Commit protocols in mobile databases

Kyle Foss
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
fossx229@morris.umn.edu

ABSTRACT

A mobile database is a database that is installed on a mobile device, and a mobile device is able to communicate and share data over wireless networks with fixed devices and other mobile devices. Complications in database transactions occur, perturbations occur which can leave the data in an inconsistent state. In this paper, we explore three atomic perturbation-resistant commit protocols that address perturbations in three types of mobile database environments, one based on improving partition tolerance in mobile ad-hoc environments, another aimed at decreasing node and communication failures in infrastructure based environments, and one at improving fault tolerance in hybrid mobile environments. Simulation results indicate that each of these three commit protocols reduce inconsistency caused by perturbations.

Keywords

Mobile databases, Commit Protocols, Atomicity

1. INTRODUCTION

When a person uses their phone to buy an item from an online store, he or she expects money to be taken away from their bank account, and the item purchased to be shipped to their address. Except, it doesn't, and when the person calls the store, the store shows no record of the purchase ever being made, despite the charge on their bank account. This situation represents a common problem in distributed systems, know as the consensus problem. A distributed system consists of a network of entities such as computers. The consensus problem arises when agreement among a number *entities* or *processes* for a single value is needed, so a consensus protocol needs to be enacted. A type of distributed system is a distributed database, which is a database spread among multiple servers. The agents in this distributed database are transactions. "A transaction is a program unit whose execution preserves the consistency of database" [5]. To ensure consistency, different fragments of a distributed database must be in agreement when committing a transaction, so all actions associated with a transaction are executed to completion or none are performed. This property of a transaction is known as *atomicity* and can be referred to as an *atomic* action. [5]

UMM CSci Senior Seminar Conference, April 2018 Morris, MN.

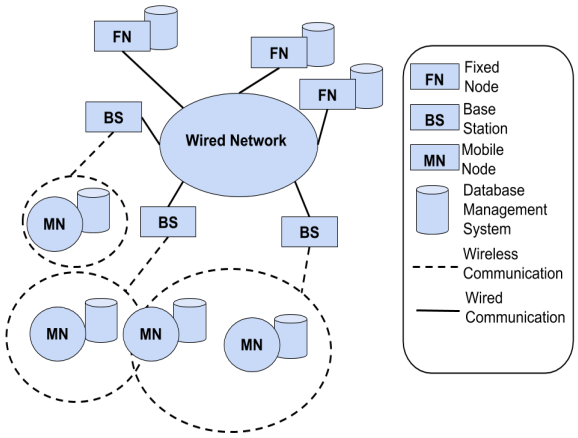


Figure 1: An overview of a mobile database system

In this paper, we present three atomic commit protocols for mobile database systems. In section 2, we introduce key terms and concepts relevant to parts of mobile database systems, transactions, and commit protocols. In section 3, we present key issues in mobile database systems. In section 4, we present a commit protocol that addresses issues in infrastructure based environments and examine its results. In section 5, we present a commit protocol that addresses issues in wireless ad hoc environments and examine its results. In section 6, we examine a commit protocol that addresses issues in hybrid mobile environments.

2. BACKGROUND

2.1 Mobile database system architecture

A Mobile database system shown in figure 1 [3], consists of several entities. One of these entities is a mobile device. Mobile devices are devices such as cellphones and laptops, and these mobile devices are known as *Mobile Nodes* (MNs) in a mobile database system. MNs have mobile databases installed on them. MNs are able to access the data stored on themselves through a *Database Management System* (DBMS). This is software that allows users to interact with the data stored on a database. MNs communicate with each other and other entities know as *Base Stations* (BSs) through wireless communication interfaces ranging including low and

high bandwidth forms of wireless communications. Since an MN is able to communicate with a BS, that MN can access entities know as *Fixed Nodes* (FNs), through a wired network, and each FN has a DBMS and a database installed on it. [3].

