break because a cluster head resigns

When the current remaining energy of a cluster head is less than the threshold *LPT*, and there exists one of its members that can be a new cluster head, this cluster head resigns and triggers a cluster head re-election.

Once a new cluster head is elected, the resigned cluster head broadcasts a JOIN message to inform all its members in one time step. After receiving the JOIN message from its cluster head, each member of this resigned cluster head has to do re-clustering, which is also the same as the case in subsection "Link Break between a Member and Its Cluster Head". In short, $T = 2$ and $M = m + 1$ for one of this kind of link breaks.

3.5.3.4 Total cluster maintenance overhead

Since $M = 1$ in the case of "link break between a member and its cluster head" and $M = (m+1)$ in the case of "link establishment because two cluster heads become neighbors", the total number of messages transmitted per network topology change due

to link state changes is (m+2). As the average network topology changes occurred per time step is f_{link}, there are totally f_{link}*(m+2) messages per time step due to link state changes.

Since a node cannot become a cluster head any more once it resigns due to its lower remaining energy, there are at most N cluster head resignations in an evaluation. An evaluation period consists of L time steps, thus, the average number of cluster head resignations per time step is N/L. Therefore, the total number of messages is $N(m+1)/L$ per time step due to the cluster head resignation.

In summary, the total cluster maintenance overhead is f_{link}*(m+2) + $N(m+1)/L$ messages per time step.

3.5.4 Total message overhead

To summarize, the message overhead of MEW (O_{MEW}) is the sum of the overhead due to the HELLO protocol, the overhead due to cluster formation and the overhead due to cluster maintenance, that is,

$$O_{MEW} = f_{hello}*N + N + f_{link}*(m+2) \; + \; N(m+1)/L$$

Since $f_{hello} = \Theta(1)$, $f_{link} = \Theta(N)$, $m = \Theta(1)$, L is an integer and $L > 1$, given some constants c_1, c_2 and c_3, we have: $f_{hello} \leq c_1$, $f_{link} \leq c_2*N$, $m \leq c_3$ and $(1/L) < 1$. Therefore, O_{MEW} can be expressed as follows:

$$
\begin{aligned}
&O_{MEW} = f_{hello}*N + N + f_{link}*(m+2) \ + \ N(m+1)/L \\
&\Rightarrow O_{MEW} \le f_{hello}*N + N + f_{link}*(m+2) \ + \ N(m+1) \\
&\Rightarrow O_{MEW} \le c_1 * N + N + c_2 * N * (c_3 + 2) + N * (c_3 + 1) \\
&\Rightarrow O_{MEW} \le (2 + c_1 + 2c_2 + c_3 + c_2 c_3) * N \\
&\Rightarrow O_{MEW} = O(N)
\end{aligned}
$$

After dividing O(N) by the number of nodes N, the message overhead of MEW is O(1) per time step per node and Claim 1 is proved.

T = 1 for a link break between a member and its cluster head, and T = 2 for both a link establishment and a link break due to the resignation of some cluster head, therefore, the convergence time is at most 2 time steps per topology change as per Claim 2.

3.6 Conclusions

In this chapter, our robust weighted clustering algorithm, called MEW (Mobility, Energy, and Workload), was introduced to form and maintain stable clusters in MANETs. Unlike the existing node clustering algorithm, MOBIC, that considers only nodes' mobility during electing cluster heads and forming clusters, MEW takes not only nodes' mobility but also nodes' energy and workload into account when clustering nodes in a MANET. We also analyzed MEW with respect to the message overhead per time step per node and time complexity per network topology change. The message overhead of MEW is O(1) packet transmissions per time step per node and the time complexity of MEW is T $\le$ 2 per network topology change.

CHAPTER 4

THE PROPOSED CONCURRENCY CONTROL ALGORITHM: SEQUENTIAL ORDER WITH DYNAMIC ADJUSTMENT

In this chapter, we describe our proposed CC technique, called Sequential Order with Dynamic Adjustment (SODA). We first provide some preliminaries to help explain our approach. Second, we describe how SODA works without the clustered MANET database involved; in other words, how SODA works in a centralized MANET database. Third, we provide the complexity analysis and correctness proof of SODA. Finally, we discuss how SODA works in a clustered MANET database presented in Chapter 3.

4.1 Preliminaries

Two operations (or transactions) are called **conflict operations (or transactions)** if they access the same data item and at least one of them is a write operation (or transaction) [Bernstein, 1987].

Let $S = \{T_1, \ldots, T_n\}$ be a set of transactions. A **History** (also called Schedule or Log) over S is an execution of S where the operations of the transactions are interleaved, but the order of operations within the same transaction is preserved [Bernstein, 1987].

Two histories are **conflict equivalent** if they involve the same set of transactions, and every pair of conflict operations is ordered in the same way in both histories [Bernstein, 1987].

A history is **serial** if each transaction is executed from the beginning to the end before the next one can start [Bernstein, 1987].

A history is **serializable** if it is conflict equivalent to a serial history [Bernstein, 1987].

The **serialization graph** (SG) of a history is a directed graph where the nodes are the transactions executed in the history. In the SG, there is an edge $T_i \rightarrow T_j$ ($i \neq j$) if and only if at least one of T_i's operations precedes and conflicts with one of T_j's operations in the execution history [Bernstein, 1987].

The **Serializability Theorem** [Bernstein, 1987]: A history H is serializable if and only if there is no cycle in the serialization graph of H.

Timestamp Ordering (TO) [Bernstein, 1987]**:** a unique timestamp is assigned to each transaction, and conflict operations between every two transactions are executed in their timestamp order. The timestamp may be assigned at the beginning, middle or end of the execution of a transaction.

Backward validation: a validating transaction is validated against only committed transactions, and the currently active transactions are not involved.

Given a transaction T, **Optimistic Concurrency Control (OCC)** [Kung, 1981] has three phases to go through:

- *Read and Compute Phase* (Phase 1): T reads the values of a set of data items (called read set, and denoted by RS(T)) and saves them into local variables. When T reads a data item d, a timestamp is assigned (denoted by TS(d)). T also computes the values for a set of data items (called write set, and denoted by WS(T)) and saves them in local variables.
- *Validation Phase* (Phase 2): the read set and write set of T are validated against a

set of committed transactions. If T passes the validation test, then a timestamp is assigned to T (denoted by TS(T)), and used as the commit time of T and the timestamp of the write set (denoted by WS_TS(T)). WS_TS(T) is $+\infty$ if T is a validating transaction.

- *Commit and Write Phase* (Phase 3): if T succeeds in the Validation Phase, then it can write the values of the write set into the database and commit; otherwise, T has to be aborted.

Definition 4.1: Given a validating (or committed) transaction T_1, a committed transaction T_2 and a commonly accessed data item d, T_1 **must-be-serialized-before** T_2 if any one of the following conditions is satisfied (Note: $T_i \rightarrow TS(d)$ stands for T_i getting the read timestamp of data d):

- *Read-Write (RW) conflict*: $RS(T_1) \cap WS(T_2) \neq \varnothing$ and $T_1 \rightarrow TS(d) < WS\_TS(T_2)$.
- *Write-Write (WW) conflict*: $WS(T_1) \cap WS(T_2) \neq \varnothing$ and $WS\_TS(T_1) < WS\_TS(T_2)$.
- *Write-Read (WR) conflict*: $WS(T_1) \cap RS(T_2) \neq \varnothing$ and $WS\_TS(T_1) < T_2 \rightarrow TS(d)$.

Definition 4.2: Given a validating (or committed) transaction T_1, a committed transaction T_2 and a commonly accessed data item d, T_1 **must-be-serialized-after** T_2 if any one of the following conditions is satisfied:

- *Read-Write (RW) conflict*: $RS(T_1) \cap WS(T_2) \neq \varnothing$ and $T_1 \rightarrow TS(d) > WS\_TS(T_2)$.
- *Write-Write (WW) conflict*: $WS(T_1) \cap WS(T_2) \neq \varnothing$ and $WS\_TS(T_1) > WS\_TS(T_2)$.
- *Write-Read (WR) conflict*: $WS(T_1) \cap RS(T_2) \neq \varnothing$ and $WS\_TS(T_1) > T_2 \rightarrow TS(d)$.

4.2 Proposed Concurrency Control Algorithm - SODA

Inspired by the dynamic adjustment technique proposed in MTC-SG/SQ [Hwang, 2000], and based on the combination of Timestamp ordering (TO), OCC, and backward validation, we propose an optimistic CC algorithm, called Sequential Order with Dynamic Adjustment (SODA).

4.2.1 Algorithm description and examples

Assume that T_i's (i = 1, …, n) are committed transactions, and T is a validating/committing transaction. If we simply let the validation/commit order be the serialization order like OCC [Kung, 1981], and if there is a RW conflict between T and T_i, i.e. $RS(T) \cap WS(T_i) \neq \varnothing$ and $T \rightarrow TS(d) < WS\_TS(T_i)$ for some data item d, then T is aborted because two orders are different. Such aborts should be avoided if possible. The following sections describe how SODA avoids such unnecessary aborts.

In SODA, a dynamic order instead of the validation order among committed transactions is used, that is, a Sequential Order (SO) of committed transactions is maintained as $\{T_1, T_2, \ldots, T_i, \ldots, T_n\}$ (also called a history list, which is ordered from left to right) and can be dynamically adjusted. The dynamic adjustment consists of simple and complex cases. In the simple case, the validating transaction T can be directly inserted into the maintained sequential order without adjustment, and the final sequential order will be: $\{T_1, T_2, \ldots,$ ***low, … T, up***$, \ldots, T_n\}$, such that T must-be-serialized-after *low* but before *up*. On the other hand, in the complex case, the sequential order must be

adjusted before the insertion of T.

The simple case: Our goal is to find the transactions *low* and *up*, such that SO(*low*) < SO(T) < SO(*up*). If we do, then T passes the validation test (lines 2 to 19 in Figure 4.3). $SO(T_i)$ is the function to get the sequential order number of T_i in the history list. For instance, $SO(T_2) = 2$ and $SO(T_i) = i$ if the sequential order is $\{T_1, T_2, \ldots, T_i, ..., T_n\}$.

Without loss of generality, we should find two transactions ***low*** and ***up*** where,

$$SO(low)=\max\{SO(T_i) | T \text{ must-be-serialized-after } T_i,\ 1 \le i \le n\},$$

$$SO(up)=\min\{SO(T_i) | T \text{ must-be-serialized-before } T_i,\ 1 \le i \le n\}.$$

If *low* (*up*) is not found, then we can conclude that T is not serialized after (before) any other transactions, and we say that SO(*low)* = 0 (SO(*up)*= n + 1) (line 1 in Figure 4.3). However, if SO(*low)* = SO(*up)*, then it is impossible for T to be serialized before and after T_i at the same time, thus, T is aborted. If SO(*low)* > SO(*up)*, T should be aborted because it cannot be inserted anywhere in the list. However, T passes the validation test if the serialization graph testing is applied instead. Thus, this kind of aborts should be avoided too if possible. The details are given in the complex case below.

To illustrate how SODA works for the easy case, let's see an example: T can be directly inserted in the maintained sequential order.

Example 1 (for the simple case): Let $\{T_1, T_2, T_3\}$ be the sequential order of committed transactions, and T be a validating transaction at a server. The read sets, write sets and the timestamps are shown in Table 4.1.

Since $WS(T_1) \cap WS(T) \neq \varnothing$ and WS_TS(T) > $WS\_TS(T_1)$, T must-be-serialized-

after T_1 and *low* = T_1. Since $WS(T_2) \cap RS(T) \neq \varnothing$ and $WS\_TS(T_2) > T \rightarrow TS(x)$, T must-be-serialized-before T_2 and *up* = T_2. Since SO(*low*) < SO(*up*), this is the simple case and T passes the validation test. T is inserted immediately before T_2, and the final sequential order is $\{T_1, T, T_2, T_3\}$ as shown in Figure 4.1, where the arrow (→) in the graph indicates the serialization order between two transactions such as $T_1 \rightarrow T_2$ means that T_1 must-be-serialized-before T_2.

Table 4.1 Transaction information used in Example 1

	T_1	**T_2**	**T_3**	**T**
Read Set (RS)	{x}	{y}	{x, y}	{x}
Write Set (WS)	{z}	{x}	∅	{z}
Read Timestamp of Data d (TS(d))	5	15	25, 30	18
Timestamp of Write Set (WS_TS)	10	20		+∞

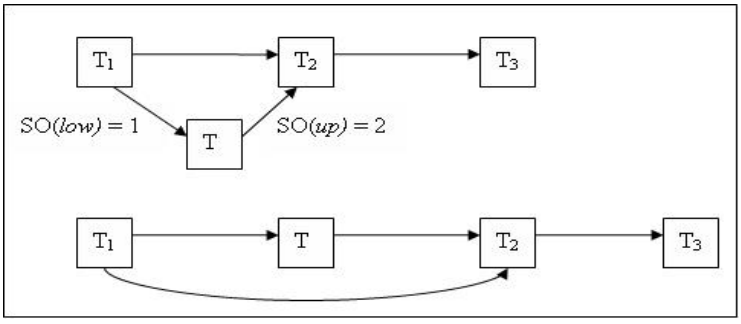

Figure 4.1 Validating transaction T in Example 1

To better understand why a validating transaction should be aborted due to SO(*low)* > SO(*up)*, let us explain by example.

Example 2: Let $\{T_1, T_2\}$ be the sequential order of committed transactions, and T

be a validating transaction at a server. The read sets, write sets and the timestamps are shown in Table 4.2.

Table 4.2 Transaction information used in Example 2

Transactions	**T_1**	**T_2**	**T**
Read set (RS)	{x}	{y}	{x}
Write set (WS)	{x}	∅	{y}
TS (d)	5	20	8
Timestamp of WS	10		+∞

Since $RS(T_2) \cap WS(T) \neq \varnothing$ and $T_2 \rightarrow TS(y) < WS\_TS(T)$, T must-be-serialized-after T_2. Since $WS(T_1) \cap RS(T) \neq \varnothing$ and $WS\_TS(T_1) > T \rightarrow TS(x)$, T must-be-serialized-before T_1. Thus, *low* = T_2, *up* = T_1, and the final sequential order should be $\{T_2, T, T_1\}$ or $SO(T_2) < SO(T) < SO(T_1)$; but this is impossible because the given sequential order is $SO(T_1) < SO(T_2)$. So, T has to be aborted because it cannot be inserted in the given sequential order. However, if the serialization graph testing is applied instead, T passes the validation because there is no cycle in the serialization graph as shown in Figure 4.2. To further reduce the abort rate, we need resolve this complex case too.

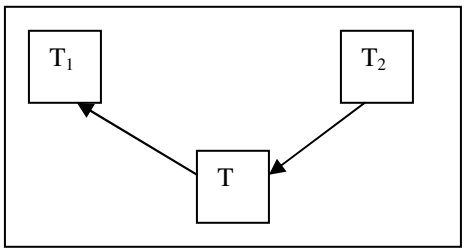

Figure 4.2 The serialization graph in Example 2

```
Boolean SODA(T, History) {
1:   low_index = 0; up_index = History→length() + 1;
2:   counter = 1; // Find transaction up
3:   for (Ti = History→begin(); Ti != History→end(); Ti++) {
4:        if (must-be-serialized-before(T, Ti)) {
5:             up = Ti; up_index = counter;
6:             break;
7:        }
8:        counter++;
9:   }
10:  counter = History→length(); // Find transaction low
11:  for (Ti =--(History→end());Ti >=History→begin();Ti--) {
12:       if ( must-be-serialized-after(T, Ti)) {
13:            low = Ti; low_index = counter;
14:            break;
15:       }
16:       counter--;
17:  }
18:  if (low_index < up_index) // The simple case
19:       return true;
20:  range = History→subset(up, low); // The complex case
21:  T_SB→push_back(T);
22:  for (Ti = range→begin(); Ti != range→end(); Ti++) {
23:      for (Tj = T_SB→begin(); Tj != T_SB→end(); Tj++) {
24:          if ( must-be-serialized-before(Tj, Ti)) {
25:               if ( must-be-serialized-after(T, Ti)
26:                    return false; // A cycle is detected
27:               T_SB→push_back(Ti);
28:               break;
29:          }
30:      }
31:  }
32:  return true; // Got here. T passes the validation test
}
```

Figure 4.3 SODA - validation and preparation

The complex case: We have $SO(T_i) < SO(T_j)$ from the maintained sequential order $\{T_1, T_2, T_3, \ldots, T_i, \ldots, T_j, \ldots, T_n\}$, but we conclude that $SO(T_i) > SO(T) > SO(T_j)$ after finding *low* and *up* where *low* = T_j and *up* = T_i. Since T is just stuck between T_i and T_j, if we can find all transactions between T_i and T_j that T must-be-serialized-before

directly and indirectly (called T_SB), and if there are no transactions in T_SB that T must-be-serialized-after, then T passes the validation test; otherwise, a cycle is detected and T has to be aborted (lines 20 to 32 in Figure 4.3).

After T passes the validation test, the sequential order has to be updated to reflect the changes. In the simple case, T is directly inserted in the position just before *up* (lines 1 to 2 in Figure 4.4). In the complex case, by looping through all transactions between *up* and *low*, all the transactions in T_SB constructed in the first part of SODA are removed first (lines 3 to 11 in Figure 4.4). To construct SO(*low*) < SO(T) < SO(*up*), T and all transactions in T_SB are inserted in the position immediately after *low* (lines 12 to 18 in Figure 4.4).

```
void update_SO(low, up, T, History,  T_SB) {
1:   if (SO(low) < SO(up)) // The simple case
2:         History→insert(up, T);
3:   range = History→subset(up, low); // The complex case
4:   T_m = T_SB→begin();
5:   for (T_i = range→begin(); T_i != range→end(); T_i++) {
6:         if (T_i == T_m) {
7:               History→erase(T_i); T_m++;
8:               if (T_m == T_SB→end())
9:                     break;
10:        }
11:  }
12:  low++; // Insert T immediately after low
13:  History→insert(low, T);
14:  // Insert all the transactions in T_SB
15:  while (!T_SB→empty()) {
16:        History→insert(low, T_SB→front());
17:        T_SB→pop_front();
18:  }
}
```

Figure 4.4 SODA - update the sequential order

Now, let us see an example of T passing the validation in a complex case.

Example 3: Let $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$ be a set of committed transactions and the sequential order, and T be a validating transaction at a server. The read sets, write sets and the timestamps are shown in Table 4.3.

Since $WS(T_3) \cap RS(T) \neq \varnothing$ and $WS\_TS(T_3) > T \rightarrow TS(a)$, T must-be-serialized-before T_3 and *up* = T_3. Similarly, *low* = T_6. Since $SO(low) > SO(up)$, this is the complex case. T_SB = $\{T_3, T_4\}$, and none of T_3 and T_4 must-be-serialized-before either T_5 or T_6, thus, T passes the validation test. T_3 and T_4 are removed first and then T, T_3 and T_4 are inserted immediately after T_6, the final sequential order is $\{T_1, T_2, T_5, T_6, T, T_3, T_4, T_7\}$ as shown in Figure 4.5, where the arrow (→) in the graph indicates the serialization order between two transactions such as $T_1 \rightarrow T_3$ (T_1 is serialized before T_3).

Table 4.3 Transaction information used in Example 3

	T_1	**T_2**	**T_3**	**T_4**	**T_5**	**T_6**	**T_7**	**T**
RS	{x}	{y}	{z}	{a}	∅	{b}	{c}	{a}
WS	{z}	{x}	{a}	∅	{b, c}	∅	∅	{b}
TS (d)	5	15	25	35		45	50	28
TS_WS	10	20	30		40			+∞

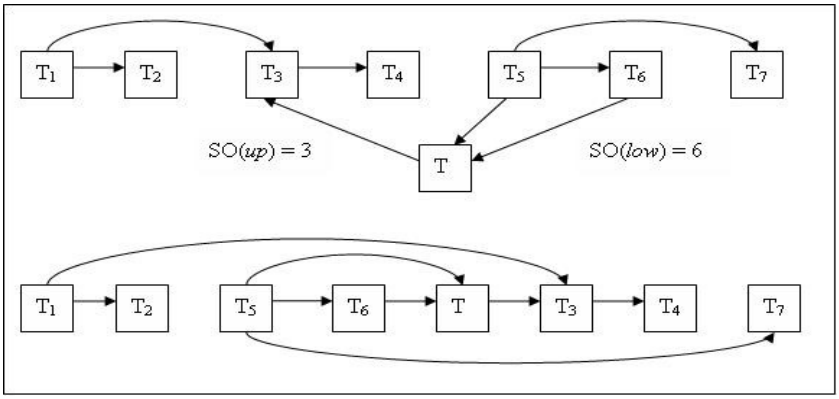

Figure 4.5 Validating transaction T in Example 3

It is time to give another example in which the sequential order is not adjustable because of the existence of a cycle.

Example 4: Let $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$ be a set of committed transactions and their sequential order. T is a validating transaction. Their read sets, write sets and their timestamps are shown in Table 4.4.

Table 4.4 Transaction information used in Example 4

Transactions	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_6**	**T_7**	**T_8**	**T**
Read set (RS)	{x}	∅	{a}	{a}	∅	{b}	∅	{c}	{y, a}
Write set (WS)	{y}	{x}	{a}	∅	{b}	∅	{b, c, w}	∅	{w}
TS (d)	5	∅	20	30	∅	40	∅	50	8, 18
Timestamp of WS	10	15	25		35		45		+∞

Since $WS(T_1) \cap RS(T) \neq \emptyset$ and $WS\_TS(T_1) > T \rightarrow TS(y)$, T must-be-serialized-before T_1 and and *up* = T_1. Since $WS(T_7) \cap WS(T) \neq \emptyset$ and $WS\_TS(T_7) < WS\_TS(T)$, T

must-be-serialized-after T_7 and *low* = T_7. Because SO(*low*) > SO(*up*), T must_be_serializes_after T_7, T_SB = {T_1, T_2, T_3, T_4}, and T_SB conflicts with T_7; thus, a cycle is detected, the sequential order is not adjustable and T cannot pass the validation. As shown in Figure 4.6, it is easy to see that there is a cycle: $T \rightarrow T_3 \rightarrow T_4 \rightarrow T_7 \rightarrow T$. Actually, T3 and T4 are part of T_SB; therefore, the cycle can be simplified as $T \rightarrow T\_SB \rightarrow T_7 \rightarrow T$.

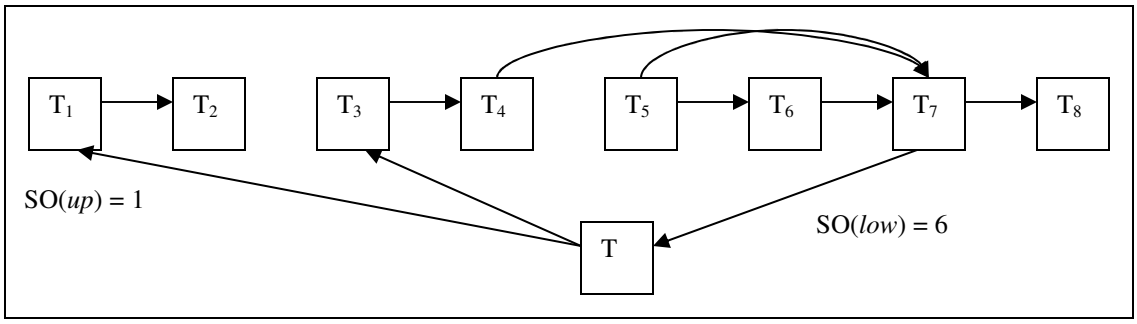

Figure 4.6 Validating transaction T in Example 4

4.2.2 Proof of correctness

To prove the correctness of SODA, we must show that any schedule produced by SODA is serializable. To fulfil this goal, we utilize the Serializability Theorem "*A schedule S is serializable iff SG(S) is acyclic*" [Bernstein, 1987], that is, we must prove that the new serialization graph is still acyclic after the addition of a newly committed transaction that has passed the validation test.

Lemma 1: Given a sequential order {T_1, T_2, T_3, …, T_n} produced by SODA, SODA either does not create any cycle or detects every cycle, if any, in SG({T_1, T_2, T_3, …, T_n}+{T}) during the validation of any committing transaction T.

Proof: Since the sequential order of transactions complies with their serialization order, every edge (T_i, T_j), if any, must have the same direction. In other words, the edge (T_i, T_j) goes from left to right because $SO(T_i) < SO(T_j)$, where $1 \leq i, j \leq n$.

In the simple case, SODA does not create any cycle in SG({T_1, T_2, T_3, …, T_n}+{T}): Since *low* and *up* are found and SO(*low*) < SO(T) < SO(*up*), all newly added edges are either (T_i, T) or (T, T_j) where $SO(T_i) \leq SO(low)$ and $SO(up) \leq SO(T_j)$. Therefore, all existing edges and newly added edges must have the same direction, i.e. going from left to right, and thus it is impossible for T to involve any cycle.

In the complex case, SODA captures every cycle in SG({T_1, T_2, T_3, …, T_n}+{T}): Since *low* and *up* are found, but SO(*low*) ≥ SO(*up*) in the sequential order and, consequently, T may be involved in cycles, such as, T → [*up*] →… T_i … → [*low*] → T, where SO(*up*) < $SO(T_i)$ < SO(*low*), and [*up*] and [*low*] are optional.

Without loss of generality, let the cycle be T → T_{i1} →… T_{im} → T, where SO(*up*) ≤ $SO(T_{ik})$ ≤ SO(*low*), $i_1 \leq i_k < i_m$, and i_m equals to the number of nodes/transactions in the cycle and between *up* and *low* in the sequential order. Now, we prove that SODA captures every cycle during the validation for T by the induction on i_m.

The basic step, for i_m = 1: that is, the cycle is T → T_{i1} → T. Since T → T_{i1}, the function must-be-serialized-before(T, T_{i1}) returns true (line 24 in Figure 4.3). Since T_{i1} → T, the function must-be-serialized-after(T, T_{i1}) returns true (line 25 in Figure 4.3). Thus, SODA returns false because a cycle is detected (line 26 in Figure 4.3).

The induction step for i_m = k: Suppose every cycle is detected for $i_m \leq k$, that is, the cycle T → T_{i1} → … T_{ik-1} → T_{ik} → T is detected because T_SB = {T_{i1}, T_{i2}, …, T_{ik-1}},

$T_{ik-1} \rightarrow T_{ik}$ and $T_{ik} \rightarrow T$ (lines 22 to 26 in Figure 4.3). Actually, this cycle is equivalent to $T \rightarrow T\_SB \rightarrow T_{ik} \rightarrow T$. Now, we show that every cycle is detected for $i_m = k+1$. Since $T_{ik-1} \rightarrow T_{ik}$, and T is not serialized after T_{ik} directly, T_{ik} is also added into T_SB (lines 24 to 29 in Figure 4.3). Since $T_{ik} \rightarrow T_{ik+1}$ and T_{ik} is part of the T_SB and $T_{ik+1} \rightarrow T$, the cycle $T \rightarrow T_{i1} \rightarrow \ldots \rightarrow T_{ik} \rightarrow T_{ik+1} \rightarrow T$ (or $T \rightarrow T\_SB \rightarrow T_{ik+1} \rightarrow T$) is detected as well. Thus, SODA returns false for $i_m = k+1$ (lines 22 to 26 in Figure 4.3).

Therefore, SODA either captures every cycle, if any, or does not create any cycle in $SG(\{T_1, T_2, T_3, \ldots, T_n\}+\{T\})$ during the validation of any committing transaction T.

Theorem 1: If S is a schedule produced by SODA, then S is serializable.

Proof: By Lemma 1, SODA either detects every cycle in SG(S) or does not create any cycle when it validates any committing transaction, so SG(S) is acyclic. Thus, S is serializable according to the Serializability Theorem [Bernstein, 1987].

4.2.3 Complexity analysis

Theorem 2: The time complexity of SODA is $(p*n^2 + n) = O(n^2)$, where n is the number of committed transactions in the sequential order, and p is the probability of a committing transaction conflicting with both *low* and *up* and SO(*low*) > SO(*up*).

Proof: Assume that the number of operations in a transaction is constant and the time to check if two transactions conflict is also constant [Hwang, 2000]. *In the simple case (case 1)*: SODA runs one FOR loop after another to find *low* and *up*, and the maximum number of iterations in each loop is n (lines 2 to 17 in Figure 4.3). *In the complex case (case 2)*: SODA runs two nested FOR loops to test the possibility of

dynamic adjustment, the maximum number of iterations in each loop is n, and the probability of the complex case to happen is p (lines 18 to 32 in Figure 4.3). *In update sequential order* (c*ase 3)*: SODA runs one FOR loop and one WHILE loop to update the sequential order, and the maximum number of iterations in each loop is n (lines 3 to 18 in Figure 4.4). By combining the three cases above, the complexity of SODA is:

$$p*n^2 + n = O(n^2)$$

If we assume the conflict probability between two transactions is x, then the value of x will be very small ($0 < x < 1$) as most of transactions in MANET are read-only, and thus, x^2 will be even smaller. Since p is the probability of a committing transaction conflicting with both *low* and *up* and SO(*low*) ≥ SO(*up*), $p < x^2$. For instance, if x = 0.01, then $p < x^2 = 0.0001$. Therefore, we can safely claim that SODA mostly runs in the linear time. In contrast, the complexity of a serialization graph testing algorithm is always $O(n^2)$ [Hwang, 2000].

4.3 How SODA Works in a Clustered MANET Database

In order to make SODA work effectively in a clustered MANET database, the coordinating server functionality is combined with the cluster head's functionality because a cluster head is elected by our MEW algorithm [Xing, 2010] as described in Chapter 3 and is the nearest server with the highest energy in clients' neighborhood. This would enable clients to save time, limited battery energy and bandwidth that they must spend on identifying suitable servers to which they send their transactions. Therefore, only three major functionalities are required: the primary cluster head

functionality, cluster head functionality, and participating server functionality as shown in Figure 4.7. Note that one server can have all the three functionalities at the same time.

4.3.1 The transaction execution model

As shown in Figure 4.7, a transaction T issued by a client is distributed to its cluster head; the cluster head divides T into sub-transactions and transmits them to the appropriate participating servers according to the global database schema. Each participating server processes the sub-transactions locally and sends the results back to the cluster head. The cluster head runs the 2-Phase Commit (2PC), and gathers all results from the participating servers. Note that we adopt 2PC here due to its simplicity as our research goal is to develop a concurrency control algorithm, not a commit algorithm; however, we do plan to include a more suitable commit protocol for MANET databases in our future work. If running 2PC successfully, the cluster head sends T to the primary cluster head to validate T globally based on the SO of committed global transactions; otherwise, the cluster head sends an abort message directly to the client. After receiving the global validation result, the cluster head sends the final results to the client.

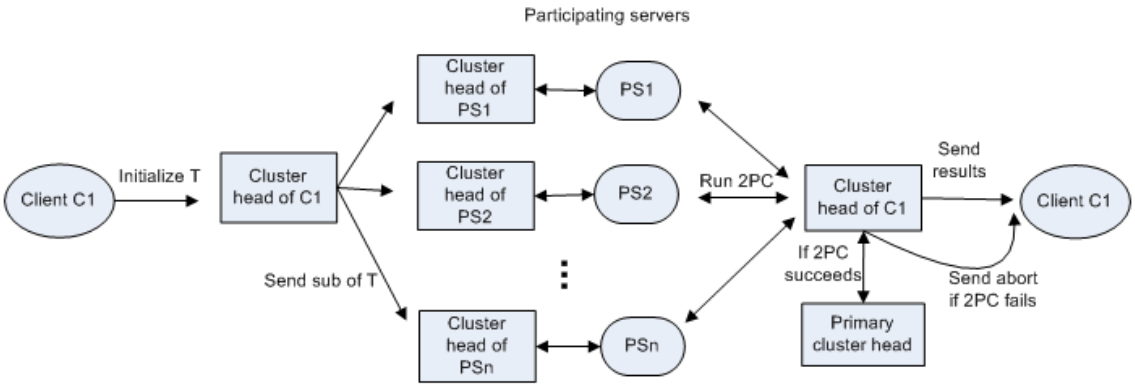

Figure 4.7 Workflow of SODA

4.3.2 The primary cluster head functionality

The primary cluster head has the following functionalities:

- It maintains the sequential order (SO) of committed global transactions.
- It receives global transaction validation requests from non-primary cluster heads.
- It validates global transactions using SODA. After validation, it sends the validation results to the non-primary cluster head.
- It updates the SO after a global transaction commits and adds this global transaction's read set, write set and the timestamp of both sets to the data structure of the maintained SO.
- It removes the old committed transactions that are not serialized after any active/committed global transaction from the maintained SO after a global transaction commits.
- It periodically checks (after a global transaction commits) its remaining energy level. If its level is below a predefined threshold *LET* and another cluster head's remaining level is above the threshold, it resigns its cluster head status and elects a new primary cluster head that has the highest remaining energy from all cluster heads. It then transfers the information of all the transactions it stores to the new primary cluster head. Note that since the primary cluster head is also a non-primary one, if the primary one resigns, the non-primary one also resigns if there is a candidate in the neighborhood.

4.3.3 The cluster head functionality

The cluster head has the following functionalities:

- It receives a global transaction from a client, divides them into sub-transactions, and sends the sub-transactions to appropriate participating servers.
- It runs 2PC to request the status of the sub-transactions and requests the timestamps of the global transaction's read set.
- It propagates the global transaction to the primary cluster head after it receives all successful messages of the sub-transactions. After receiving the validation result, it sends the final results to the client.
- It periodically checks (after a global transaction commits) its remaining energy level. If the level is below a predefined threshold and there is a candidate for cluster head in the neighborhood, it resigns its cluster head status and elects a new cluster head in the neighborhood. It then transfers the information of all the transactions it stores to the new cluster head. Note that if the old cluster head is also the primary cluster head, then the new cluster head can be the new primary cluster head as well if this new one has the highest remaining energy among all cluster heads.

4.3.4 The participating server functionality

A participating server has the following functionalities:

- It receives and processes sub-transactions, and maintains the SO of committed sub-transactions.
- It runs SODA locally based on the local SO of committed sub-transactions when it receives the request about the status of the sub-transactions.
- It sends the final status of the sub-transactions to the requesting cluster head. It also sends the timestamps of the read sets of the sub-transactions to the cluster head if the sub-transactions pass the validation.
- It updates the local SO of committed sub-transactions if a sub-transaction commits and adds this sub-transaction's read set, write set and timestamps of both sets to the data structure of the maintained SO. It removes the old committed sub-transactions that are not serialized after any active/committed sub-transaction from the maintained SO after a sub-transaction commits.

4.4 Conclusions

In this chapter, we introduced and proved the correctness of our energy-efficient CC algorithm, called Sequential Order with Dynamic Adjustment (SODA), for mission-critical MANET databases in a clustered network architecture. In SODA, in order to conserve energy and balance the energy consumption among servers so that the lifetime of the network is prolonged, we elected cluster heads using our weighted clustering algorithm MEW to work as coordinating servers. SODA is based on optimistic CC to offer high concurrency and avoid unbounded blocking time. It utilizes the sequential order of committed transactions to simplify the validation process, and dynamically

adjusts the sequential order of committed transactions to reduce transaction aborts. Its complexity is $O(n^2)$, where n is the number of committed transactions in the sequential order.

CHAPTER 5

PERFORMANCE EVALUATION OF MEW USING THE NS-2 SIMULATOR

In this chapter, we present the performance evaluation of our network clustering algorithm MEW (Mobility, Energy, and Workload) using simulation. First, we describe the simulation parameters and performance metrics. We then present and analyze the simulation results.

5.1 Simulation Description and Parameters

The performance of MEW and MOBIC [Basu, 2001] is evaluated using the NS-2 simulator with clustering framework [Basagni, 2006]. Table 5.1 lists the simulation parameters, most of which are the same as the ones in [Basu, 2001]. Since mobility is the major cause of re-clustering, the weighting factor of mobility $f_1 = 0.8$, the weighting factor of energy $f_2 = 0.15$ and the weighting factor of workload $f_3 = 0.05$ are used. The initial energy level of each node is randomly distributed between 20% and 100%.

To measure the stability of a clustered MANET, we consider the following metrics:

- The lifetime of the network: the duration from the beginning of the simulation until a node runs out of its energy [Choi, 2006; Sheu, 2006].
- The cluster head change rate (per second): the total number of cluster heads is divided by the total simulation time [Basu, 2001].

- The re-affiliation (joining a cluster and becoming a member) rate (per second): the total number of cluster members is divided by the total simulation time [Choi, 2006].

Table 5.1 Simulation parameters

Parameter	Value	Reference
Number of nodes (N)	50	[Basu, 2001]
Simulation area	670 * 670 meters^2	[Basu, 2001]
Maximum speed of node movement	1, 10, 20, 30 meters/second (or m/s)	[Basu, 2001]
Transmission range (TR)	10 meters – 250 meters	[Basu, 2001]
Pause time (PT)	0 second, 30 seconds	[Basu, 2001]
Broadcast interval (BI)	1 second	
Cluster contention interval (CCI)	3 seconds	
Low energy threshold (LET)	30%	
Mobility weighting factor (f_1)	0.8	
Energy weighting factor (f_2)	0.15	
Workload weighting factor (f_3)	0.05	
Initial energy level	20% - 100%	
Simulation time	200 seconds	[Viswacheda, 2007]

5.2 Simulation Results

This section presents the results of the experiments performed by varying maximum speed and transmission range. All metrics in the following figures are collected from an average value of 50 simulation runs in 50 different scenarios, which are randomly generated using the random waypoint model (built-in in NS-2). To better mimic a real wireless network, 25 constant bit rate (CBR) connections are randomly generated by the traffic-scenario generator. Each source sends a 512-byte packet through

UDP (User Datagram Protocol) [Wang, 2007b] at a rate of one packet per second.

5.2.1 Effect of maximum speed

In this experiment, the maximum node moving speed is varied to study the effect on the performance. The maximum speed of a node is varied from 1 m/s to 30 m/s. The experiment results are shown in Figures 5.1 - 5.3.

In Figure 5.1, the lifetime of the network decreases as the maximum node moving speed increases in both MOBIC and MEW. This is expected because nodes with higher speed are more likely to become neighbors or get disconnected. This will trigger more cluster head changes and more re-affiliations, thus, more energy are consumed to maintain clusters. Regardless of the pause time PT = 0s or 30s, MEW prolongs the lifetime of the network by 9% to 42% (or 23% on average) better than that of MOBIC. Since mobility is dealt with in the same way in both MOBIC and MEW, how to address mobility is not the main cause of longer lifetime of the network. In other words, these promising results confirm affirmatively the effects of taking into consideration the energy and workload in the weight calculation and the forced resignation of a cluster head when its remaining energy becomes too low.

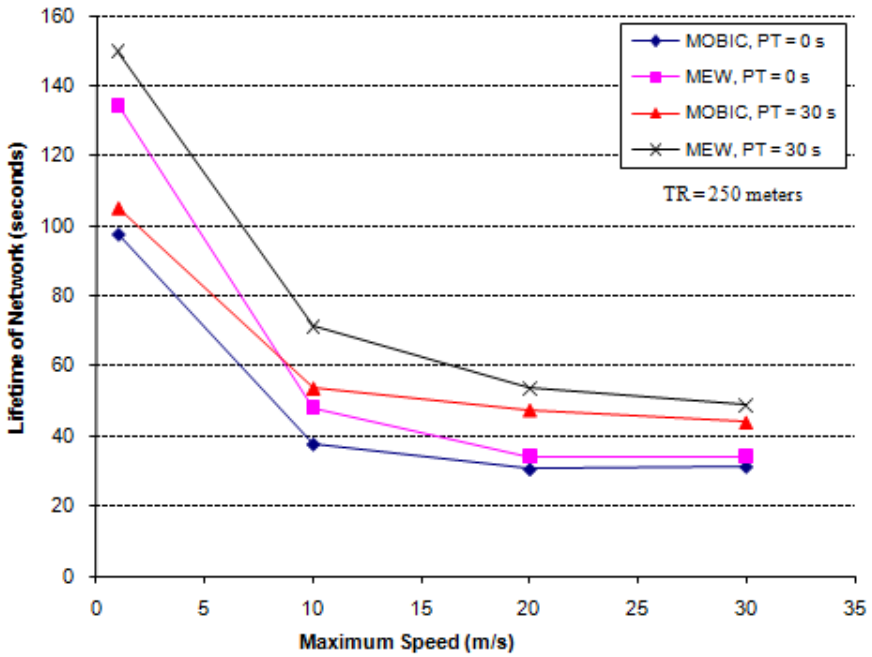

Figure 5.1 Lifetime of network by varying maximum speed

Figure 5.2 shows that the cluster head change rate of MEW and MOBIC increases as the node speed increases no matter PT = 0s or PT = 30s. This is because cluster heads with higher speed are more likely to become neighbors and, consequently, the one with a lower weight has to resign. MEW produces 5 to 12 (or 7 on average) fewer cluster heads than MOBIC when PT = 0s and PT = 30s, respectively. Given that both MOBIC and MEW deal with mobility using the relative mobility, MEW still produces fewer cluster heads mainly because nodes with higher energy are likely to get elected as cluster heads and, hence, they can function as cluster heads for a longer time.

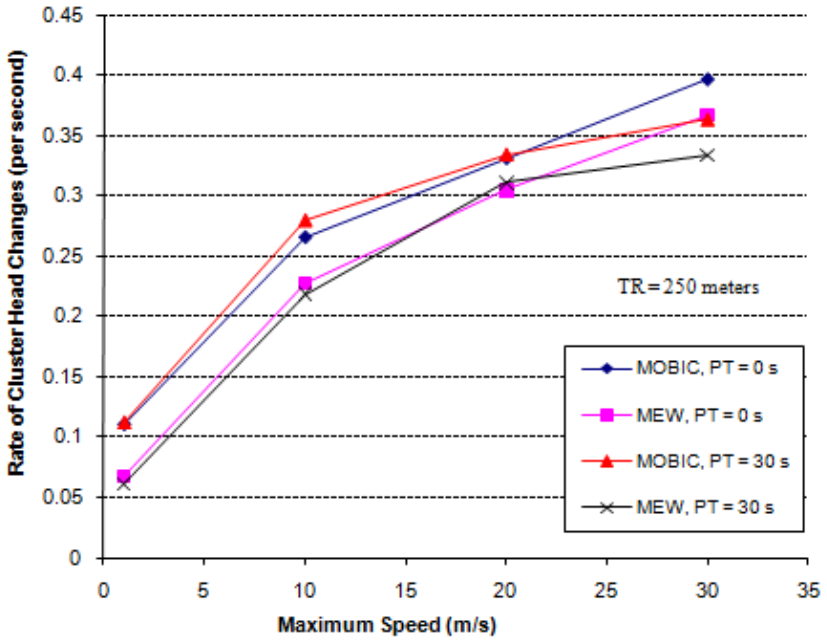

Figure 5.2 Rate of cluster head changes by varying maximum speed

Figure 5.3 shows that the re-affiliation rate of MEW and MOBIC increases as the node speed increases. This is because cluster members with higher speeds are likely to get disconnected from their original cluster heads and join other cluster heads. MEW produces 34 to 66 (or 44 on average) fewer cluster members than MOBIC for both PT = 0s and PT = 30s. The advantage of MEW having a lower re-affiliation rate is mainly attributed to the less likelihood of the resignation of a cluster head due to energy exhaustion.