2.2 Transactions and Transaction Processing

Transactions represent real life events such as sales from a website and bank statements. For example, a transaction to transfer fifty dollars from a savings account to a checking account will perform two actions: change the savings account record by reducing the balance of the savings account, and change the balance of the checking account by adding fifty dollars. If the second step fails, and the first step succeeds, then the checking account won't have a correct balance. The transaction would end in an inconsistent state. Thus, data can unexpectedly change due to how entities in a transaction communicate, so the transaction needs to be monitored. This is known as *Transaction Processing*. Transaction processing is important for data integrity, specifically for the data in a database. Data integrity is the maintenance of, and the assurance of accuracy and consistency over the data's life cycle. [1]

2.3 Commit protocols

Commit protocols are used in transactions to ensure data consistency through atomicity. To ensure atomicity, every participant in the transaction will vote on whether or not to commit the transaction they're involved in. Every participant must agree complete the transaction. Therefore, the voting process must be unanimous. Every time a transaction is processed it will go through either a commit or a rollback procedure. If a participant votes to not commit a transaction, the transaction will be aborted, and data will be rolled back to a previous state. [5]

3. PERTURBATIONS

Problems that occur in mobile database systems are known as *commit perturbations*. Commit perturbations in mobile database systems are categorized into different categories: *environment constraints* and *failures*. Environment constraints are caused by *heterogeneity*, *unstable storage*, and *energy constraints*. Heterogeneity in a mobile database system occurs due to the variety of MNs inside a mobile database system. Not all MNs are guaranteed to have the same computing resources and ways of communication. MNs have unstable storage and power issues. A MN is prone to lose data stored on it because it is easy to damage or lose, so data stored cannot be recovered. Additionally, MNs can lose power due to battery life.

The other set of commit perturbations are know as failures. Failures are classified in two different ways: *node failures* and *communication failures*. Node failures occur in MNs and FNs. The type of failures are known as *transient failures* and *permanent failures*. Transient failures are temporary failures and occur due to software or hardware faults. Permanent failures are failures that are irreversible and occur due to theft or damage. *Communication failures* are classified into two categories: *network disconnections* and *message loss*. Network disconnections are transient. They can be predictable and unpredictable. If a MN, BS, or FN needs to be updated, it will have a scheduled time when it's offline, and it's therefore a predictable disconnection. An unpredictable occurs when a MN loses connection to the mobile database system. This can occur when a MN loses connection to a base station. The last category message loss occurs due to network congestion and collisions. This occurs when too many MNs are attempting to communicate with a base station.

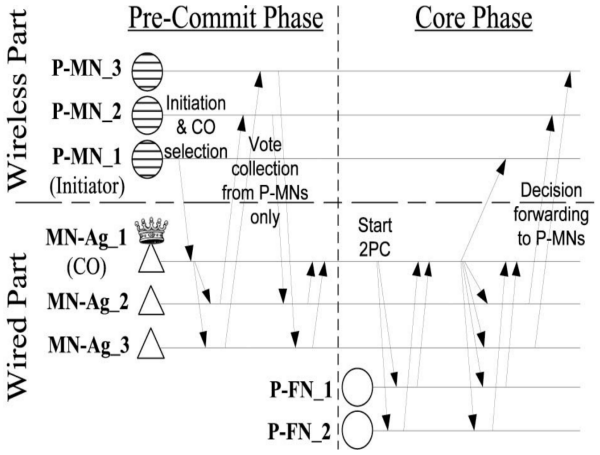


Figure 2: An overview of FT-PPTC

4. FT-PPTC

Fault-Tolerant Pre-Phase Transaction Commit (FT-PPTC) is used in *infrastructure based environments*, and these are environments where *participating mobile nodes* (P-MNs) are able to communicate with BSs but not other P-MNs. The objective of FT-PPTC is to separate P-MNs from *participating fixed nodes* (P-FNs) In FT-PPTC, P-MNs and P-FNs have various roles. One role of a P-MN is to start the transaction. This P-MN is know as a *Host Mobile Node* (H-MN). Each P-MN including the H-MN relies on a representative known as a *mobile node agent* (MN-Ag). A MN-Ag is a BS which is a P-FN, and a MN-Ag allows P-MNs to communicate with a wired network through wireless communication. A MN-Ag known as a Coordinator (CO) is responsible for monitoring the transaction. In order to seperate the P-MNs and P-FNs, the transaction is split into two phases: a *pre-commit phase* and *core phase*. The pre-commit phase serves to collect information from P-MNs, and the core phase only involves P-FNs and takes place in the wired network of the environment. The nodes in the wired network of the environment are fixed, so a *two-phase commit protocol* is used. [1]

4.1 Pre-commit Phase

To follow with the actions each node takes during the pre-commit phase refer to figure 2. [1]

Actions of the H-MN. The H-MN takes various actions during the pre-commit phase in a somewhat structured way. First, the H-MN will notify a MN-Ag and select it as a CO to start the transaction. Since the H-MN is a P-MN, it will record its data before updates and calculate two upper bounds for time-out values. The first upper bound calculates the amount of time it takes to record its data before any changes, the time to make those changes, and the amount of time, and the time to forward those changes to the CO. This

time-out value is known as *shipping time-out value*. The second upper bound is the amount of time it takes forward a P-MN's vote to commit or abort the transaction to the CO. This time-out value is known as the *execution time-out value*. Each of these upper bounds are used as time-outs for the transaction, and during the transaction the execution time-out value will be extended if needed. If the CO sends a message to abort the transaction, the H-MN will record the message locally and send an acknowledgement message to the CO, and the H-MN's changes will be rolled back based on its records. [1]

Actions of a P-MN. A P-MN will follow the same procedure as the H-MN replacing anything that's done with the CO with a MN-Ag. For information of the procedure see the start of section 4.1.

Actions of a MN-Ag. During the pre-commit phase, when receiving information from its corresponding P-MNs, the MN-Ag creates a record for each of its P-MNs and stores it in stable storage. A MN-Ag will then wait for any messages sent by the CO or P-MNs. If the message is sent by the CO, the MN-Ag will update its record with the received message and forward the message to all of its corresponding P-MNs. If the message is sent by a P-MN, one of two things happen. If the message contains updates, the record(s) of the MN-Ag will be updated to keep track of the new updates, and a vote to commit the transaction will be sent to the CO. If message doesn't contain updates, the record of the MN-Ag will be updated with the message, and the message will be sent to the CO. [1]

Actions of the CO. During the pre-commit phase, the CO will take various actions, and these actions will happen in somewhat arranged order. First, the CO will receive information from the H-MN, the CO creates a record for the transaction which will contain information about the transaction. Every participant involved in the transaction is stored in the record, and the record will contain a state for processing occurring at each participant to reflect the status of each participant in the transaction. The states are: idle, active, pre-committed, committed, and aborted. The state of the H-MN is set to active when it sends its time-out values, updates, and vote to the CO, and the state of each P-MN is set to idle. After this step, the actions and the order the CO executes these actions will depend on the other nodes in the transaction. [1]

The P-MNs will send their corresponding time-out values, votes, and updates to the CO through their MN-Ags. When receiving the time-out values from any MN-Ag, the CO will set the state of its self to being active. It will also set the the state of each P-MN to being active, when receiving its time-out values, updates and votes from the P-MNs. If new time-out values are calculated, the P-MNs will update the time-out values stored in the record(s) at the CO. The CO will then wait for the the maximum time-out value of execution time-out value and shipping time-out value combined. If the CO receives the updates from the H-MN, and a commit vote from each MN-Ag within the time-out, the CO writes all updates locally and sets the state of each P-MN to be pre-committed. During the pre-commit phase, the transaction can be aborted in two ways. First, if a CO receives a vote to abort the transaction, the CO will end the pre-commit phase by an abort decision and forward its decision to all P-MNs. The other way a transaction is aborted is through the expirations of the two time-out values. When a transaction is aborted, all P-MNs will rollback to a previous state based on their logs to maintain data consistency. If a CO receives all data logs and votes from each P-MN, then the transaction will proceed into the core phase. [1]

4.2 Core Phase

To follow with the actions each node takes during the core phase refer to figure 2. [1] If the pre-commit stage succeeds, the core phase begins, and the CO will execute the two-phase commit protocol, which involves the P-FNs of the transaction. (for more information about the two-phase commit protocol see [5]). When the core commit stage of the transaction begins, the CO sends the updates and votes of the P-MNs to their corresponding P-FNs, and the two-phase commit protocol is started. Votes are collected from each P-FN. During the two-phase commit protocol the transaction can be aborted in two ways. First, the transaction can time-out due to P-FN not sending its vote, or the transaction will be aborted due to the CO receiving a vote to abort from a P-FN. If the time-out values don't expire, and if the CO doesn't receive an abort vote, the transaction will succeed. In both cases, the CO will inform all participating nodes of the result. [1]

4.3 Results

To simulate a transaction in an infrastructure based environment Ayari et al., used a program called *SimJava*. [1] The simulation's parameters had a fixed number of P-FNs, a varied amount of MNs in the range of 1-25, a fixed processing time of 5ms for each P-MN to perform its actions. A fixed processing time of 2ms for each F-MN to perform its duties in two-phase commit, a fixed wireless communication delay of 10ms, and a fixed wired communication delay of 5ms, for information on these settings see [1].