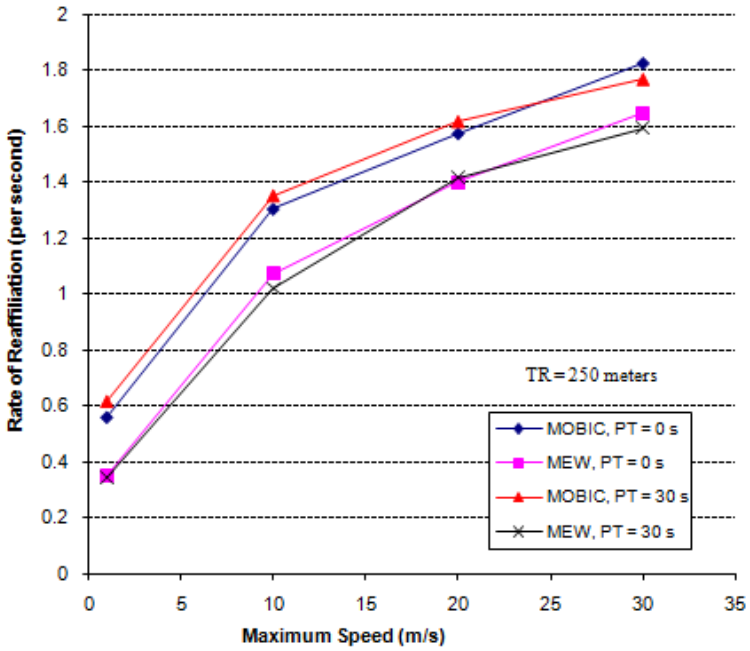

Figure 5.3 Rate of re-affiliation by varying maximum speed

5.2.2 Effect of transmission range

In this experiment, the transmission range is varied to study the effect on the performance. The transmission range of a node is varied from 10 meters to 250 meters. The experiment results are shown in Figures 5.4 - 5.6.

In Figure 5.4, the lifetime of the network decreases as the transmission range increase in both MOBIC and MEW. This is expected because the larger the transmission range is, the more energy is required to transmit packets, and thus, the more energy is consumed. MEW outperforms MOBIC by 2% to 27% (or 15% on average) when the transmission range is larger than 50 meters. These promising results confirm that MOBIC is mobility-only-based algorithm. In other words, MOBIC does not consider energy during cluster head election, so some nodes with low remaining energy become cluster heads and, consequently, these cluster heads run out of energy soon.

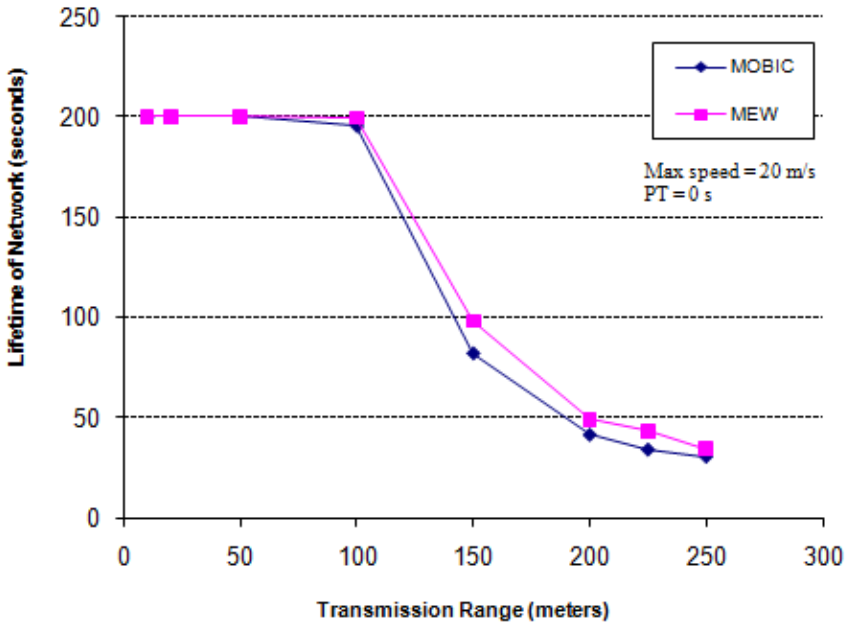

Figure 5.4 Lifetime of network by varying transmission range

In Figure 5.5, the cluster head change rate of both algorithms increases when the transmission range is less than 50 meters, this is expected because more nodes appear within range of each other for shorter periods of time as the transmission range increases, so that more cluster heads have to give up their roles and join others as cluster members. However, when the transmission range becomes larger than 50 meters, the cluster head change rate decreases as more nodes are within range of other nodes and stay together for longer periods of time. MEW produces 5 to 13 (or 10 on average) fewer cluster heads than MOBIC. Given that both MOBIC and MEW deal with mobility using the same way, MEW still produces fewer cluster heads mainly because nodes with higher energy are likely to get elected as cluster heads and, hence, they can function as cluster heads for a longer time.

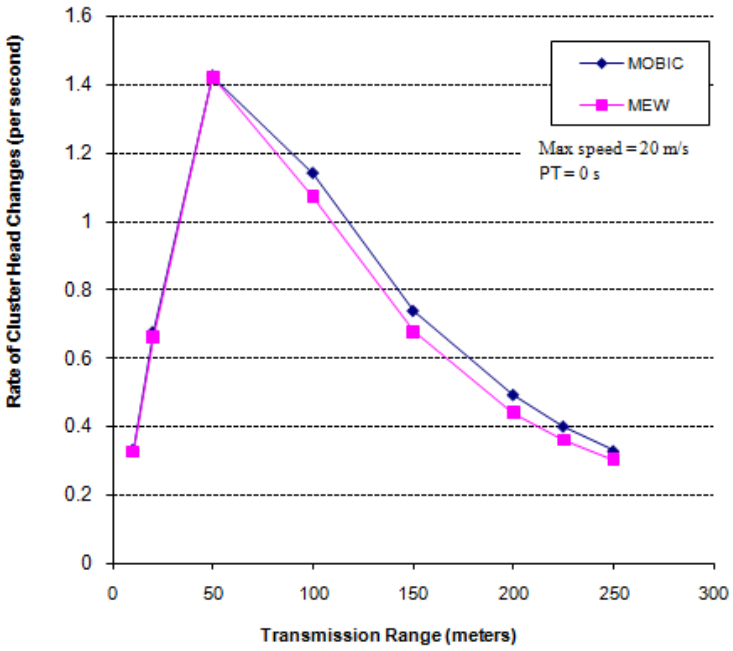

Figure 5.5 Rate of cluster head changes by varying transmission range

In Figure 5.6, the re-affiliation rates of MEW and MOBIC increase as the transmission range is less than 100 meters and increases. This is expected because more nodes appear within range of more than one cluster heads and join the one with largest weight. However, when the transmission range becomes larger than 100 meters, the re-affiliation rates of both algorithms decrease because cluster members are within range of their cluster heads and stay together for longer periods of time. MEW produces 34 to 43 (or 40 on average) fewer re-affiliations than MOBIC when the transmission range is greater than or equal to 100 meters. Due to the same solution of addressing mobility, the advantage of MEW having a lower re-affiliation rate is mainly attributed to the less likelihood of the resignation of a cluster head due to energy exhaustion.

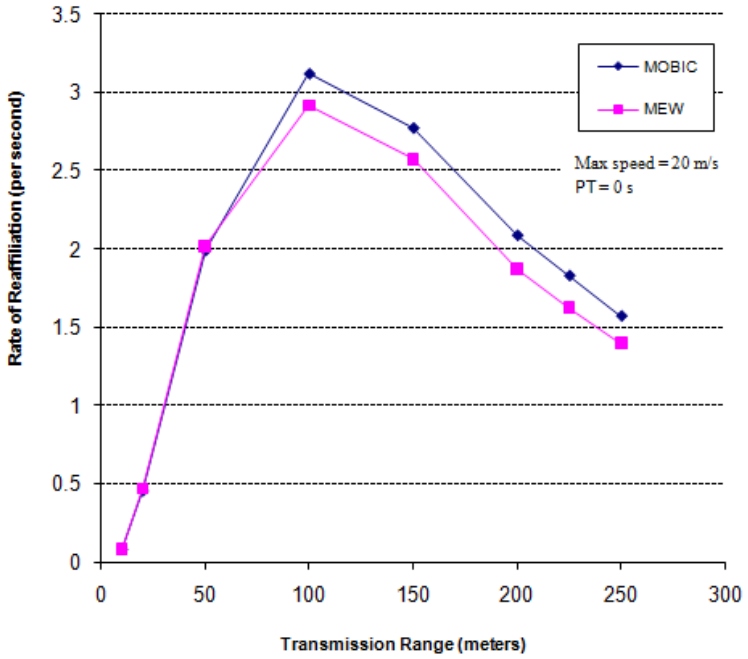

Figure 5.6 Rate of re-affiliation by varying transmission range

5.3 Conclusions

In this chapter, we presented the performance evaluation of our weighted clustering algorithm MEW using NS-2 simulation by varying the maximum node moving speed and the transmission range. MEW is compared with MOBIC. The simulation results show that MEW prolongs the lifetime of MANETs and has a lower cluster head change rate and re-affiliation rate than the existing algorithm MOBIC.

CHAPTER 6

PERFORMANCE EVALUATION OF SODA USING THE SIMULATIONS

The simulation experiments are conducted to compare the performance of our proposed SODA with those of SESAMO [Brayner, 2005] and the most widely used CC protocol - S2PL (Strict 2-Phase Locking) [Bernstein, 1987]. As we discussed in Chapter 2, SESAMO relaxes atomicity and global serializability due to its assumption. However, global serializability is guaranteed by S2PL when S2PL is combined with 2PC [Abdouli, 2005].

Our simulation model consists of a transaction generator, a real-time scheduler that schedules transactions using early deadline first [Pabmanabhan, 2006], participating servers, coordinating servers or cluster heads for SODA only, and a deadlock manager for SESAMO and S2PL. In the SODA model shown in Figure 4.7, a transaction T issued by a client is transmitted to its cluster head CHc; CHc divides T into several sub-transactions, and transmits them to the appropriate participating servers through their cluster heads according to the global schema. Each participating server processes the sub-transactions locally, and sends the results back to CHc. CHc runs the 2PC and gathers all results from the participating servers. If running 2PC successfully, CHc sends T to the primary cluster head to validate T globally based on the SO of committed global transactions; otherwise, CHc sends an abort message directly to the client. After receiving the global validation result, CHc sends the final results to the client.

The simulation models for SESAMO and S2PL are similar to that of SODA except for a couple of points. One is that SODA is applied locally and globally to validate transactions, while in SESAMO, strict 2PL is applied globally [Brayner 2005] and locally [Holanda, 2008], and in S2PL, strict 2PL is run only locally. The other point is that SESAMO and S2PL have no any cluster head and use coordinating servers instead.

Three simulation models are built to compare SODA with S2PL and SESAMO. All three simulation models are implemented using the AweSim simulation language [Pritsker, 1999]. Each simulation model is defined in the following three aspects: mobile hosts, transactions and mobility model [Li, 2004]. The static parameters and dynamic parameters about the database and system settings are shown in Tables 6.1 and 6.2. These values are chosen in order to create scenarios with high utilization of data and more data contention. Since transactions in mission-critical applications must be executed not only correctly but also within their deadlines where,

$$Deadline = creation\ time + (estimated\ execution\ time + estimated\ disconnection\ time) * slack\ factor$$

In other words, we use real-time firm transactions to evaluate the performance. Therefore, in our simulation, a transaction will be aborted if either it missed its deadline or the system could not complete it successfully (e.g. when it is aborted by the CC technique).

1. **Mobile nodes:** In the simulation model, 10 servers and 40 clients are system resources, randomly deployed in 3 areas initially, and the radius of each area is about 100 meters. Each mobile node is assigned with a unique id, x and y coordinates as location, moving direction and initial energy level between 80%

and 100%. Each of the servers stores a portion of the whole database, and the data stored on one server are not replicated on other servers. The transmission range of a server is 250 meters and of a client is 100 meters. The bandwidth is fixed at 11 Mbps according to the current wireless technology such as the Intel Wireless WiFi Link 5300 wireless card [Intel, 2008]. The server is modelled from the Lenovo Thinkpad T400s notebook [Notebookcheck, 2009], which has Intel Core 2 Duo SP 9600 2.53 GHz CPU, a 4 GB DDR3 RAM and 23240 MIPS (Million Instructions per Second). The client is modelled from the HP iPAQ 210, which has Marvell PXA310 624MHz Processor with the 128MB SDRAM [HP, 2008] and 800 MIPS.

2. **Transactions:** Global transactions are entities, request and release system resources during the execution. Transaction start time, transaction id, transaction type (read-only or write), deadline, and number of sub-transactions are assigned when they are generated. The inter-arrival time, proportion of read-only transactions, number of sub-transactions, and number of participating servers are defined in Table 6.1 and Table 6.2.

3. **Mobility Model:** Mission-critical (or tactical) applications are strictly structured (e.g., platoons in military operation) and their actions are strictly organized. There is a leader or a group of leaders who tells everybody where and how to move or in which area to work. In general, their movements are driven by tactical reasons. Due to this, the units normally use the optimal path to a destination. The destinations depend on the work area that is based on tactical issues. The tactics as well as the scene are usually hierarchically organized. Typically, the site is

divided into different tactical areas. Each unit belongs to one of these areas. For example, in a disaster rescue scenario, firefighters belong to an incident site and medical workers are in the casualty's treatment area. Once the units are sent to a specific location, they stay close to this location. Thus, the area in which a unit moves depends on tactical issues but is restricted to one specific area [Aschenbruck, 2008]. The simulation area is fixed in a 1000x1000 meters^2 region. All the nodes are divided into groups, and in each group, nodes are moving within a relative direction angle being in the range (-30^o, 30^o) [Lu, 2008] and the moving direction is random from a set of eight possible directions (↗, ↙, ↖, ↘, ←, →, ↑, ↓) [Li, 2004]. By placing 10 servers and 40 clients onto the region with the size of 1000x1000 meters^2, the MANET is assumed to remain good connectivity, implying the network partitions occur rarely.

6.1 Simulation Parameters and Performance Metrics

The simulation static parameters and their values are shown in Table 6.1. Note that in a clustered MANET, cluster heads are more stable than non-cluster head nodes; so cluster heads should have lower disconnection probability than non-cluster head nodes. In order to include this observation, the percentage of disconnection that cluster heads can have is set to 10%. For example, if the default disconnection probability is 0.3, then the disconnection probability of cluster heads is 0.27 = 0.3 – 10% * 0.3.

Table 6.1 Static parameters

Parameter	Values	Reference
Server energy consumption rate in active mode	30.3 Watts	[Notebookcheck, 2009]
Server energy consumption rate in idle mode	12.5 Watts	[Notebookcheck, 2009]
Client energy consumption rate in active mode	0.99 Watts	[HP, 2008]
Server transmission range	250 meters	[Zhang, 2010]
Client transmission range	100 meters	[HP, 2008]
Speed of server processor	2.53GHZ(23240 MIPS)	[Notebookcheck, 2009]
Speed of client processor	624MHZ(800 MIPS)	[HP, 2008]
Packet size	512 bytes	[Zhang, 2010]
Bandwidth	2 Mbps	[Zhang, 2010]
No. of sites in global transaction	Triangular(3,4,5)	[Li, 2004]
No. of operations or sub-transactions	Uniform(5, 10)	[Lei, 2009]
CPU computation time	10 ms	[Lei, 2009]
Low energy threshold	50%	
Percentage off disconnection probability due to being a stable cluster head	10%	
No. of clients	40	[Li, 2004]
No. of servers	10	
Slack factor	4	[Lei, 2009]
Simulation area	1000x1000 meters^2	[Li, 2004]

Table 6.2 Dynamic parameters

Parameter	Value Range	Default Value	Reference
Mean inter-arrival time	1 to 10 seconds (exponentially distributed)	5	[Leu, 2007; Lei, 2009]
Proportion of read-only transactions	0.1 to 0.85	0.8	[Li, 2004; Nouali, 2010]
Disconnection probability	0.1 to 0.9	0.3	[Li, 2004]
Mean disconnection time	1 to 10 seconds	5	[Guo, 2008; Lei, 2009]
Node moving speed	1 to 10 m/s	3	[Denko, 2009; Li, 2007]

The dynamic parameters, their value ranges and their default values are listed in Table 6.2. We use these five dynamic parameters to study their effects on the performance of the concurrency control algorithms.

- **Inter-arrival time** is the mean of an exponentially distributed time between the arrivals of two consecutive transactions; it varies over the range from 1 to 10 seconds in order to vary the system load [Gruenwald, 2007] and create a scenario with high data contention.

- **Proportion of read-only transactions** is the percentage of read-only transactions among the total simulated transactions. More read-only transactions mean fewer conflicts among transactions. In other words, proportion of read-only transactions can also create a scenario with high or low data contention.

- One of the major characteristics of MANET is frequent disconnections due to the mobility and energy limitation of nodes and unreliable wireless communication between nodes; so two disconnection parameters are studied: **disconnection probability** and **mean disconnection time**. Disconnection probability is the probability of communication that is disconnected when a node tries to communicate with another node. Disconnection time is the time interval during which a node is unavailable to communicate with.

- **Node moving speed** varies from 1 to 10 m/s to study the effect of node mobility on the performance.

Eight performance metrics are used and they are defined in Equations (6.1), (6.2), (6.3), (6.4), (6.5), (6.6),(6.7) and (6.8), respectively: total time when servers are in active mode, abort rate, system throughput, average validation time that the primary cluster head spends on a global transaction, response time, total number of cluster head reelections, total energy consumed by all servers, and average difference in remaining energy between two servers. Among these metrics, total time when servers are in active mode, average validation time that the primary cluster head spends on a global transaction, and total number of cluster head reelections are utilized to support other performance metrics.

The first performance metric is the total time when servers are in active mode. A server is in active mode only if it is processing transactions; otherwise, it is in doze mode to save energy. This metric evaluates whether servers are busy to process transactions most of time, where m is the total number of servers and $T_{a,i}$ is the total time when server S_i is in active mode.

$$Total\ \ time\ when\ servers\ \ are\ \ in\ active\ mode\ =\ \sum_{i=1}^{m} T_{a,i} \tag{6.1}$$

The second performance metric is the abort rate to measure the percentage of aborted transactions, and can be computed as below:

$$Abort\ rate = \frac{Total\ \#\ of\ aborted\ \ transactions}{Total\ \#\ of\ \ generated\ \ \ transactions} * 100\% \tag{6.2}$$

The third performance metric is the system throughput to measure the performance of a database system in terms of the number of transactions completed in a minute. Note that the time unit is not second because transaction response time is larger

than a second.

$$System\ throughput = \frac{Total\ \#\ of\ committed\ transactions}{Total\ simulation\ time / 60} \tag{6.3}$$

The fourth performance metric is the average validation time that the primary cluster head spends on a global transaction. It is the elapsed time between submitting a global transaction to the primary cluster head for validation and receiving the validation result. It is used to verify how long the primary cluster head prolongs the transaction response time, where t_s is the time at which a global transaction is submitted by the client's cluster head (or called coordinating server), and t_e is the time at which the validation result is received by the same coordinating server.

$$Average\ \ validation\ \ time\ = t_e - t_s \tag{6.4}$$

The fifth performance metric is the transaction response time that is the elapsed time between submitting a database transaction for execution and receiving a response. It is used to evaluate how an application is performing in the measurement of time, where t_s is the time at which a transaction is submitted by a client, and t_e is the time at which a response is received by the same client. The major influences on transaction response time are communication delays and the database access time for data items accessed by the transaction.

$$Transaction\ response\ \ time\ = t_e - t_s \tag{6.5}$$

The sixth performance metric is the total number of cluster head (primary and non-primary) reelections to evaluate whether an algorithm takes balancing energy among servers into consideration, where N_{primay} ($N_{non\text{-}primay}$) is the number of primary (non-primary) cluster head reelections. However, more reelections do not guarantee more

balanced energy among servers because there is an overhead of transferring the information from the old cluster head to the new one.

$$Total\ number\ of\ cluster\ head\ reelections = N_{primay} + N_{non-primary} \quad (6.6)$$

The seventh performance metric is the total amount of energy consumed by all servers in both active mode and doze mode. This metric evaluates how energy-efficient each technique is, where m is the total number of servers, ECR_a (ECR_d) is the energy consumption rate when a server is in active (doze) mode, and $T_{a,i}$ ($T_{d,i}$) is the total time when server S_i is in active (doze) mode.

$$Total\ energy\ consumed\ by\ servers = \sum_{i=1}^{m} (ECR_a * T_{a,i} + ECR_d * T_{d,i}) \quad (6.7)$$

The eighth performance metric is the average difference in remaining energy between two servers to evaluate how balanced the system is in terms of energy consumption. The more balanced the system is, the longer lifetime the system has. This metric is computed using the following formula, where m is the total number of servers, and RE_i *and* RE_j are the remaining energy of servers S_i and S_j, respectively.

$$Average\ difference\ in\ remaining\ energy\ between\ two\ servers = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} |RE_i - RE_j|}{(m-1)*m} \quad (6.8)$$

6.2 Simulation Results

This section presents the results of the experiments performed. In each simulation run, 1000 transactions are simulated and results are collected at the end of each run. When one dynamic parameter is studied, all other dynamic parameters are

fixed with their default values specified in Table 6.2. The three compared algorithms are labelled as S2PL, SESAMO and SODA in the result figures.