The simulation was evaluated on two criteria: *throughput* and *resource blocking time*. "Throughput is the number of successful commit transaction per time unit, and resource blocking time is the amount of time a F-MN is accessed during the transaction excluding the CO and MN-Ags" [1].

FT-PPTC was simulated against two other commit protocols, two-phase commit and *mobile two-phase commit*. Results show that FT-PPTC has a higher throughput than two-phase commit but not mobile two-phase commit due to the amount of messages that are communicated in each protocol, for more information on these results see [1].

For resource blocking time, FT-PPTC has a lower blocking time than two-phase commit and mobile two-phase commit. This is due to the separation of the P-FNs and the P-MNs since resource blocking time occurs during the core phase. [1]

5. ParTAC

The goal of the *partition tolerant atomic commit protocol* (ParTAC) is to minimize and control the decision time of a transaction in a *wireless ad hoc environment*, which is an environment where MNs can only communicate with each other. It improves the decision time by tolerating message loss, network disconnections, and *network partitioning*, which is when a cluster of P-MNs is separated from another cluster of P-MNs that are communicating during the transaction and later reestablish communication during the transaction. ParTAC tolerates these perturbations by setting a lifetime time-out value for each transaction and by its

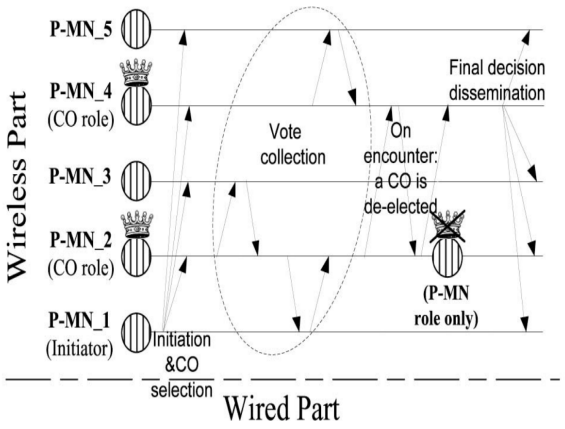


Figure 3: An overview of ParTAC

CO selection process. [2]

To follow along ParTAC's procedure refer to figure 3. [2] ParTAC consists of four parts. First, a P-MN initiates the transaction. Next, the P-MN that initiated the transaction will notify other P-MNs of the transaction, and a set of P-MNs is to become COs. The COs receive the transactions lifetime value, so each CO can abort the transaction when the lifetime of the transaction expires. Next, each CO will vote on whether or not to commit the transaction and add its own vote to the list of P-MNs that have voted to commit the transaction. The CO then collects a vote from each P-MN. If a CO encounters another CO during this process, then the COs will exchange their lists of votes and add each others list to their own. One of the COs will then change its role to being a P-MN and no longer be a CO. During the vote collection process the transaction can be aborted. If not aborted, a CO with a list of all P-MNs votes will notify all other P-MNs to commit the transaction. [2]

5.1 Roles of P-MNs

During the protocol, P-MNs make updates locally and are required to send their votes to each CO encountered provided no final decision has been made, and each P-MN is not allowed to change their vote during the transaction. The P-MN can either decide to vote to abort the transaction or vote to commit the transaction. If the P-MN decides to vote to abort the transaction, it sends a vote to abort to each CO encountered in the transaction. If the P-MN decides to commit, it will send a vote to commit each CO encountered in the transaction. The P-MN then will periodically communicate with the COs in the transaction by receiving a beacon that's sent by a CO. The beacon is used to inform P-MNs of the presence of COs in the transaction. When the beacon is received, the P-MN will respond back by sending its vote. The P-MN will repeat this process for every CO that has not been encountered in the transaction. During and towards the end of the transaction, the beacon is used to inform all P-MNs of the CO's decision to abort or commit the transaction. If a P-MN receives an abort or commit decision, it will send an acknowledgement message to notify a CO that it received its decision. The P-MN will then either commit its changes or rollback the changes. [2]

5.2 Roles of COs

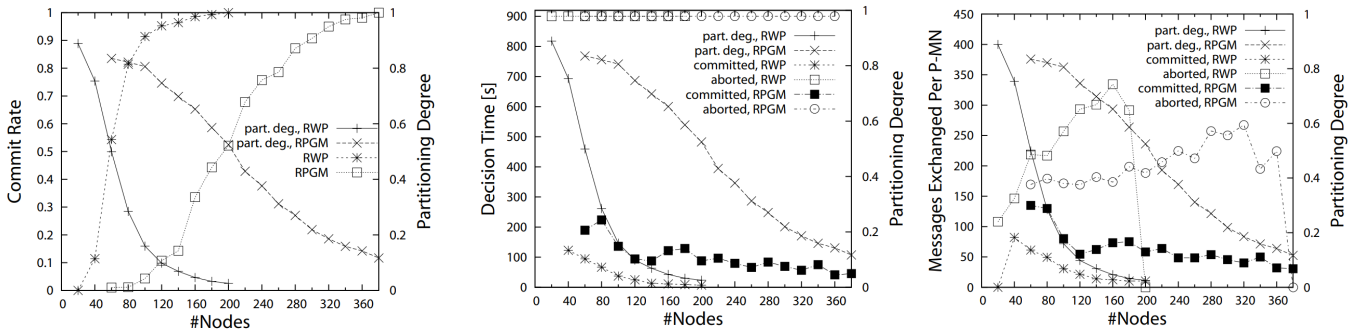
Table 1: Test settings for ParTAC

Parameter	values
Geographical area	$2km$ by $2km$
Communication range	$250m$
Mobility models	Random Waypoint (RWP), RPGM
Node speed	LOW uniform in [0.5, 1.5] m/s MEDIUM uniform in [3, 10] m/s HIGH uniform in [10, 25] m/s
Number of nodes	[20, 200] for Random Waypoint [60, 380] for RPGM
Number of COs	2, 3, 4, 7, 10
Number of P-MNs	10
lifetime	$60s$, $120s$, $300s$, $900s$

As described in section 5, a CO's main role is to oversee vote collection in the transaction. Each CO receives a copy of the transaction's lifetime value. Each CO then starts processing its updates and adds its vote its list. While waiting for time-out to expire, the CO will periodically send beacons to other P-MNs and COs involved in the transaction. The beacons allow the CO to communicate with other COs and P-MNs in the transaction. If a P-MN sends a message to the CO, it will either be a vote to commit the transaction or to abort it. If the message is to abort the transaction, the CO will send an abort message to all P-MNs. Alternatively, a CO can vote to abort the transaction since it is a P-MN. If the CO receives a vote to commit the transaction, the CO will add the ID of the P-MN to a list of P-MNs that voted to commit the transaction, and each CO keeps its own list. Each time an ID is added to a CO's list, a checklist algorithm is run to check if the CO's list contains the IDs of all P-MNs involved in the transaction. In the case that a CO encounters another CO in the transaction, one CO will be chosen to remain a CO as described in section 5. If any of the COs don't collect all votes before the time-out of the transaction expires, the CO will send an abort message to all P-MNs, and the transaction will be aborted. [2]

5.3 Testing ParTAC

The results of ParTAC are based on three criteria: *commit rate*, *transaction decision time*, and *message complexity*. [2] Commit rate is measured by the ratio of successful committed transactions to the total number of initiated transactions, and it measures service availability. Transaction decision time is the amount of time needed to complete a transaction. The message complexity is defined as the number of messages sent and received on average by each P-MN during the execution of the transaction. Message complexity determines the scalability of ParTAC. To simulate ParTAC Ayer et al., [2], used a common component-based, compositional based simulation environment called *J-Sim*. The program uses two mobility models: *random waypoint mobility* model (RWP) and *reference point group mobility model* (RPGM) to simulate the movement of nodes in wireless ad-hoc environment. In the RWP model, each node is given a random velocity and will pause at times to change the direction. In the RPGM model, nodes move in a group setting. Each group contains a leader node which the other nodes in the group follow. The velocity of the other nodes randomly deviates from the leader node. For test settings, Ayer et al [2]