6.2.1 Effect of inter-arrival time

In this experiment, the inter-arrival time between two consecutive transactions is varied to test the system load and create scenarios with low or high data contention. The inter-arrival time is generated using the exponential distribution with mean from 1 second to 10 seconds. The experiment results are shown in Figures 6.1 - 6.8.

Figure 6.1 shows that the total time when servers are in active mode of S2PL, SESAMO and SODA increases as the transaction inter-arrival time increases. When transactions enter into the database system at a slow inter-arrival rate, which is the reciprocal of inter-arrival time, system has low workload. Thus, transactions have less waiting time for resources and have more chances to complete before their deadlines. Since the more transactions are committed, the more time servers spend on processing these committed transactions. It is obvious that SESAMO performs the worst, SODA performs the best and S2PL is in the middle. Furthermore, the increasing rate of SEASAMO and S2PL is much higher than that of SODA. This happens because S2PL and SESAMO are pessimistic and utilize locks to hold limited system resources to prevent conflicting transactions from accessing them. In other words, servers in S2PL and SESAMO have to be in active mode most of time to keep processing transactions. SESAMO performs the worst because it takes SESAMO more time to run strict 2PL locally and globally.

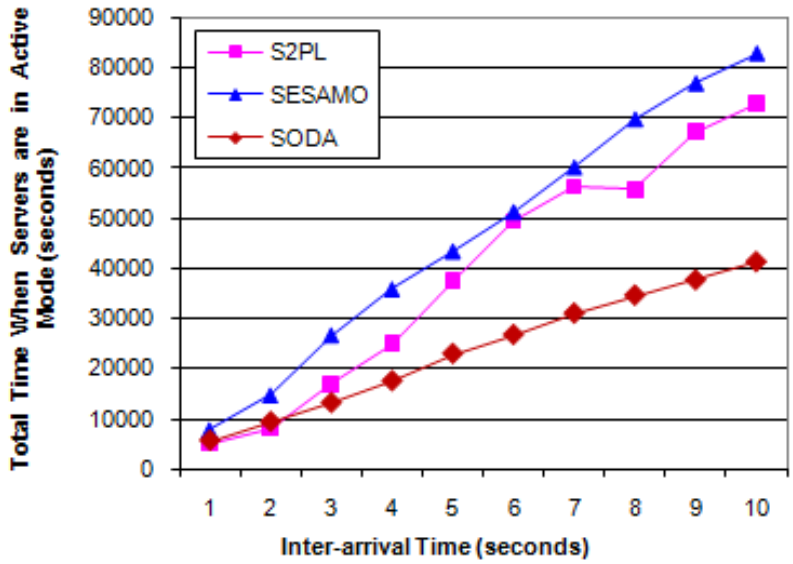

Figure 6.1 Total time when servers are in active mode vs. inter-arrival time

In Figure 6.2, the abort rates of S2PL, SESAMO and SODA decrease when the transaction inter-arrival time increases. This is expected because when fewer transactions are in the system, fewer conflicts among transactions, so that servers are not overloaded, and transactions have less waiting time for resources and have more chances to commit before their deadlines. The abort rate of SODA is much lower than those of SESAMO and S2PL right after the inter-arrival time is longer than 1 second. This is mainly because transactions arrive at the system with a slow rate, and conflicts among transactions become rare, so that optimistic algorithms perform better than pessimistic algorithm due to no prevention of conflicts overhead. SESAMO's abort rate is lower than S2PL's except after the inter-arrival time = 9 seconds. Although SESAMO does not enforce global serializability, it still blocks many conflicting transactions due to running strict 2PL both locally and globally. S2PL runs strict 2PL locally only, but it enforces

global serializability using 2PC. In other words, in S2PL, all locks of sub-transactions are held until global transactions commit, which also increases significant waiting time of conflicting transactions. When the inter-arrival time is getting shorter, it is easy to see that the abort rate of SODA is close to SESAMO's and S2PL's because conflicts among transactions increase; in addition, this confirms the fact that optimistic CC techniques work well only if conflicts among transactions are rare.

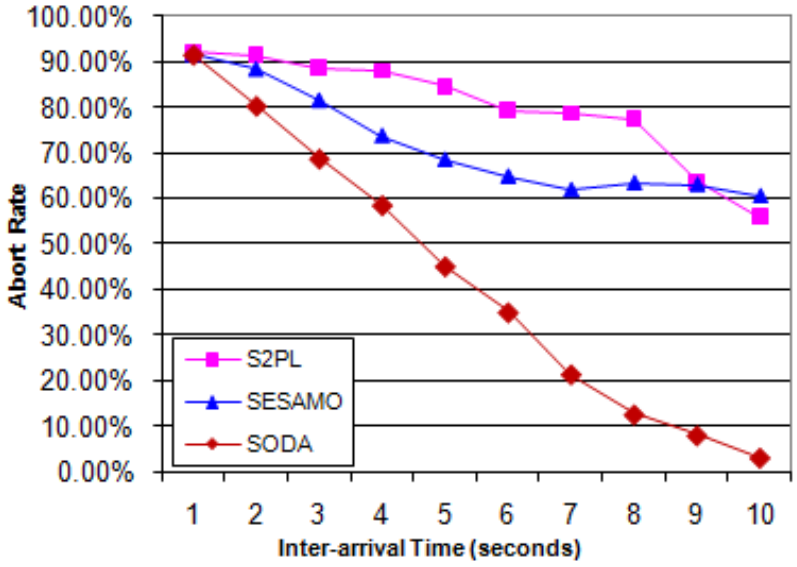

Figure 6.2 Abort rate vs. inter-arrival time

Figure 6.3 shows that the system throughput of the three algorithms does not have strict trends of increase or decrease as the inter-arrival time increases. This seems not correct because the throughput should increase as the inter-arrival time increases due to the facts demonstrated in Figure 6.2: fewer transactions are aborted as the inter-arrival time increases. However, when the inter-arrival time increases, more transactions are

committed, but at same time, the total simulation time becomes longer as well. The system throughput of SODA is as least two more transactions/minute than those of SESAMO and S2PL right after the inter-arrival time is longer than 1 second. This is mainly because SODA can commit more transactions than S2PL and SESAMO at each inter-arrival time (based on the fact: the less abort rate an algorithm has as shown in Figure 6.2, the more transactions it can commit), but the corresponding total simulation time is about the same. SESAMO's system throughput is higher than S2PL's when the transaction inter-arrival time is between 2 seconds and 8 seconds.

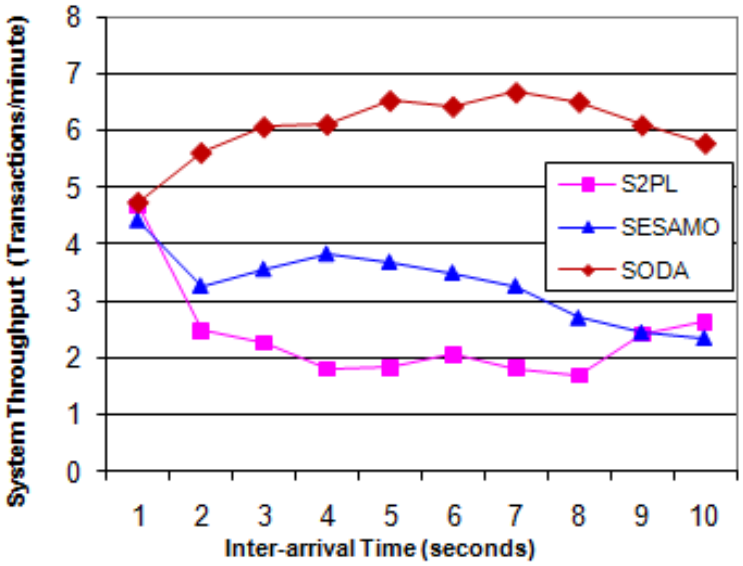

Figure 6.3 System throughput vs. inter-arrival time

Figure 6.4 shows SODA's average validation time that the primary cluster head spends on a global transaction increases first and then decreases as the inter-arrival time increases. However, the average validation time of S2PL and SESAMO is always zero because their designs do not involve any cluster head. In other words, S2PL and

SESAMO do not prolong the response time due to the primary cluster head.

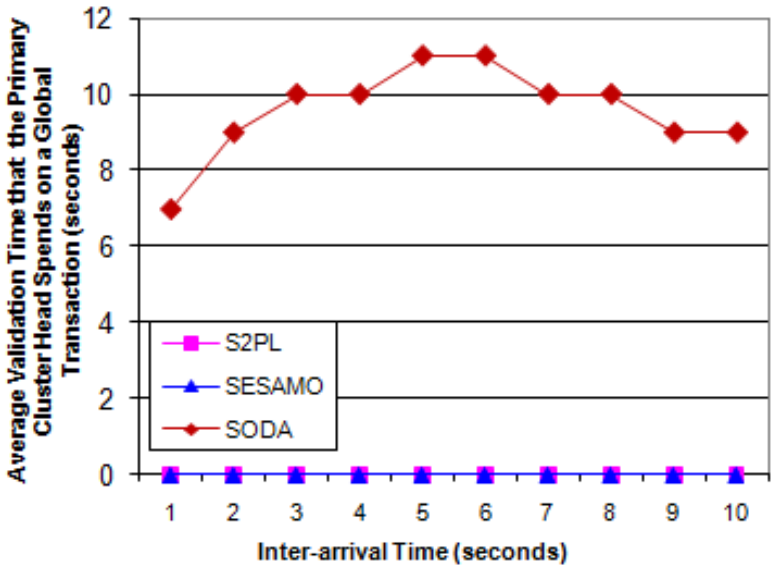

Figure 6.4 Average validation time that the primary cluster head spends on a global transaction vs. inter-arrival time

As shown in Figure 6.5, the response time of the S2PL, SESAMO and SODA roughly decreases as the inter-arrival time increases after the inter-arrival time is longer than 3 seconds. This trend is expected because servers are not overloaded due to low transactions entry rate, so that servers can process transactions in time. The response time of SODA is higher than those of S2PL and SESMO when the inter-arrival time is between 1 second and 9 seconds. This happens because SODA utilizes the primary cluster head to validate all global transactions and enforce the global serializability, thus, the primary cluster head has the bottleneck problem and, consequently, the transaction processing time is prolonged as shown in Figure 6.4. However, the prolonged response time in SODA is reasonable (averagely 17 seconds longer) because it is still within the transaction deadline; otherwise, SODA should not have the lowest abort rate shown in

Figure 6.2. SESAMO has shorter response time than S2PL right after the inter-arrival time = 5 seconds. Although shorter response time is expected by every algorithm, but the trade off has to be done among all performance metrics. In other words, SODA trades off the response time for lower abort rate, higher throughput, lower energy consumed by all servers and balancing energy better among all servers.

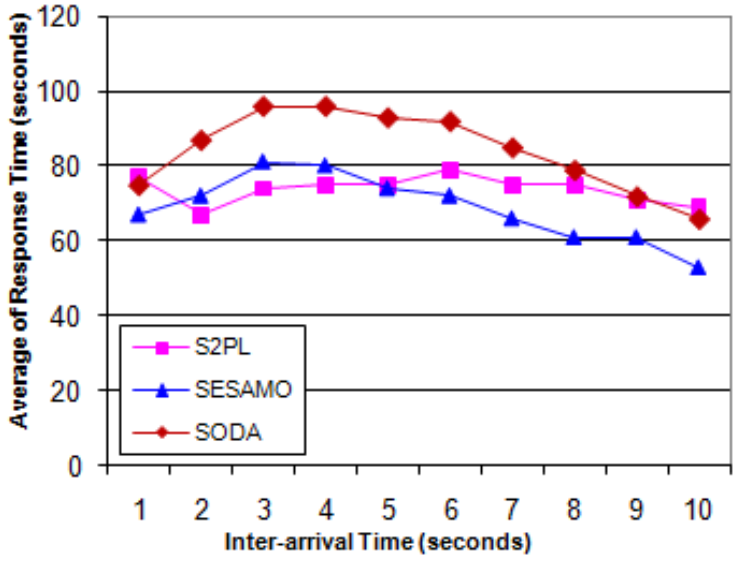

Figure 6.5 Average of response time vs. inter-arrival time

Figure 6.6 shows the total number of cluster head reelections of SODA increases as the inter-arrival time increases. When the inter-arrival time reaches 10 seconds, the total simulation time is around 3 hours (1000 transactions * 10 seconds = 10,000 seconds). Consequently, more cluster heads have the remaining energy below the predefined threshold *LET*, and more reelections are triggered to change roles for preserving energy and balancing energy usage. However, the total number of reelections

of S2PL and SESAMO is always zero because their designs do not involve any cluster head. In other words, S2PL and SESAMO do not rotate roles among servers to balance energy.

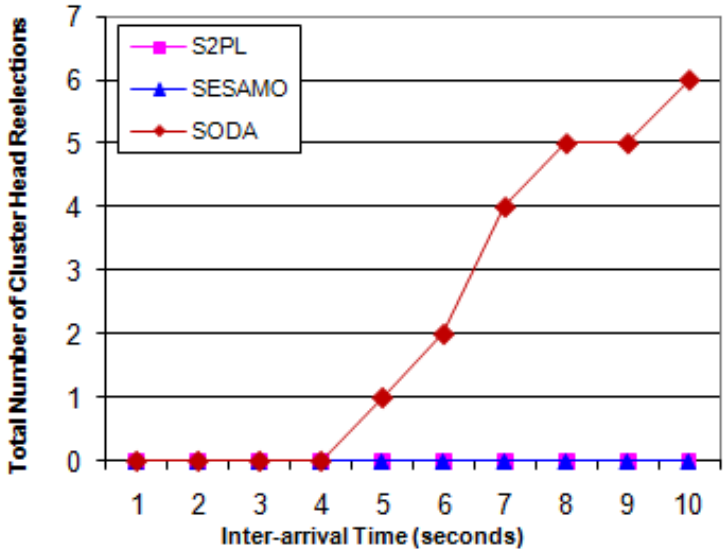

Figure 6.6 Total number of cluster head reelections vs. inter-arrival time

Figure 6.7 shows that the total energy consumption of all servers increases with the increase of the inter-arrival time. This is expected because more transactions are committed as inter-arrival time increases as shown in Figure 6.2, so that each server has to spend more time in active mode on processing these committed transactions as shown in Figure 6.1. In other words, the more transactions are committed and the more time servers are in active mode, the more energy is consumed, and Figure 6.7 confirms this fact. SODA consumes at least 64,632 J and at most 563,676 J less than both S2PL and SESAMO right after the inter-arrival time is longer than 2 seconds. This happens because transactions arrive into the system with a slow rate, and conflicts among

transactions become much rarer, so that optimistic SODA performs better than pessimistic S2PL and SESAMO due to no prevention of conflicts overhead.

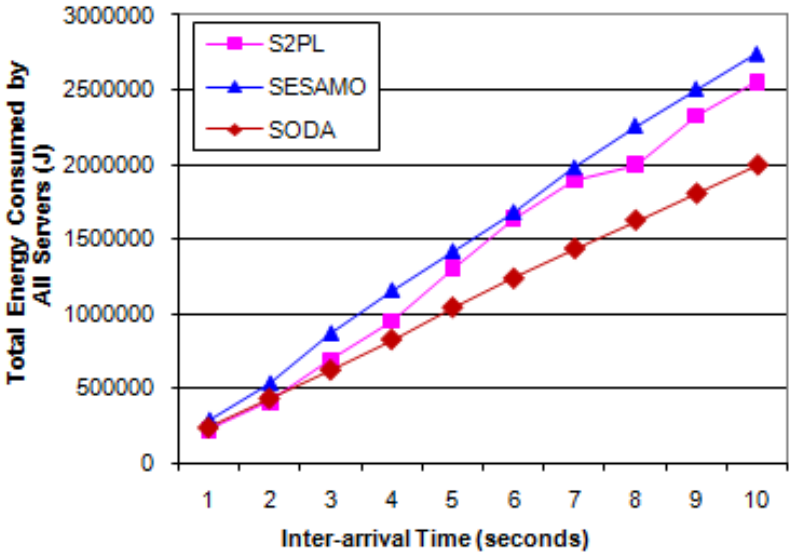

Figure 6.7 Total energy consumed by all servers vs. inter-arrival time

The average difference in the remaining energy between two servers in the three algorithms does not have strict trends of increase or decrease as the inter-arrival time increases as shown in Figure 6.8. Through this metric, we want to check whether the energy consumption is balanced among servers. If a technique does not balance energy consumption among servers, some servers may run out of energy quickly and, consequently, those servers without energy affect the whole database system. It is easy to see that SODA is the best to balance energy consumption, and S2PL does the worst except when the inter-arrival time = 2 seconds. This is because more non-primary cluster heads and primary cluster heads with higher energy are reelected as shown in Figure 6.6. However, in S2PL and SESAMO, there is no role rotation strategy and clients may keep

submitting transactions to the same servers so that these servers are overloaded.

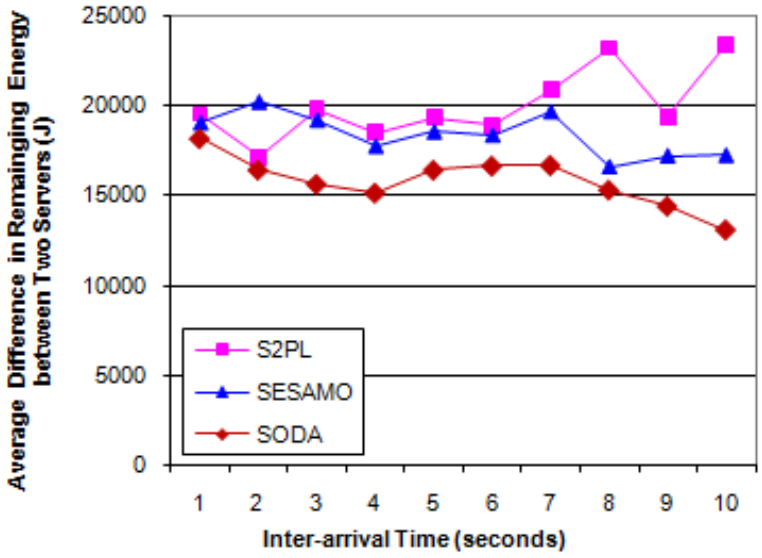

Figure 6.8 Average difference in remaining energy between two servers vs. inter-arrival time

6.2.2 Effect of proportion of read-only transactions

In this experiment, the proportion of read-only transactions is varied to test the system load and create scenarios with low or high data contention similar to the inter-arrival time discussed in Section 6.2.1. The experiment results are shown in Figures 6.9-6.16.

When more read-only transactions are initiated, conflicts between transactions become rare, thus more transactions have chances to complete before their deadlines. The more transactions are committed, the more time servers spend on processing these committed transactions. Figure 6.9 confirms that the total time when servers are in active mode in S2PL and SESAMO roughly increases when the proportion of read-only

transactions increases. However, SODA does not follow the trend; instead, its total time almost remains unchanged because it is not sensitive to the proportion of read-only transactions. It is obvious that SESAMO performs the worst, SODA performs the best and S2PL falls into the middle. This reflects the fact that S2PL and SESAMO are pessimistic and utilize locks to hold data to prevent conflicting transactions from accessing the common data even though most of transactions are read-only.

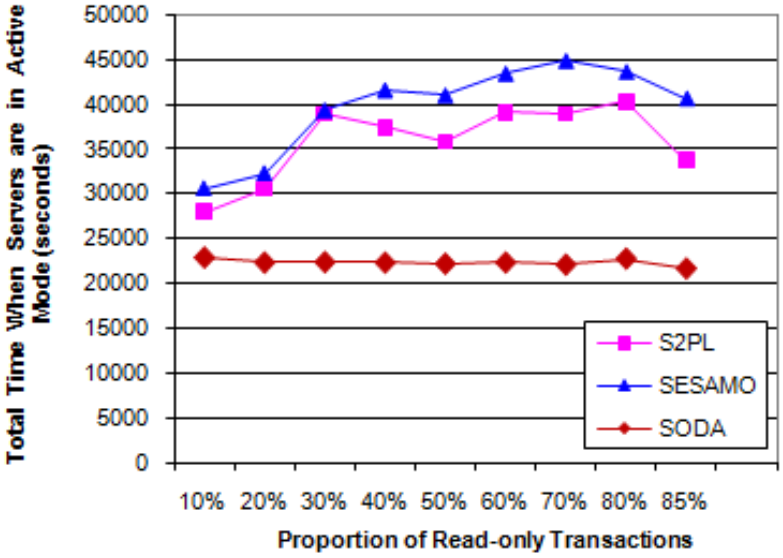

Figure 6.9 Total time when servers are in active mode vs. proportion of read-only transactions

In Figure 6.10, the abort rates of S2PL, SESAMO and SODA decrease when the proportion of read-only transactions increases, but SODA does not decrease significantly because it is not sensitive to this parameter once the proportion of read-only transactions > 10%. The abort rate of SODA is much lower (at most 47%) than those of SESAMO and S2PL. This is mainly because SODA is optimistic and conflicts among transactions

become rarer as the proportion of read-only transactions increases, so that SODA can perform optimally due to the optimistic CC algorithm existence assumption: conflicts among transactions are rare. SESAMO's abort rate is lower than S2PL's right after the proportion is 30%. This implies that S2PL running the 2PC to guarantee global serializability causes more aborts than SESAMO running the strict 2PL at the global level because both S2PL and SESAMO run the strict 2PL at the local level.

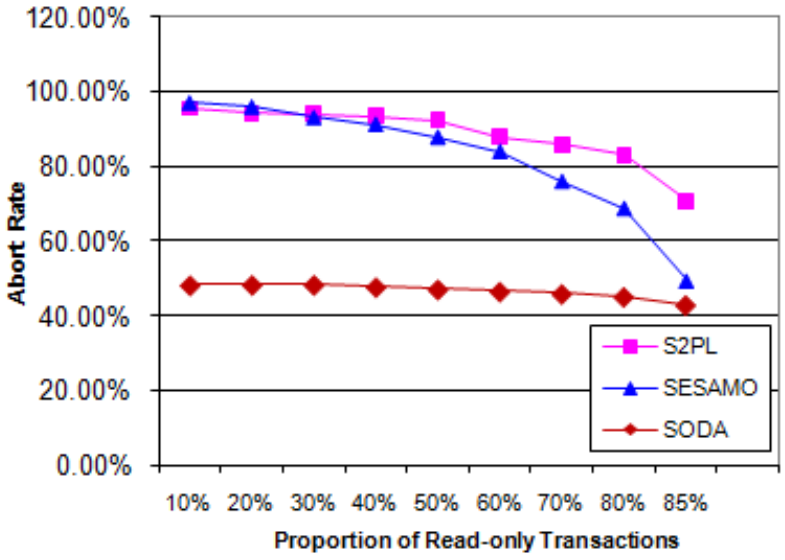

Figure 6.10 Abort rate vs. proportion of read-only transactions

Figure 6.11 shows that the system throughput of these three algorithms increases as the proportion of read-only transactions increases. This happens because conflicts between transactions become rarer when more read-only transactions are in the system, so that transactions do not compete with each other for common data and commit before their deadlines. The system throughput of SODA is as least two more transactions/minute than those of SESAMO and S2PL when the proportion of read-only transactions <=

80%. This further confirms that SODA can perform optimally due to the assumption of optimistic CC algorithm existence: conflicts among transactions are rare. SESAMO's system throughput is higher than S2PL's when the proportion of read-only transactions > 20%, and this trend is consistent with the fact that SESAMO has less abort rate than S2PL as shown in Figure 6.10.

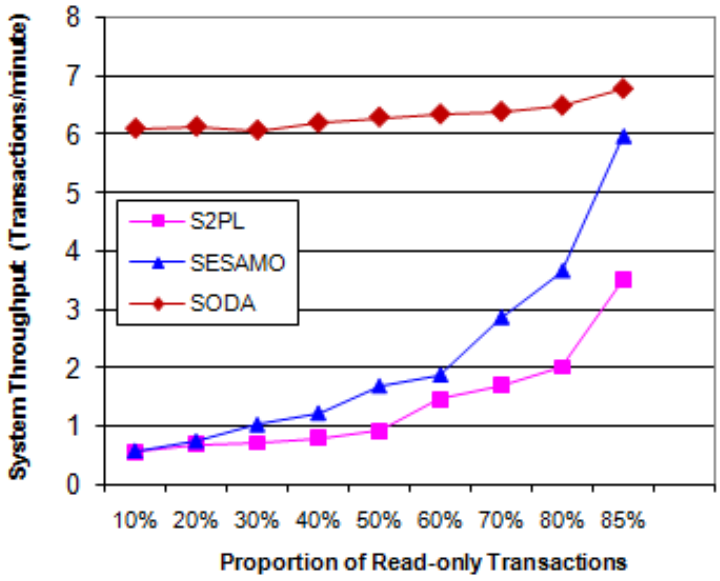

Figure 6.11 System throughput vs. proportion of read-only transactions

Figure 6.12 shows SODA's average validation time that the primary cluster head spends on a global transaction has no significant changes as the inter-arrival time increases. However, the average validation time of S2PL and SESAMO is always zero because their designs do not involve any cluster head. In other words, S2PL and SESAMO do not prolong the response time due to the primary cluster head.

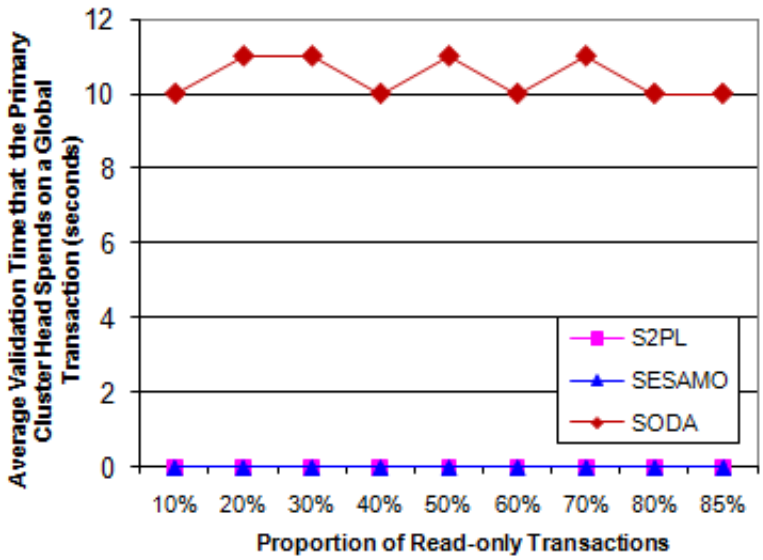

Figure 6.12 Average validation time that the primary cluster head spends on a global transaction vs. proportion of read-only transactions

As shown in Figure 6.13, the response time of S2PL and SESAMO roughly increases when the proportion of read-only transactions increases. However, SODA does not follow the trend; instead, it has almost the same response time because it is not sensitive to this parameter once the proportion of read-only transactions > 10%. It is obvious that the response time of SODA is higher than that of S2PL and SESMO. This happens because SODA has the bottleneck problem due to primary cluster head and, consequently, the transaction processing time is prolonged as shown in Figure 6.12. However, the prolonged response time in SODA is still within the reasonable range (averagely 32 seconds longer); otherwise, SODA should not have the lowest abort rate shown in Figure 6.10. S2PL and SESAMO have shorter response time alternately. Although shorter response time is expected by every algorithm, but the trade off has to be done among all performance metrics. In other words, SODA trades off the response time for lower abort rate, higher throughput, lower energy consumed by all servers and

balancing energy better among all servers.

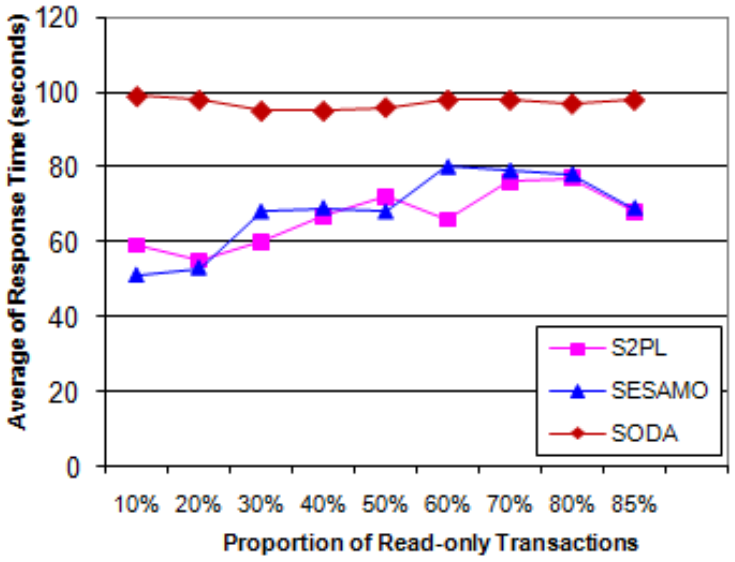

Figure 6.13 Average of response time vs. proportion of read-only transactions

Figure 6.14 shows the total number of cluster head reelections of SODA occurs only once when the proportion of read-only transactions is 80%. This is because the inter-arrival time is fixed with its default value 5 seconds when we study the effect of the proportion of read-only transactions, thus, the total simulation time is around 1.5 hours (1000 transactions * 5 seconds = 5,000 seconds) no matter how the proportion of read-only transactions varies. After running 1.5 hours, most cluster heads' remaining energy is not below the predefined threshold *LET* yet, therefore, only one reelection is triggered to change roles for preserving energy and balancing energy usage. However, the total number of reelections of S2PL and SESAMO is always zero because their designs do not involve any cluster heads. In other words, S2PL and SESAMO do not rotate roles among servers to balance energy.

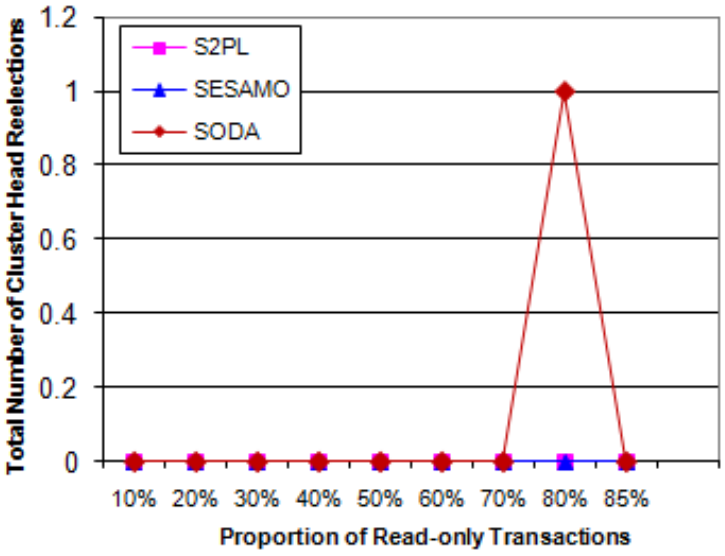

Figure 6.14 Total number of cluster head reelections vs. proportion of read-only transactions

Figure 6.15 shows that the total energy consumption of all servers in S2PL and SESAMO roughly increases with the increase of the proportion of read-only transactions. This is expected because more transactions are committed as the proportion of read-only transactions increases as shown in Figure 6.10, so that each server has to spend more time in active mode on processing transactions as shown in Figure 6.9. In other words, the more transactions are committed, the more energy is consumed, and Figure 6.15 confirms this fact except for SODA. However, SODA does not follow the trend due to reaching its transaction processing capacity or not being sensitive to this parameter. In addition, SODA consumes at least 178,615 J and at most 311,750 J less than both S2PL and SESAMO when the proportion of read-only transactions > 20%. This happens because SODA is optimistic and is not in active mode most of the time as shown in Figure 6.9.

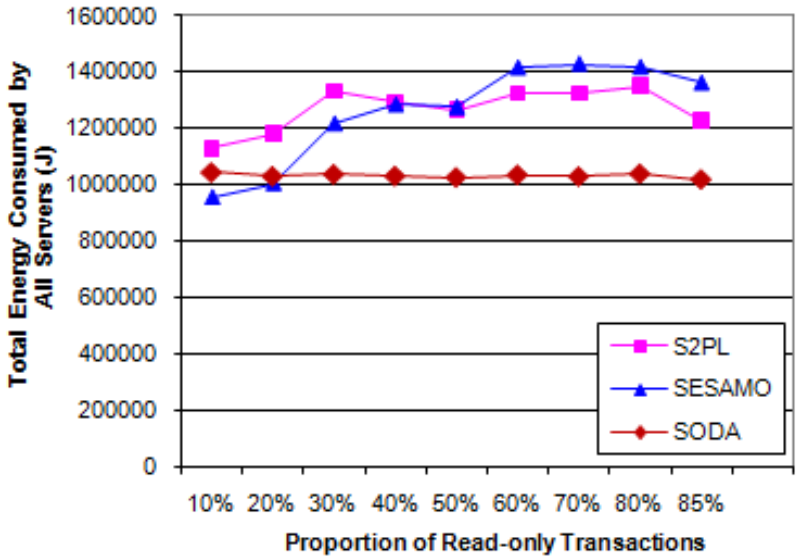

Figure 6.15 Total energy consumed by all servers vs. proportion of read-only transactions

In Figure 6.16, the average difference in remaining energy between two servers does not have strict trends of increase or decrease as the proportion of read-only transactions increases. It is obvious that SODA is the best to balance energy consumption, and S2PL does the worst except when the proportion of read-only transactions = 40%. This is because SODA elects nodes with higher remaining energy and less workload to be cluster heads, and these cluster heads work as coordinating servers and will be reelected when their remaining energy is low. However, in S2PL and SESAMO, there is no clustering and role rotation strategy and clients may keep submitting transactions to the same servers so that these servers are overloaded.

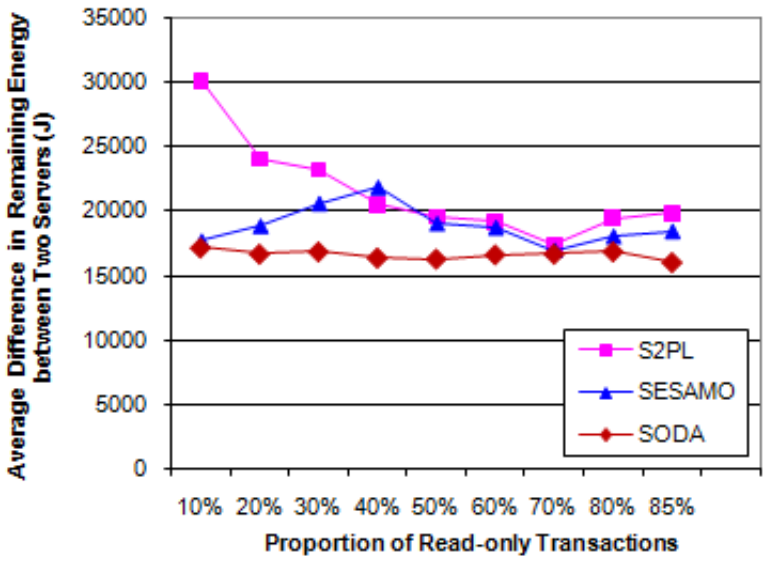

Figure 6.16 Average difference in remaining energy between two servers vs. proportion of read-only transactions

6.2.3 Effect of disconnection probability

In this experiment, the disconnection probability is varied to study the effect on the performance due to frequent disconnections in a MANET. The experiment results are shown in Figures 6.17-6.24.

Figure 6.17 shows that in SODA, the total time when servers are in active mode increases as the disconnection probability increases, but in S2PL and SESAMO, this metric does not always increase or decrease as the disconnection probability increases. It is obvious that SESAMO performs the worst, and SODA performs the best. This confirms the fact that S2PL and SESAMO cannot work effectively in MANETs. In other words, even though servers are disconnected in S2PL and SESAMO, their data are still locked by some transactions, so that these servers have to be in active mode to keep processing transactions.

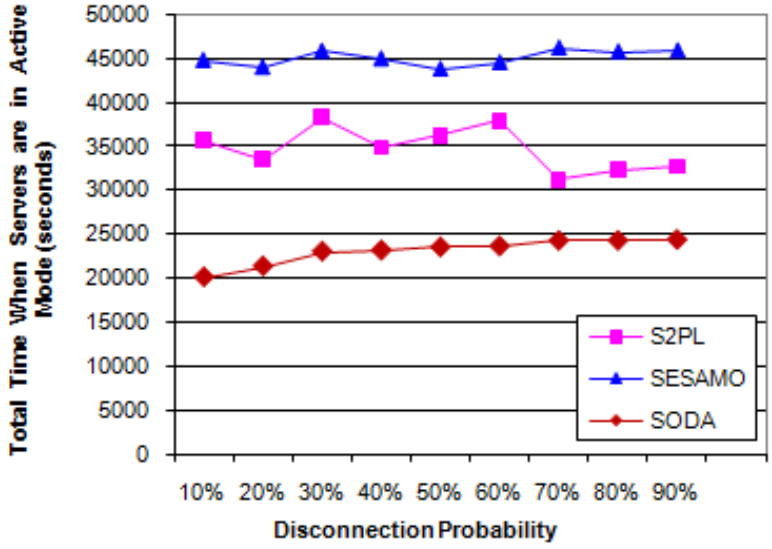

Figure 6.17 Total time when servers are in active mode vs. disconnection probability

As shown in Figure 6.18, the abort rates of S2PL, SESAMO and SODA increase when the disconnection probability increases. This reflects the fact that fewer servers are available as more and more servers are disconnected. The abort rate of SODA is much lower (at most 37%) than those of SESAMO and S2PL when the disconnection probability < 70%. This is mainly because S2PL and SESAMO utilize locks to prevent conflicting transactions from accessing common data and now servers are frequently disconnected, so that lots of transactions are aborted because they could not access the required data and thus missed their deadlines. SESAMO's abort rate is lower than S2PL's because S2PL runs strict 2PL locally along with 2PC to enforce global serializability, but 2PC does not work effectively when disconnections are frequent in the network. In other words, since 2PC needs two rounds of communications between the coordinating sever and participating servers to determine a commit or not, and now

servers are frequently disconnected, it takes significant waiting time for 2PC to finish, and thus, more transactions are aborted due to missing their deadlines.

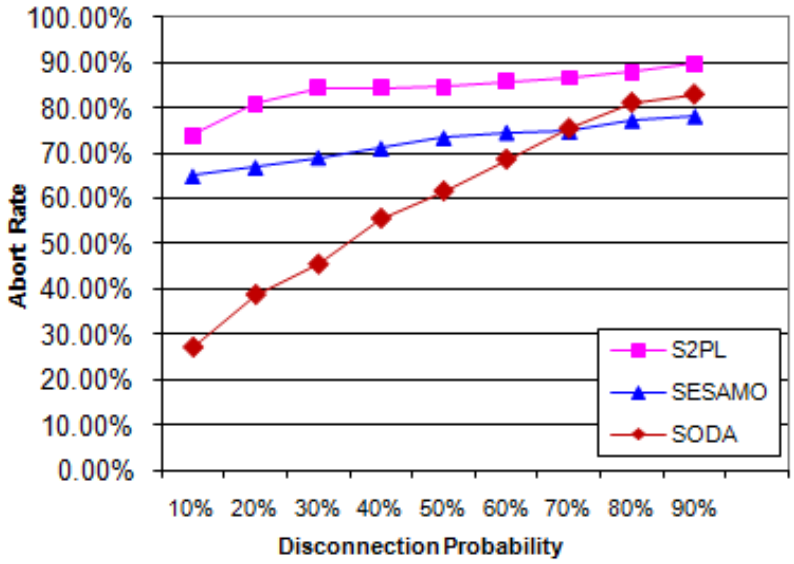

Figure 6.18 Abort rate vs. disconnection probability

Figure 6.19 shows that the system throughput of these three algorithms decreases as the disconnection probability increases. This happens because servers are frequently disconnected and are not available to process transactions, so that lots of transactions are aborted because they missed their deadlines as shown in Figure 6.18. The system throughput of SODA is higher than that of SESAMO and S2PL until the disconnection probability = 70%. This is still mainly because SODA is optimistic and non-blocking, so that servers do not lock data and can process transactions in time. SESAMO's system throughput is higher than S2PL's all the time because SESAMO does not enforce global serializability and more transactions can complete before their deadlines.

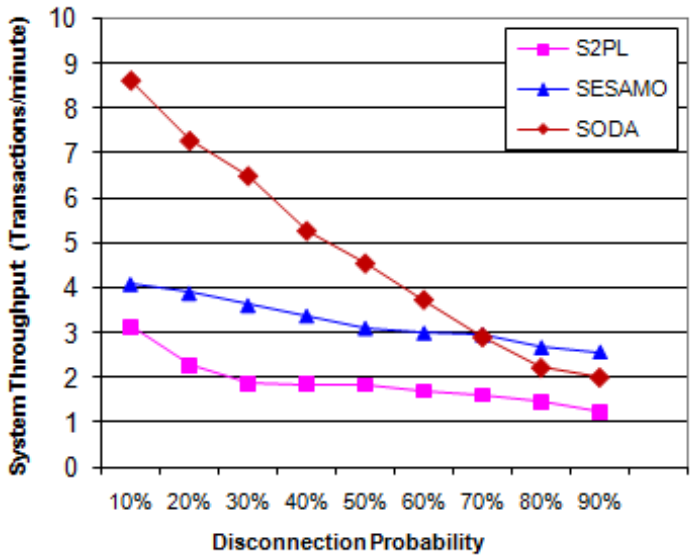

Figure 6.19 System throughput vs. disconnection probability

Figure 6.20 shows the average validation time that the primary cluster head spends on a global transaction of SODA increases as the disconnection probability increases. This happens because when the primary cluster head disconnects more frequently, it is often unavailable to validate transactions. However, the average validation time of S2PL and SESAMO is always zero because their designs do not involve any cluster head. In other words, S2PL and SESAMO do not prolong the response time due to the primary cluster head.