(a) Impact of Mobility Model on Commit Rate

(b) Impact of Mobility Model on Decision Time

(c) Impact of Mobility Model on Message Complexity

Figure 4: ParTAC results

used, different movement speeds for the nodes. The movement speeds represented three different speed categories and moved uniformly. The number of COs and transaction lifetime values were also varied. For more information see table 1. [2]

5.4 Results

As mentioned before, ParTAC is simulated under different network conditions and protocol parameters to simulate the behavior of various development scenarios. Results indicate that the commit rate doesn't depend on the mobility model but on the amount of network partitioning occurring in the transaction, which was described in section 5. For more information on the results see figure 4a. [2]

The decision time and messages exchanged is higher in RPGM than Random Waypoint. In the RPGM nodes are concentrated in the same area, so as the number of MNs increases, greater message loss and network congestion occurs. For information on these results see figures 4b and 4c. [2]

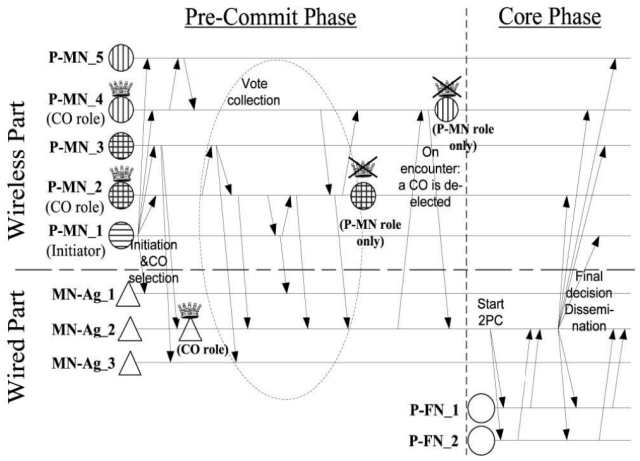


Figure 5: An overview of GMTC

6. GMTC

FT-PPTC and ParTAC are not optimal commit protocols for *hybrid mobile environments*, which is an environment where MNs can communicate with each other and BSs. FT-PPTC fails since it cannot support a mobile ad hoc environment since only MNs exist in this environment. ParTAC fails since it's very inefficient in this environment due to the amount of messages that are communicated due to COs, and the amount of time to complete a transaction. [4]

Generalized mobile transaction commit (GMTC) is a protocol that is combination of the ParTAC and FT-PPTC protocols. It is an atomic perturbation-resilient commit protocol for hybrid mobile environments. GMTC inherits the pre-commit phase and core phase from FT-PPTC and modifies the phases slightly. In the pre-commit phase, COs are selected the same way as in ParTAC and FT-PPTC as described in sections 4 and 5. However, the COs that are selected are P-MNs that have the ability to connect with infrastructure during the selection process. [4]

6.1 Pre-commit Phase

To follow with the actions each node takes during the precommit phase refer to figure 5. [4] **Actions of a P-MN.** The behavior of a P-MN in GMTC will behave similarly to their behavior in FT-PPTC and ParTAC as described in 4 and 5. However, a small difference exist between the two. When aborting or committing a transaction, a P-MN will send its vote to a MN-Ag instead of a CO if possible. When a P-MN doesn't have access to a MN-Ag, it will follow ParTAC's procedure for P-MNs. [4]

Actions of COs. In the precommit phase, COs manage the transaction based on vote and lifetime value of the transaction. COs will have the same behavior as described in sections 4 and 5. However, when two COs encounter one another, a CO that can connect with a MN-Ag will always remain a CO. If two COs cannot connect with a MN-Ag, CO resolution will remain the same as ParTAC as described in 5. Additionally, the MN-Ag that acts as a CO will follow this procedure and the checklist procedure as described in section 5. I[4]

Actions of MN-Ags In pre-commit phase, MN-Ags in GMTC will behave the same as described in section 4.1.

6.2 Core Phase

To follow with the actions each node takes during the core phase refer to figure 5. [4] To enter the core phase, as described in section 4.2 the CO in wired part of environment will begin the core phase based on the information communi-

Table 2: Test settings for GMTC

Parameter	values
Geographical area	$2km$ by $2km$
Communication range	$250m$
Mobility models	Random Waypoint (RWP)
Node speed	uniform in [0.5, 1.5] m/s
Number of nodes	[20, 200]
Number of Pre-selected COs	[3, 5, 10]
Number of P-MNs	10
Lifetime	$60s$, $120s$, $300s$

cated from the MN-Ags. P-FNs are apart of the core phase. During the core phase, P-FNs follow procedure as described in section 4.

6.3 Results

The results of GMTC are measured simulated based on the same criteria and the same test software as described section 5.3. However, some of the simulation settings were changed and are described in table 2.

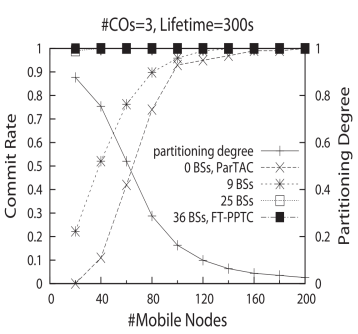
Results show that having BSs as a part of the transaction increases the commit rate, despite having a low number of P-MNs in the transaction when compared to ParTAC. This becomes less important as the number of P-MNs increase. When compared to FT-PPTC, there appears to be no difference between the two protocols in terms of commit rate. For information on these results see figure 6a [4].

For the transaction decision time, the decision time is lower for GTMC than ParTAC when the BS coverage is 44.17 percent. This is caused by having access to BSs, so P-MNs can communicate despite network partitioning. However, the decision time increases as number of MNs increase, and ParTAC will have a lower decision time in this case. This due to the number P-MNs that communicate with a corresponding BS. This same result occurs when comparing decision time in FT-PPTC to the decisions time in GMTC, but the difference between the decision times is less apparent. For information on these results see figure 6b [4].

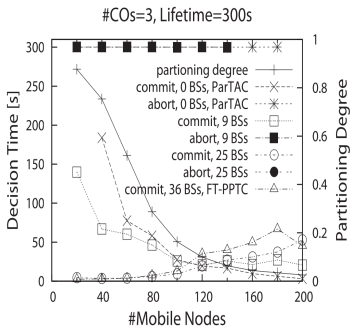
For message complexity, results indicate that amount of messages exchanged per P-MNs remained mostly constant across all three protocols. The only case when more messages were exchanged, was when transaction were aborted and the number of MNs was high. For information on these results see [4].

7. CONCLUSION

In the present paper, we discussed the importance of transactions in a mobile database system and presented three atomic commit protocols that address perturbations that occur in three mobile database environments: infrastructure environments, wireless ad hoc environments, and hybrid environments. We discussed the issues that arise in each, and we found that FT-PPTC and ParTAC improve the reliability and performance in infrastructure environments and wireless ad hoc environments respectively. However, in hybrid environments we found that GMTC is slower than FT-PPTC and ParTAC in some cases. As the number of mobile environments grow, there will be more commit protocols that will need to be developed and geared to handle pervasive perturbations in mobile database environments. It is important for us to address these perturbations so that



(a) Impact of BSs' coverage on commit rate



(b) Impact of BSs' coverage on decision time

Figure 6: GMTC results

our data doesn't change in unexpected ways.

8. ACKNOWLEGEMENTS

I would like to thank Kristin Lamberty, Elena Machkasova, and Thomas Hagen for help and feedback on my senior seminar paper.

9. REFERENCES

[1] B. Ayari, A. Khelil, and N. Suri. Ft-pptc: An efficient and fault-tolerant commit protocol for mobile environments. In *2006 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 96–105, Oct 2006.

[2] B. Ayari, A. Khelil, and N. Suri. Partac: A partition-tolerant atomic commit protocol for manets. In *2010 Eleventh International Conference on Mobile Data Management*, pages 135–144, May 2010.

[3] B. Ayari, A. Khelil, and N. Suri. On the design of perturbation-resilient atomic commit protocols for mobile transactions. *ACM Trans. Comput. Syst.*, 29(3):7:1–7:36, Aug. 2011.

[4] B. Ayari, A. Khelil, and N. Suri. Gmtc: A generalized commit approach for hybrid mobile environments. *IEEE Transactions on Mobile Computing*, 12(12):2399–2411, Dec 2013.

[5] C. Date. *An Introduction to Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 8 edition, 2003.