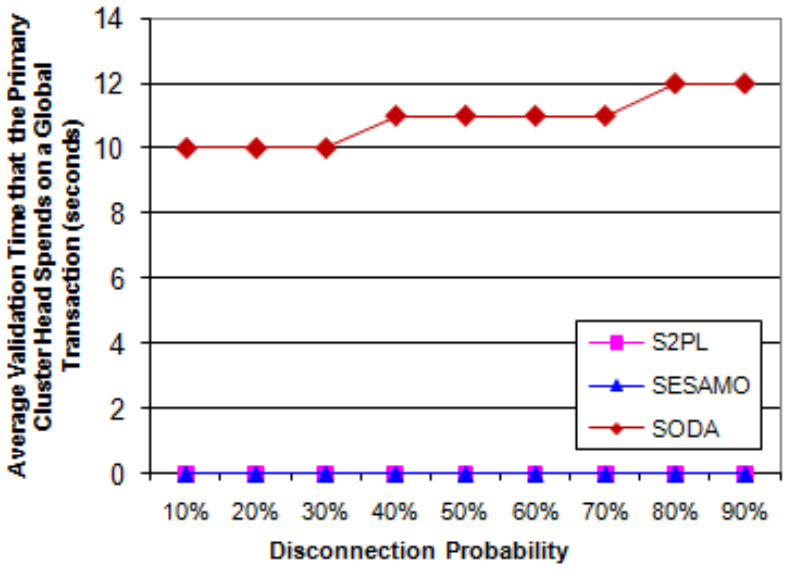

Figure 6.20 Average validation time that the primary cluster head spends on a global transaction vs. disconnection probability

As shown in Figure 6.21, the response time of S2PL, SESAMO and SODA roughly increases when the disconnection probability increases. This happens because servers are frequently disconnected and not available more often, so that transaction execution time is prolonged. S2PL has shorter response time than both SESAMO and SODA when the disconnection probability > 30%, but S2PL has higher abort rate than both SESAMO and SODA as shown in Figure 6.18. Again, the response time of SODA is higher than those of S2PL and SESMO because the primary cluster head is applied for validating global transactions as shown in Figure 6.20. However, the prolonged response time in SODA is reasonable (averagely 22 seconds longer) because it is still within the transaction deadline; otherwise, SODA should not have the lowest abort rate shown in Figure 6.18.

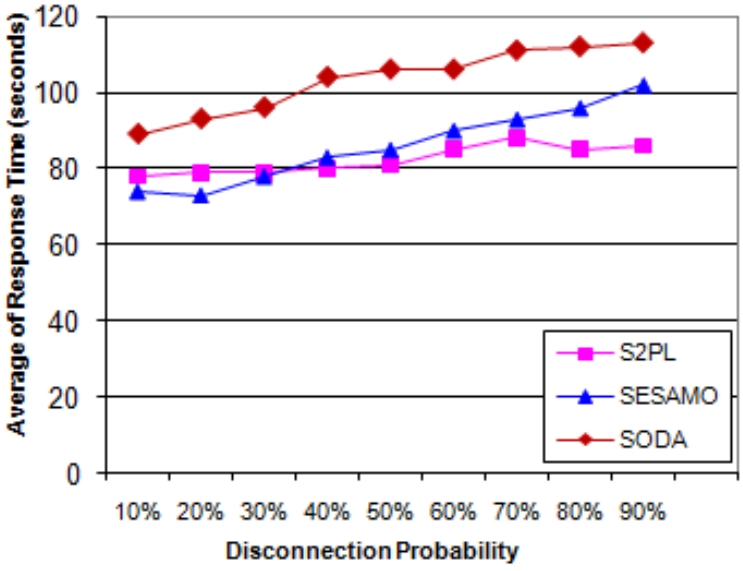

Figure 6.21 Average response time vs. disconnection probability

Figure 6.22 shows that the total number of cluster head reelections of SODA occurs only once when the disconnection probability is 50%. This is because the inter-arrival time is fixed with its default value 5 seconds when we study the effect of the disconnection probability, thus, the total simulation time is around 1.5 hours (1000 transactions * 5 seconds = 5,000 seconds). After running 1.5 hours, most cluster heads' remaining energy is not below the predefined threshold *LET* yet, therefore, only one reelection is triggered to change roles for preserving energy and balancing energy usage. However, the total number of reelections of S2PL and SESAMO is always zero because their designs do not involve any cluster heads. In other words, S2PL and SESAMO do not rotate roles among servers to balance energy.

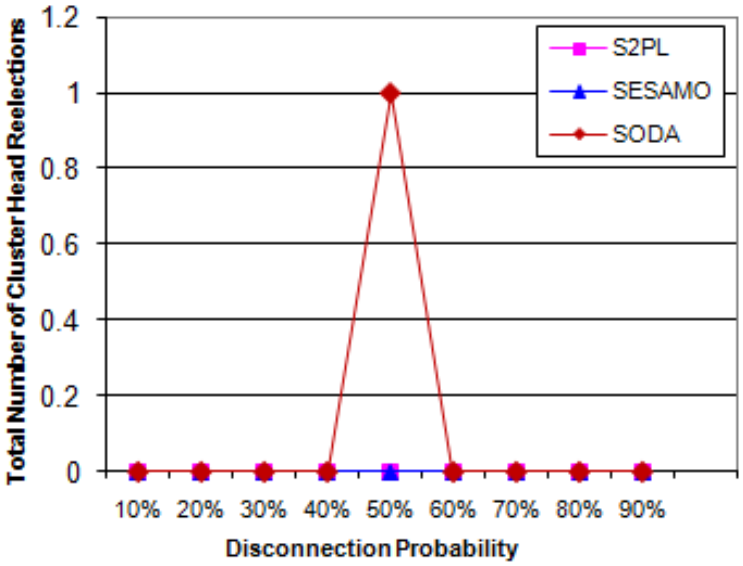

Figure 6.22 Total number of cluster head reelections vs. disconnection probability

In Figure 6.23, the total energy consumed by all servers of SODA slightly increases as the disconnection probability increases, but those of S2PL and SESAMO do not always increase or decrease as the disconnection probability increases. It is easy to observe that SESAMO has the highest total energy consumed by all servers, followed by S2PL, and SODA has the lowest energy consumption. This happens because S2PL and SESAMO utilize locks to hold limited system resources to prevent conflicting transactions from accessing them. In other words, even though servers are disconnected in S2PL and SESAMO, their data are still locked by some transactions, and these servers have to be in active mode to keep processing transactions. SODA consumes at least 115,368 J and at most 271,638 J less than both S2PL and SESAMO. This happens because SODA is optimistic and is not in active mode most of the time as shown in Figure 6.15.

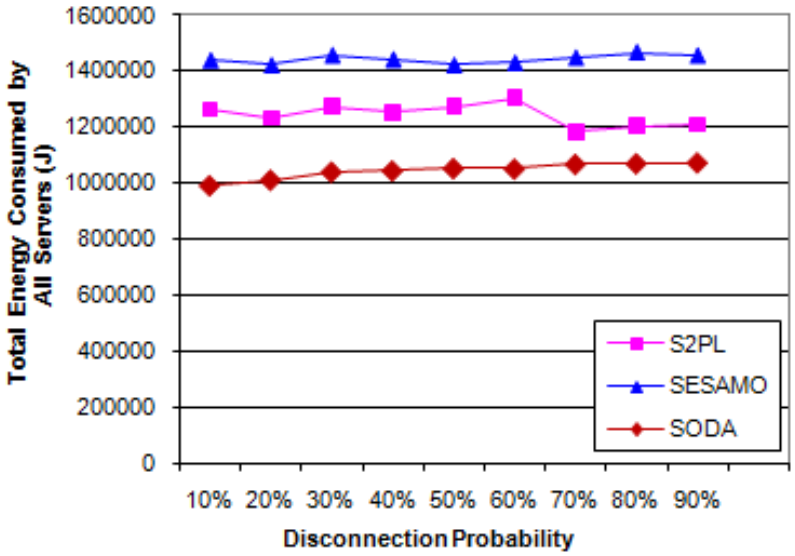

Figure 6.23 Total energy consumed by all servers vs. disconnection probability

In Figure 6.24, the average difference in remaining energy between two servers of S2PL roughly increases as the disconnection probability increases, but those of SESAMO and SODA goes up or down slightly as the disconnection probability increases. It is easy to see that SODA is the best to balance energy consumption, and S2PL does the worst except when the disconnection probability <= 20%. This is because SODA elects nodes with higher remaining energy and less workload to be cluster heads, and these cluster heads work as coordinating servers and will be reelected when their remaining energy is low. However, in S2PL and SESAMO, there is no clustering and role rotation strategy and clients may keep submitting transactions to the same servers so that these servers are overloaded.

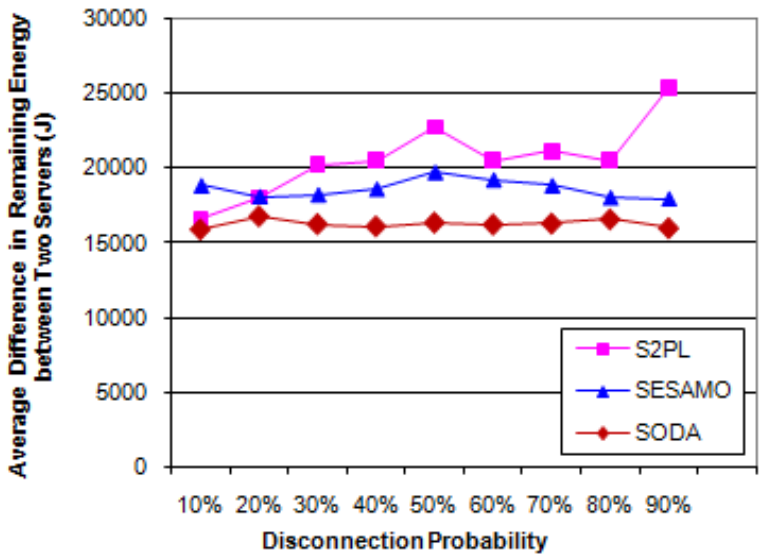

Figure 6.24 Average difference in remaining energy between two servers vs. disconnection probability

6.2.4 Effect of disconnection time

In this experiment, the disconnection time is varied to study the effect on the performance of the three algorithms because frequent disconnections are common in MANETs. The experiment results are shown in Figures 6.25-6.32.

Figure 6.25 shows that the total time when servers are in active mode of SODA slightly increases as the disconnection time increases, but those of S2PL and SESAMO goes up or down as the disconnection time increases. It is obvious that servers in SESAMO spend the longest total time in active mode on processing transactions, followed by S2PL, and SODA has the shortest total time. This happens because S2PL and SESAMO are pessimistic and utilize locks to hold common data to prevent conflicting transactions from accessing them. In other words, even though servers are disconnected in S2PL and SESAMO, their data are still locked by some transactions, and

these servers have to be in active mode to keep processing transactions.

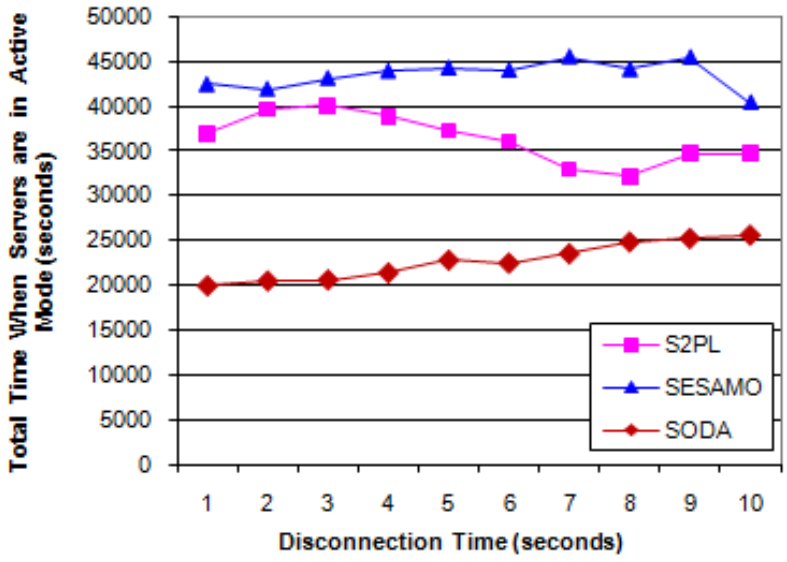

Figure 6.25 Total time when servers are in active mode vs. disconnection time

In Figure 6.26, the abort rates of S2PL, SESAMO and SODA increase when the disconnection time increases. This reflects the fact that fewer servers are available as servers are disconnected from the network longer and longer. The abort rate of SODA is much lower (at most 28%) than those of SESAMO and S2PL. This is mainly because S2PL and SESAMO utilize locks to prevent conflicting transactions from accessing common data and now servers are frequently disconnected, so that lots of transactions are aborted due to not able to access data and missing deadlines. SESAMO's abort rate is lower than S2PL's because SESAMO does not enforce global serializability.

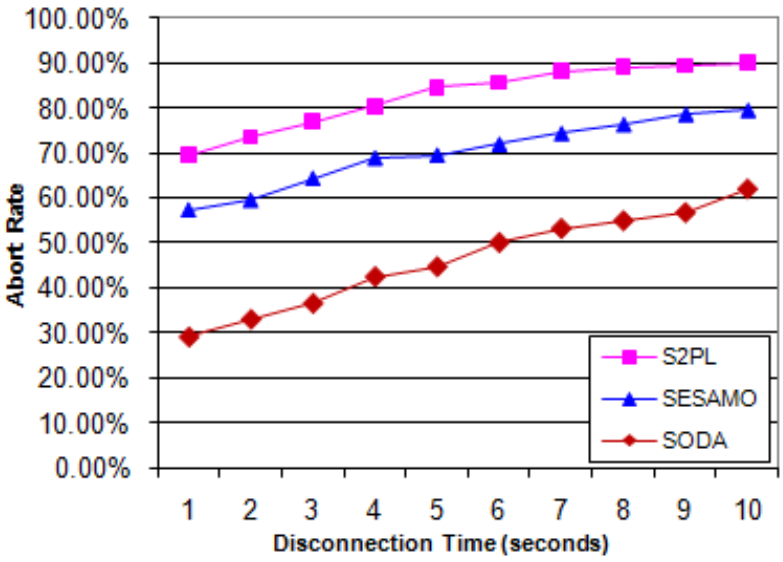

Figure 6.26 Abort rate vs. disconnection time

Figure 6.27 shows that the system throughput of these three algorithms decreases as the disconnection time increases. This happens because servers are frequently disconnected and are not available for a while to process transactions, so that lots of transactions are aborted because they missed their deadlines as shown in Figure 6.26. The system throughput of SODA is at least two more transactions/minute than that of SESAMO and S2PL. This is still mainly because SODA is optimistic and non-blocking, so that servers can process transactions in time. SESAMO's system throughput is higher than S2PL's all the time because SESAMO does not enforce global serializability and more transactions can complete before their deadlines.

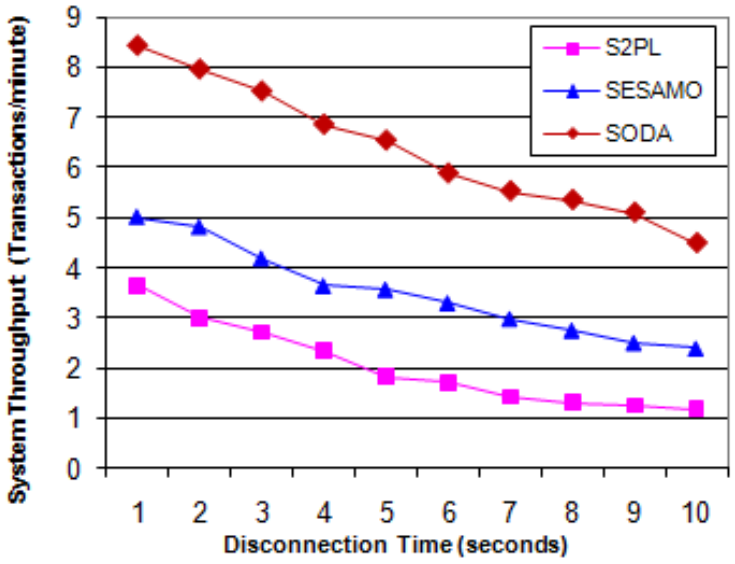

Figure 6.27 System throughput vs. disconnection time

Figure 6.28 shows SODA's average validation time that the primary cluster head spends on a global transaction increases as the disconnection time increases. This confirms that when the primary cluster's disconnection time gets longer, transactions have to wait longer for it to become available again before they can be validated. However, the average validation time of S2PL and SESAMO is always zero because their designs do not involve any cluster head. In other words, S2PL and SESAMO do not prolong the response time due to the primary cluster head.

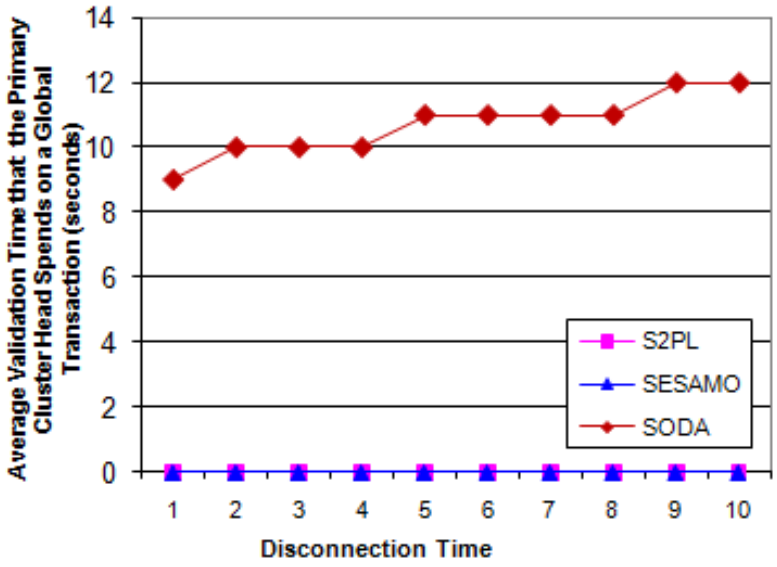

Figure 6.28 Average validation time that the primary cluster head spends on a global transaction vs. disconnection time

As shown in Figure 6.29, the response time of S2PL, SESAMO and SODA roughly increases when the disconnection time increases. This happens because when servers' disconnection time gets longer, transactions have to wait longer for servers to become available again before they can access their required data. S2PL has shorter response time than both SESAMO and SODA when the disconnection time > 5 seconds, but S2PL has higher abort rate than both SESAMO and SODA as shown in Figure 6.26. Again, the response time of SODA is higher than those of S2PL and SESMO, and one of the major causes of this is due to the primary cluster head as shown in Figure 6.28. However, the prolonged response time in SODA is reasonable (averagely 25 seconds longer) because it is still within the transaction deadline; otherwise, SODA would not have the lowest abort rate shown in Figure 6.26.

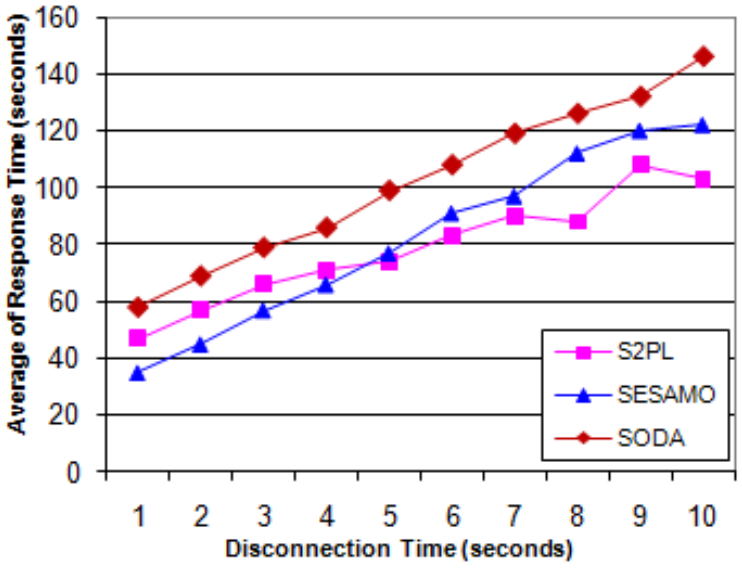

Figure 6.29 Average response time vs. disconnection time

Figure 6.30 shows the total number of cluster head reelections of SODA occurs only 4 times when the disconnection times are 5s, 8s, 9s and 10s. This is because the inter-arrival time is fixed with its default value 5 seconds when we study the effect of the disconnection time, thus, the total simulation time is around 1.5 hours (1000 transactions * 5 seconds = 5,000 seconds). After running 1.5 hours, most cluster heads' remaining energy is not below the predefined threshold *LET* yet, therefore, only 4 reelections are triggered to change roles for preserving energy and balancing energy usage. However, the total number of reelections of S2PL and SESAMO is always zero because their designs do not involve any cluster heads. In other words, S2PL and SESAMO do not rotate roles among servers to balance energy.

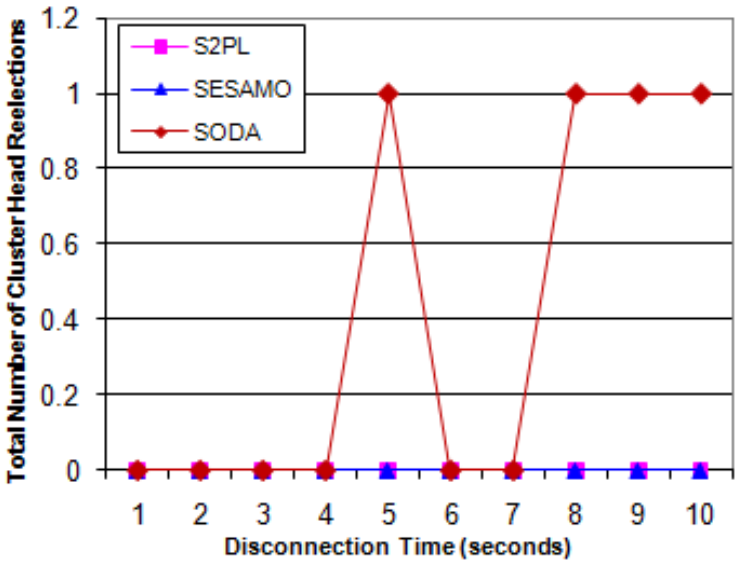

Figure 6.30 Total number of cluster head reelections vs. disconnection time

In Figure 6.31, it can be observed that the total energy consumed by all servers of SODA slightly increases as the disconnection time increases, but those of S2PL and SESAMO do not always increase or decrease as the disconnection time increases. SESAMO has the highest total energy consumed by all servers, followed by S2PL, and SODA has the lowest energy consumption. This happens because S2PL and SESAMO utilize locks to hold limited system resources to prevent conflicting transactions from accessing them. In other words, even though servers are disconnected in S2PL and SESAMO, their data are still locked by some transactions, and these servers have to be in active mode to keep processing transactions. SODA consumes at least 115,890 J and at most 299,643 J less than both S2PL and SESAMO. This is expected because the more time when servers are in active mode as shown in Figure 6.25, the more energy is consumed by all servers.

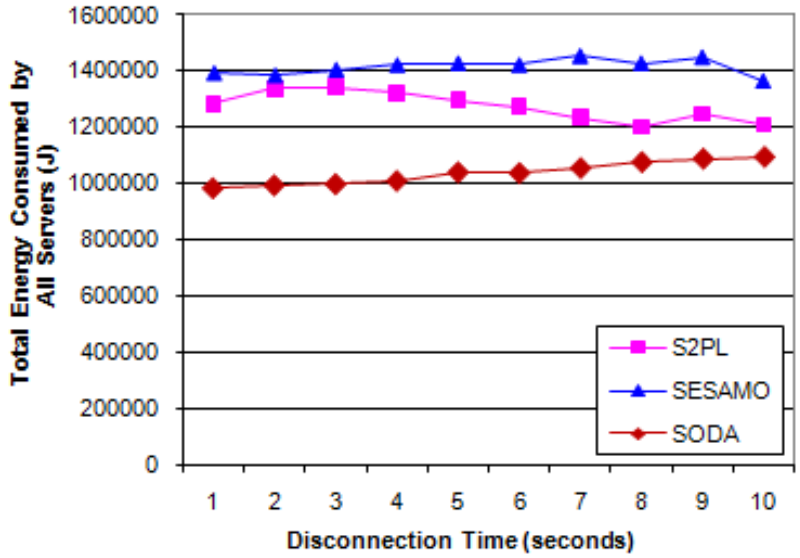

Figure 6.31 Total energy consumed by all servers vs. disconnection time

Figure 6.32 shows that the average difference in remaining energy between two servers of the three algorithms does not always increase or decrease as the disconnection time increases, and S2PL has a sudden increase at the disconnection time = 8 seconds and 9 seconds. It is easy to observe that SODA is the best to balance energy consumption, followed by SESAMO, and S2PL is the worst. This is expected because SODA elects nodes with higher remaining energy and less workload to be cluster heads, and these cluster heads work as coordinating servers and will be reelected when their remaining energy is low as shown in Figure 6.30. However, in S2PL and SESAMO, there is no clustering and role rotation strategy and clients may keep submitting transactions to the same servers so that these servers are overloaded.

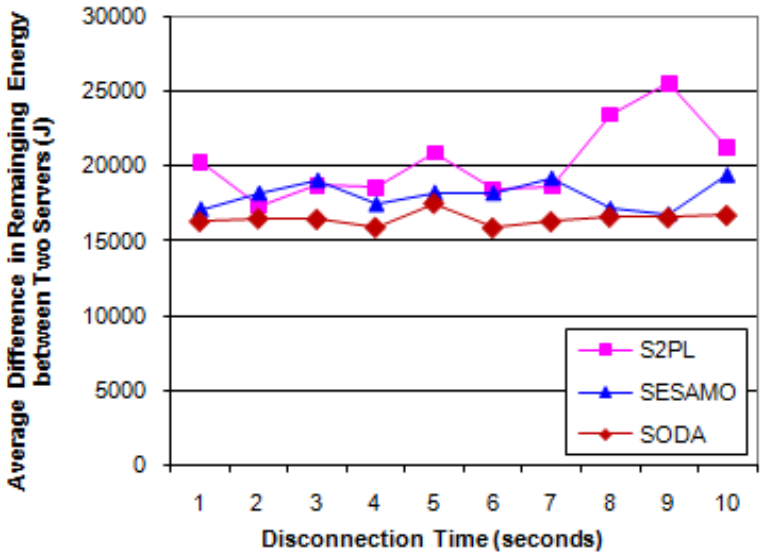

Figure 6.32 Average difference in remaining energy between two servers vs. disconnection time

6.2.5 Effect of node moving speed

The effect of the node mobility on the performances is studied in this section since every node can move freely in a MANET. Unlike other MANET characteristics, such as disconnection probability and disconnection time studied in Sections 6.2.3 and 6.2.4, respectively, the node moving speed has negligible effects on all seven performance metrics. In other words, regardless of the node moving speed, the performance metrics remain more or less the same. This is because even if all nodes move with the maximum moving speed 10 m/s, they move along with their groups (due to application semantics) and are limited within the area 1000*1000 meters^2, thus, the distances between nodes do not change significantly and the routine of transaction processing is not heavily impacted. The experiment results are shown in Figures 6.33 - 6.40 to confirm this observation.

In Figure 6.33 the total time when servers are in active mode of the three algorithms shows no significant changes when the node moving speed increases except for S2PL at the speed = 7 m/s. It is obvious that servers in SESAMO spend the longest total time in active mode on processing transactions, followed by S2PL, and SODA has the shortest total time. This happens because S2PL and SESAMO are pessimistic and utilize locks to hold limited system resources to prevent conflicting transactions from accessing them. In other words, even though servers are disconnected in S2PL and SESAMO, their data are still locked by some transactions, and these servers have to be in active mode to keep processing transactions.

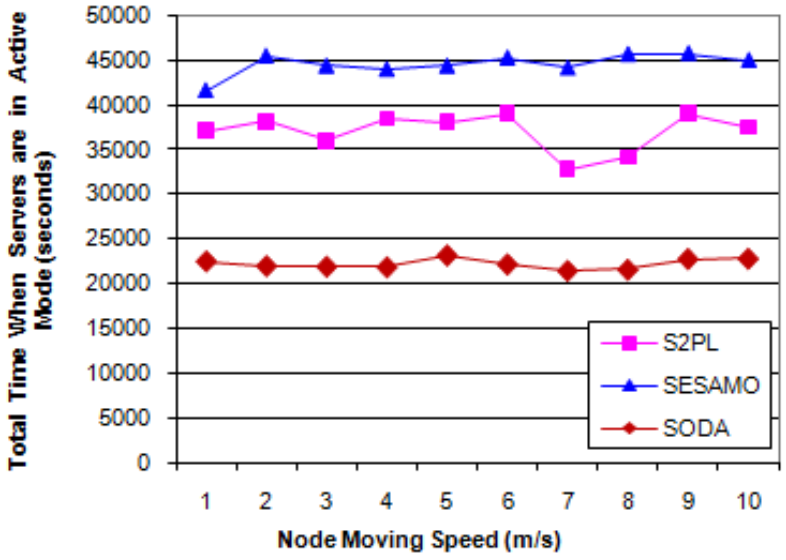

Figure 6.33 Total time when servers are in active mode vs. node moving speed

In Figure 6.34, the abort rates of S2PL, SESAMO and SODA show no significant changes when the node moving speed increases. The abort rate of SODA is much lower

(at most 27%) than those of SESAMO and S2PL. This is mainly because SODA is optimistic and non-blocking, and conflicts among transactions become rare, so that servers are not in active mode most of time as shown in Figure 6.33, and can process transactions in time. SESAMO's abort rate is lower than S2PL's because SESAMO does not enforce global serializability and more transactions can complete before their deadlines.

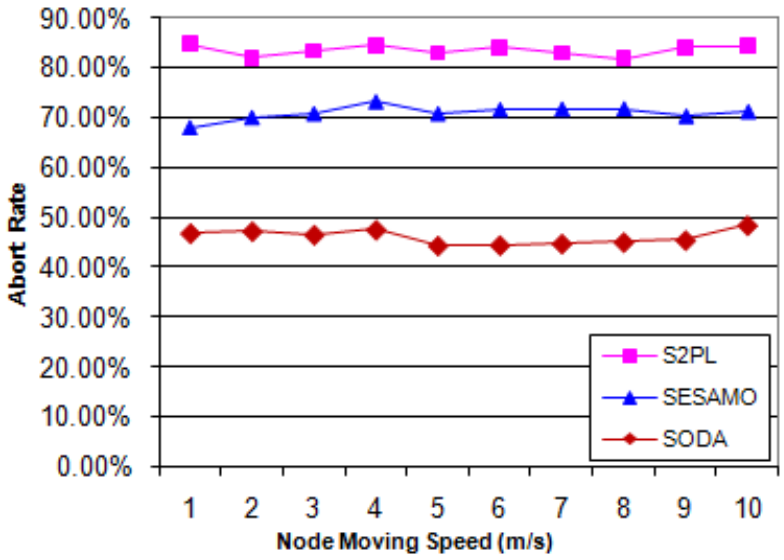

Figure 6.34 Abort rate vs. node moving speed

In Figure 6.35, the system throughput of the three algorithms shows no significant changes when the node moving speed increases, but it is easy to observe that the system throughput of SODA is at least 2 more transactions/minute than those of SESAMO and S2PL. This is still mainly because SODA is optimistic and non-blocking, so that servers can process transactions in time. SESAMO's system throughput is higher

than S2PL's all the time because SESAMO does not enforce global serializability and more transactions can complete before their deadlines.

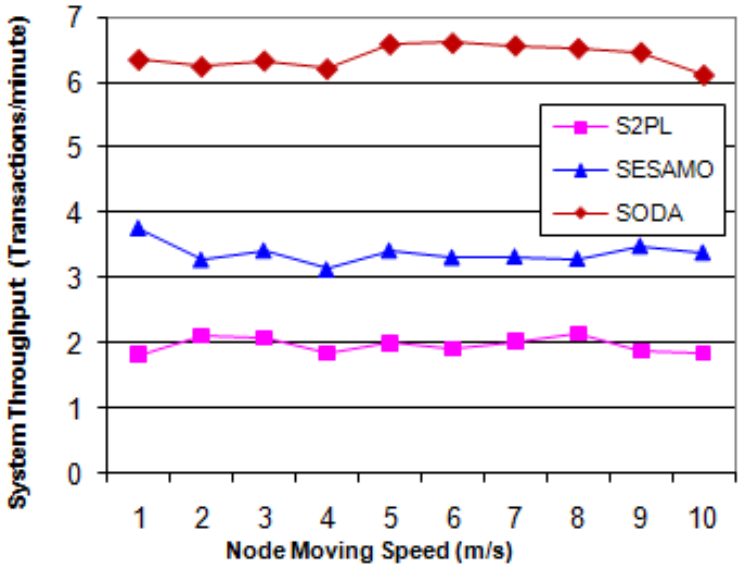

Figure 6.35 System throughput vs. node moving speed

In Figure 6.36, SODA's average validation time that the primary cluster head spends on a global transaction shows no significant changes as the node moving speed increases. However, the average validation time of S2PL and SESAMO is always zero because their designs do not involve any cluster head. In other words, S2PL and SESAMO do not prolong the response time due to the primary cluster head.

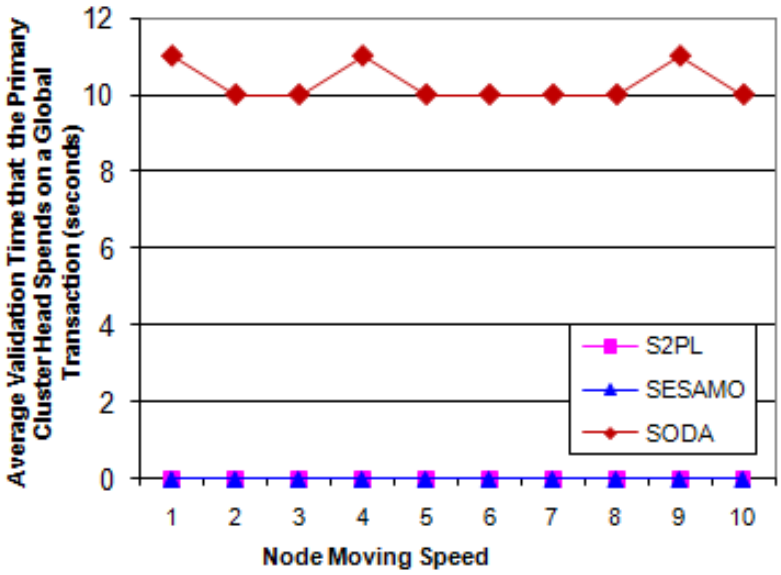

Figure 6.36 Average validation time that the primary cluster head spends on a global transaction vs. node moving speed

As shown in Figure 6.37, the response time of S2PL, SESAMO and SODA show no significant changes when the node moving speed increases. S2PL and SESAMO have shorter response time alternately, but S2PL has higher abort rate than both SESAMO and SODA as shown in Figure 6.34. Again, due to the bottleneck problem at the primary cluster as shown in Figure 6.36, the response time of SODA is higher than those of S2PL and SESMO. However, the prolonged response time in SODA is reasonable (averagely 19 seconds longer) because it is still within the transaction deadline; otherwise, SODA should not have the lowest abort rate shown in Figure 6.34.

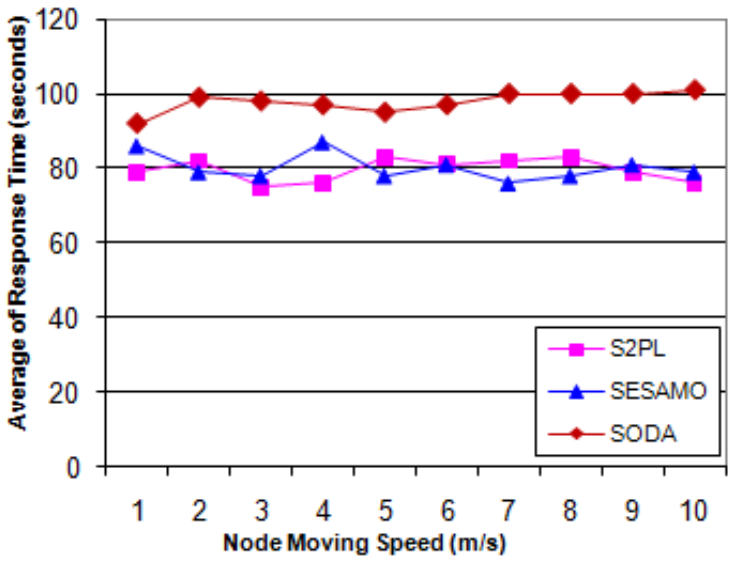

Figure 6.37 Average of response time vs. node moving speed

Figure 6.38 shows the total number of cluster head reelections of SODA occurs only once at 2 m/s as the node moving speed increases. This is because the inter-arrival time is fixed with its default value 5 seconds when we study the effect of the node moving speed, thus, the total simulation time is around 1.5 hours (1000 transactions * 5 seconds = 5,000 seconds). After running 1.5 hours, most cluster heads' remaining energy is not below the predefined threshold *LET* yet, therefore, only one reelection is triggered to change roles for preserving energy and balancing energy usage. However, the total number of reelections of S2PL and SESAMO is always zero because their designs do not involve any cluster heads. In other words, S2PL and SESAMO do not rotate roles among servers to balance energy.

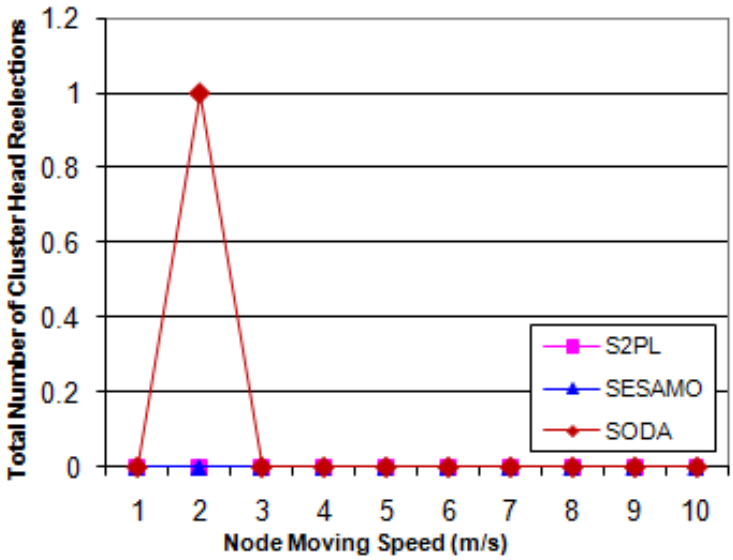

Figure 6.38 Total number of cluster head reelections vs. node moving speed

Figure 6.39 shows that the total energy consumption of all servers does not change significantly with the increase of the node moving speed, but it is easy to observe that SESAMO has the highest total energy consumed by all servers, followed by S2PL, and SODA has the lowest energy consumption. This happens because S2PL and SESAMO utilize locks to hold limited system resources to prevent conflicting transactions from accessing them and, consequently, servers have to be in active mode longer to process transactions as shown in Figure 6.33. SODA consumes at least 199,388 J and at most 296,734 J less than both S2PL and SESAMO. This is expected because SODA is optimistic and is not in active mode most of the time as shown in Figure 6.33.

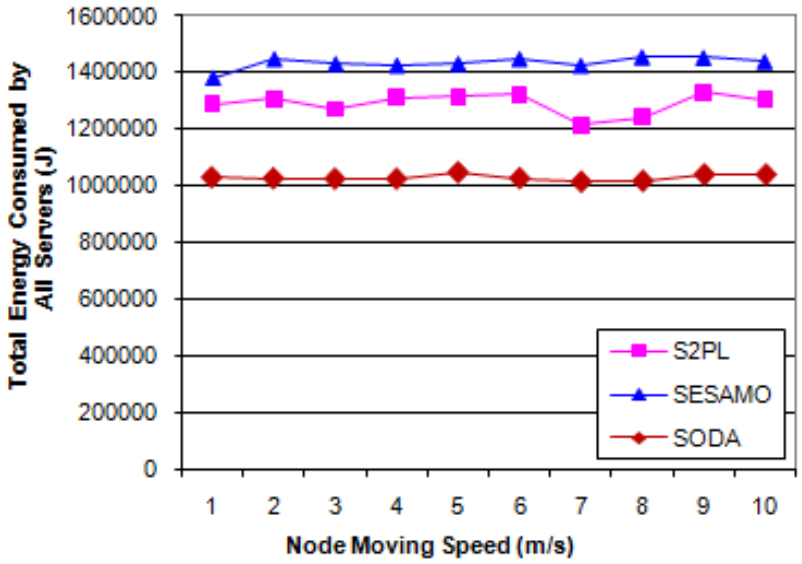

Figure 6.39 Total energy consumed by all servers vs. node moving speed

As shown in Figure 6.40, when varying the node moving speed, the average difference in remaining energy between two servers in the three algorithm does not change significantly except for S2PL at the speed = 2 m/s and 3 m/s. It is easy to observe that SODA is the best to balance energy consumption, followed by SESAMO, and S2PL is the worst. This is because SODA elects nodes with higher remaining energy and less workload to be cluster heads, and these cluster heads work as coordinating servers and will be reelected when their remaining energy is low as shown in Figure 6.38. However, in S2PL and SESAMO, there is no node clustering and role rotation strategy and clients may keep submitting transactions to the same servers so that these servers are overloaded.

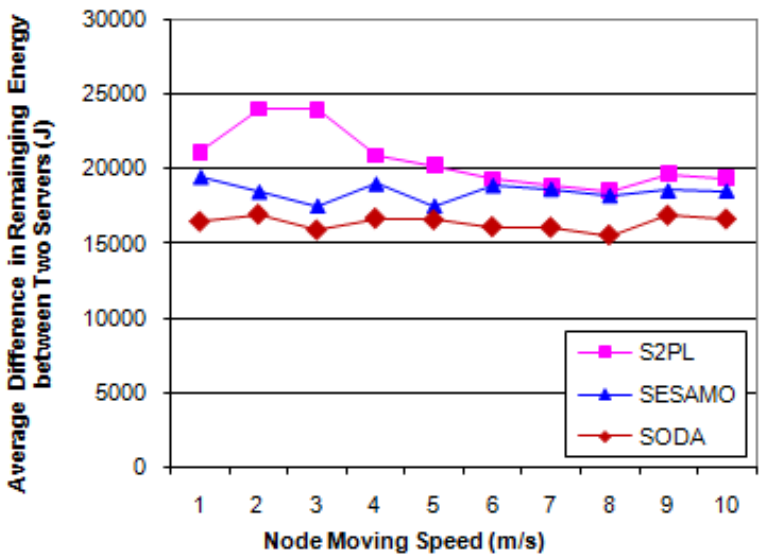

Figure 6.40 Total average difference in remaining energy between two servers vs. node moving speed

6.3 Conclusions

In this chapter, simulation experiments were conducted to compare the performance of our proposed SODA algorithm with those of two existing algorithms, SESAMO and S2PL, when varying the inter-arrival time, proportion of read-only transactions, disconnection probability, disconnection time and node moving speed. The simulation results show that SODA most of time performs better than both SESAMO and S2PL in terms of transaction abort rate, system throughput, total energy consumption by all servers, and average difference in remaining energy between two servers. However, SODA has to tradeoff transaction response time for these superiorities.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

7.1 Conclusions

In this dissertation, we proposed an energy-efficient CC algorithm, called Sequential Order with Dynamic Adjustment (SODA), for mission-critical MANET databases in a clustered network architecture. In this architecture, nodes are divided into clusters, each of which has a node, called cluster head, responsible for the processing of all nodes in the cluster. In SODA, in order to conserve energy and balance the energy consumption among servers so that the lifetime of the network is prolonged, we elected cluster heads using our weighted clustering algorithm MEW (Mobility, Energy, and Workload) to work as coordinating servers. SODA is based on optimistic CC to offer high concurrency and avoid unbounded blocking time. It utilizes the sequential order of committed transactions to simplify the validation process, and dynamically adjusts the sequential order of committed transactions to reduce transaction aborts. The simulation results show that MEW prolongs the lifetime of MANETs and has a lower cluster head change rate and re-affiliation rate than the existing algorithm MOBIC. The simulation results show the superiority of SODA over the existing techniques SESAMO and S2PL, in terms of transaction abort rate, system throughput, total energy consumption by all servers, and degree of balancing energy consumption among servers. However, SODA has to prolong transaction response time for these achievements.

7.2 Summary of Simulation Results

Based on the simulation results presented in Chapter 6, we observed the following trends:

- The abort rates of SODA, SESAMO and S2PL decrease as the transaction inter-arrival time and proportion of read-only transactions increase, but they increase with the increase of disconnection probability and disconnection time. The abort rates of these three algorithms have no significant changes when the node moving speed varies. S2PL has the highest abort rate, followed by SESAMO, and SODA has the lowest abort rate, i.e., S2PL > SESAMO > SODA.
- The system throughput of SODA, SESAMO and S2PL increases as the proportion of read-only transactions increases, but, it decreases with the increase of disconnection probability and disconnection time. The system throughput of these three algorithms follows no significant trend when the inter-arrival time and node moving speed vary. SODA has the highest throughput, followed by SESAMO, and S2PL has the lowest throughput, i.e., SODA > SESAMO > S2PL.
- The average response time of SODA, SESAMO and S2PL increases as the disconnection probability and disconnection time increase, but it follows no significant trend when the inter-arrival time, proportion of read-only transactions and node moving speed vary. The average response time of SODA is higher than that of S2PL and SESMO. However, the average response time of SODA is still within the reasonable range. In other words, SODA has higher average response time, but its transactions are not aborted due to missing deadlines, i.e., SODA >

SESAMO > S2PL.

- The total energy consumed by all servers of SODA, SESAMO and S2PL increases as the inter-arrival time increases, but it does not follow a significant trend when the proportion of read-only transactions, disconnection probability, disconnection time and node moving speed vary. SESAMO has the highest energy consumption, followed by S2PL, and SODA has the lowest energy consumption most of time, i.e., SESAMO > S2PL > SODA.
- The average difference in remaining energy between two servers of SODA, SESAMO and S2PL follows no significant trend with the increase of the inter-arrival time, proportion of read-only transactions, disconnection probability, disconnection time and node moving speed. Most of time, SODA has the lowest difference, followed by SESAMO, and S2PL has the highest difference, i.e., S2PL > SESAMO > SODA.

From the above observations, we conclude that SODA is the first choice if the prolonged transaction response time is not an issue; otherwise, S2PL is the best choice to get the results back in time regardless of high transaction abort rate. If applications do not require global serializability and transactions should be completed within short deadlines, SESAMO is the right choice.

7.3 Future Research

Since SODA has prolonged response time, a new version of SODA should be explored to shorten the response time. To guarantee global serializability, a primary

cluster head is elected to validate global transactions and, consequently, it becomes the bottleneck, thus, other solutions to enforce global serializability should be explored as well.

In the simulation of S2PL and SODA, 2PC is simply applied to guarantee transaction atomicity, but 2PC does not take MANET characteristics into consideration, thus, a new commit protocol should be investigated to overcome the drawbacks of 2PC.

In order to deal with network partition and improve data access time and data availability, a suitable data replication technique should be adopted into our simulation model.

REFERENCES

[Abdouli, 2005] M. Abdouli, L. Amanton, B. Sadeg, and A. Alimi, "A System Supporting Nested Transactions DRTDBSs," Proceedings of the 1st International High Performance Computing and Communications, 2005, pp 888-897.

[Alampalayam, 2009] S. P. Alampalayam and S. Srinivasan, "Intrusion Recovery Framework for Tactical Mobile Ad hoc Networks," International Journal of Computer Science and Network Security, Vol. 9, No. 9, 2009, pp. 1-10.

[Aschenbruck, 2008] N. Aschenbruck, E. Gerhards-Padilla, and P. Martini, "A Survey on Mobility Models for Performance Analysis in Tactical Mobile Networks," Journal of Telecommunications and Information Technology, Vol. 2, 2008, pp. 54-61.

[ATAC, 2005] http://www.atacwireless.com/adhoc.html. Last accessed – April 2011.

[Banerjee, 2001] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Mult-hop Wireless Networks," Proceedings of the 20th IEEE International Conference on Computer Communications, 2001, pp.1028-1037.

[Basagni, 1999] S. Basagni, "Distributed Clustering for Ad Hoc Networks," Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks, 1999, pp. 310-315.

[Basagni, 2006] S. Basagni, M. Mastrogiovanni, A. Panconesi, and C. Petrioli, "Localized Protocols for Ad Hoc Clustering and Backbone Formation: A Performance Comparison," IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 4, 2006, pp. 292-306.

[Basu, 2001] P. Basu, N. Khan, and T. D. C. Little, "A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks," Proceedings of the 21st International Conference on Distributed Computing Systems, 2001, pp. 413-418.

[Bernstein, 1987] P. Bernstein, V. Hadzilacos and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, MA, 1987.

[Berrabah, 2009] S. A. Berrabah, "GPS data correction using encoders and INS sensors," the 3rd International Workshop on Robotics for risky interventions and Environmental Surveillance-Maintenance, 2009.

[Bettstetter, 2002] C. Bettstetter and S. Konig, "On the Message and Time Complexity of a Distributed Mobility-Adaptive Clustering Algorithm in Wireless Ad Hoc Networks," Proceedings of the 4th European Wireless, 2002, pp.128-134.

[Brayner, 2005] A. Brayner and F. S. Alencar, "A Semantic-serializability Based Fully-Distributed Concurrency Control Mechanism for Mobile Multi-database Systems," Proceedings of the 16th International Workshop on Database and Expert Systems

Applications, 2005, pp. 1085-1089.

[Bruning, 2007] S. Bruning, J. Zapotoczky, P. Ibach, and V. Stantchev, "Cooperative Positioning with MagicMap," Workshop on Positioning, Navigation and Communication 2007, 2007, pp. 17-22.

[Catarci, 2008] T. Catarci, M. de Leoni, A. Marrella, M. Mecella, B. Salvatore, G. Vetere, S. Dustdar, L. Juszczyk, A. Manzoor, and H. Truong, "Pervasive Software Environments for Supporting Disaster Responses," IEEE Internet Computing, Vol. 12, No. 1, 2008, pp. 26-37.

[Chatterjee, 2002] M. Chatterjee, S. K. Das, and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks," Cluster Computing, Vol. 5, No. 2, 2002, pp. 193-204.

[Chlamtac, 2003] I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad Hoc Networking: Imperatives and challenges," Ad Hoc Networks Publication, Vol. 1, No. 1, 2003, pp. 13-64.

[Choi, 2006] H. Choi and B. Jeong, "A Timestamp-based Optimistic Concurrency Control for Handling Mobile Transactions," International Conference on Computational Science and its Application, Vol. 3981, 2006, pp. 796-805.

[Choi, 2009] M. Choi, W. Park, and Y. Kim, "Two-phase Mobile Transaction Validation in Wireless Broadcast Environments," Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, 2009, pp. 32-38.

[Denko, 2009] M.K. Denko, J. Tian, T. Nkwe, and M.S. Obaidat, "Cluster-Based Cross-Layer Design for Cooperative Caching in Mobile Ad Hoc Networks," IEEE Systems Journal, Vol. 3, No. 4, 2009, pp. 499-508.

[Dirckze, 2000] R. Dirckze and L. Gruenwald, "A pre-serialization transaction management technique for mobile multi-databases," ACM Mobile Networks and Applications, Vol. 5, No. 4, 2000, pp. 311-321.

[ER, 2005] I. ER, and W. Seah, "Clustering Overhead and Convergence Time Analysis of the Mobility-based Multi-hop Clustering Algorithm for Mobile Ad Hoc Netorks," Proceedings of the 11th International Conference on Parallel and Distributed System, 2005, pp. 1144–1155.

[Fei, 2008] Y. Fei, L. Zhong, and N. K. Jha, "An energy-aware framework for dynamic software management in mobile computing systems," ACM Transactions on Embedded Computing Systems, Vol. 7, No. 2, 2008, pp. 1-31.

[Fife, 2003] L. Fife and L. Gruenwald, "Research issues for Data Communication in Mobile Ad-Hoc Network Database Systems," ACM SIGMOD RECORD, Vol. 32, No.2, 2003, pp. 42-47.

[Georgakopoulos, 1991] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth, "On serializability of multidatabase transactions through forced local conflicts," Processing of the 7th International Conference on Data Engineering, 1991, pp. 314-323.

[Gruenwald, 2007] L. Gruenwald, S. M. Banik, and C. N. Lau, "Managing real-time database transactions in mobile ad-hoc networks," Distributed and Parallel Databases Journal, Vol. 22, No. 1, 2007, pp. 27-54.

[Guo, 2008] S. Guo and O. Yang, "Maximizing multicast communication lifetime in wireless mobile ad-hoc networks," IEEE Transactions on Vehicular Technology, Vol. 57, No. 4, 2008, pp. 2414–2425.

[Hofmann, 2006] P. Hofmann, K. Kuladinithi, A. Timm-Giel, C. Gorg, C. Bettstetter, F. Capman, and C. Toulsaly, "Are IEEE 802 Wireless Technologies Suited for Fire Fighters," the 12th European Wireless Conference 2006 - Enabling Technologies for Wireless Multimedia Communications, 2006.

[Holanda, 2008] M. Holanda, A. Brayner, and S. Fialho, "Introducing self-adaptability into transaction processing," Proceedings of the ACM Symposium on Applied Computing, 2008, pp. 992-997.

[HP, 2008] http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/HP_iPAQ_210_Enterprise_Handheld_Data_Sheet_02_08.pdf. Last accessed – April 2011.

[Hwang, 2000] S. Hwang, "On Optimistic Methods for Mobile Transactions," Journal of Information Science and Engineering, Vol. 16, 2000, pp. 535-554.

[Intel, 2008] http://www.intel.com/Assets/PDF/prodbrief/319982.pdf. Last accessed - April 2011.

[Kim, 2006] K. Kim, "A Novel Factor for Robust Clustering in Mobile Ad Hoc Networks," IEICE Transactions on Communications, 2006, pp. 1436-1439.

[Kung, 1981] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," ACM Transactions on Database Systems, Vol. 6, No. 2, 1981, pp. 213-226.

[Lam, 2005] K. Lam, C. S. Wong and W. Leung, "Using Look-ahead Protocol for Mobile Data Broadcast," Proceedings of the 3rd International Conference on Information Technology and Applications, 2005, pp. 342-345.

[Lee, 1993] J. Lee and S. H. Son, "Using Dynamic Adjustment of Serialization Order for Real-time Database Systems," Proceedings of 14th Real-Time Systems Symposium, 1993, pp. 66-75.

[Lee, 2002a] V. Lee, K. Lam, and S. H. Son, "Concurrency Control Using Timestamp Ordering in Broadcast Environments," The Computers Journal, Vol. 45, No.4, 2002, pp. 410-422.

[Lee, 2002b] V. Lee, K. Lam, S. H. Son, and E. Chan, "On Transaction Processing with Partial Validation and Timestamp Ordering in Mobile Broadcast Environments," IEEE Transactions on Computers, Vol. 51, No.10, 2002, pp. 1196-1211.

[Lei, 2008] X. Lei, Y. Zhao, S. Chen and X. Yuan, "Scheduling Real-Time Nested Transactions in Mobile Broadcast Environments," Proceedings of the 9th International Conference for Young Computer Scientists, 2008, pp. 1053-1058.

[Lei, 2009] X. Lei, Y. Zhao, S. Chen, and X. Yuan, "Concurrency control in mobile distributed real-time database systems," Journal of Parallel and Distributed Computing, Vol. 69, No. 10, pp. 866-876.

[Leu, 2007] Y. Leu, J. Hung, and M. Lin, "A new cache invalidation and searching policy for mobile ad hoc networks," Proceedings of the 2007 annual Conference on International Conference on Computer Engineering and Applications, 2007, pp. 337-343.

[Li, 2004] Y. Li, "A Caching Model in Managing Real-Time Transactions in Group-based Mobile Ad-Hoc Network (GMANET)," Master Thesis, University of Oklahoma, Norman, OK, 2004.

[Liu, 2005] J. Liu, F. Sailhan, D. Sacchetti, and V. Issarny, "Group Management for Mobile Ad Hoc Networks: Design, Implementation and Experiment," Proceedings of the 6th international conference on Mobile Data Management, 2005, pp. 192-199.

[Lu, 2007] W. Lu, W. K. G. Seah, E. W. C. Peh, and Y. Ge. "Communications Support for Disaster Recovery Operations using Hybrid Mobile Ad-Hoc Networks," Proceedings of the 32nd IEEE Conference on Local Computer Networks, 2007, pp. 763-770.

[Lu, 2008] X. Lu, Y. C. Chen, I. Leung, X. Zhang, and P. Lio, "A novel mobility model from a heterogeneous military manet trace," Proceedings of the 7th international conference on Ad-hoc, Mobile and Wireless Networks, 2008, pp. 463-474.

[Madria, 2007] S. K. Madria, V. Kumar, and S. Bhowmick, "A Transaction Model and Multiversion Concurrency Control for Mobile Database Systems," Distributed Parallel Databases, Vol. 22, No. 2, 2007, pp.165-196.

[Moiz, 2007] S. A. Moiz and L. Rajamani, "Single Lock Manager Approach for Achieving Concurrency Control in Mobile Environments," International Conference on High Performance Computing, Vol. 4873, 2007, pp. 650-660.

[Moiz, 2008] S. A. Moiz and Mo. K. Nizamuddin, "Concurrency Control without Locking in Mobile Environments," Proceedings of the 1st International Conference on Emerging Trends in Engineering and Technology, 2008, pp.1336-1339.

[Moss, 1985] J. E. B. Moss, Nested Transactions: An Approach to Reliable Distributed Computing, The MIT Press, Cambridge, MA, 1985.

[Notebookcheck, 2009] http://www.notebookcheck.net/Review-Lenovo-ThinkPad-

T400s-Notebook.21081.0.html. Last accessed - April 2011.

[Nouali, 2010] N. Nouali-Taboudjemat, F. Chehbour, and H. Drias, "On performance evaluation and design of atomic commit protocols for mobile transactions," Distributed and Parallel Databases, Vol. 27, No. 1, 2010, pp. 53-94.

[Obermeier, 2009] S. Obermeier, S. Böttcher, M. Hett, P. K. Chrysanthis, and G. Samaras, "Blocking reduction for distributed transaction processing within MANETs," Distributed and Parallel Databases Vol. 25, No. 3, 2009, pp. 165-192.

[Özsu, 1999] M. T. Özsu and P. Valduriez, Principles of Distributed Database Systems, 2nd edition, Prentice-Hall, 1999.

[Pabmanabhan, 2006] P. Pabmanabhan and L. Gruenwald, "DREAM: A Data Replication Technique for Real-Time Mobile Ad-hoc Network Databases," Proceedings of the 22nd International Conference on Data Engineering, 2006, pp. 134-137.

[Pritsker, 1999] A. Pritsker and J. O'Reilly, Simulation with Visual SLAM and AweSim, 2nd edition, New York: John Wiley & Sons, 1999.

[Serrano-Alvarado, 2004] P. Serrano-Alvarado, C. Roncancio, and M. Adiba, "A Survey of Mobile Transactions," Distributed and Parallel Databases, Vol. 16, No. 2, 2004, pp. 193-230.

[Sheu, 2006] P. Sheu and C. Wang, "A Stable Clustering Algorithm Based on Battery Power for Mobile Ad Hoc Networks," Tamkang Journal of Science and Engineering, Vol. 9, No. 3, 2006, pp. 233-242.

[Silberschatz, 2005] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database Systems Concepts, McGraw-Hill College, 2005.

[Sklavos, 2007] N. Sklavos and K. Touliou, "Power Consumption in Wireless Networks: Techniques & Optimizations," Proceedings of the IEEE Region 8 EUROCON 2007 International Conference on Computer as a Tool, 2007.

[StarTech, 2011] http://us.startech.com/product/ST1000BT32-10100-1000-Mbps-32-bit-PCI-Gigabit-Ethernet-Card. Last accessed - April 2011.

[Sucec, 2004] J. Sucec and I. Marsic, "Hierarchical Routing Overhead in Mobile Ad Hoc Netorks," IEEE Transactions on Mobile Computing, Vol. 3, No. 1, pp. 46-56.

[Viswacheda, 2007] D. V. Viswacheda, M. S. Arifianto and L. Barukang, "Architectural Infrastructural Issues of Mobile Ad hoc Network Communications for Mobile Telemedicine System," Proceedings of 4th International Conference on Sciences of Electronic, Technologies of Information and Telecommunications, 2007.

[Wang, 2007a] H. Wang, B. Crilly, W. Zhao, C. Autry, and S. Swank, "Implementing Mobile Ad hoc Networking (MANET) over Legacy Tactical Radio Links," Proceedings

of Military Communications Conference, 2007, pp. 1-7.

[Wang, 2007b] Y. Wang and M. S. Kim, “Bandwidth-adaptive Clustering for Mobile Ad Hoc Networks,” International Conference on Computer Communications and Networks, 2007, pp. 103-108.

[Xing, 2008] Z. Xing, L. Gruenwald, and K. K. Phang, "SODA: an Algorithm to Guarantee Correctness of Concurrent Transaction Execution in Mobile P2P Databases," Proceedings of the 19th International Conference on Database and Expert Systems Application Workshop, 2008, pp. 337-341.

[Xing, 2010] Z. Xing, L. Gruenwald, and K. K. Phang, “A Robust Clustering Algorithm for Mobile Ad-hoc Networks,” In a chapter on the Handbook of Research on Next Generation Mobile Networks and Ubiquitous Computing, ISBN: 160566250X, Editor Samuel Pierre, IGI Global, 2010, pp. 187-200.

[Xue, 2006] M. Xue, I. ER, and W. K. G. Seah, “Analysis of Clustering and Routing Overhead for Clustered Mobile Ad Hoc Networks,” Proceedings of the 26th IEEE international Conference on Distributed Computing Systems, 2006, pp. 46-53.

[Yu, 2005] J. Y. Yu and P. H. J. Chong, “A Survey of Clustering Schemes for Mobile Ad Hoc Networks,” IEEE Communications Survey & Tutorials, Vol. 7, No. 1, 2005, pp. 32-48.

[Zhang, 2010] X. Zhang, T. Kunz, L. Li, and O. Yang, “An Energy efficient Broadcast Protocol in MANETs,” Proceedings of the 8th Annual Communication Networks and Services Research Conference, 2010, pp. 199-206.