On the Design of Perturbation-Resilient Atomic Commit Protocols for Mobile Transactions

BRAHIM AYARI, ABDELMAJID KHELIL and NEERAJ SURI
TU Darmstadt, Germany

Distributed mobile transactions utilize commit protocols to achieve atomicity and consistent decisions. This is challenging as mobile environments are typically characterized by frequent perturbations such as network disconnections and node failures. On one hand environmental constraints on mobile participants and wireless links may increase the resource blocking time of fixed participants. On the other hand frequent node and link failures complicate the design of atomic commit protocols by increasing both the transaction abort rate and resource blocking time. Hence, the deployment of classical commit protocols (such as two-phase commit) does not reasonably extend to distributed infrastructure-based mobile environments driving the need for perturbation-resilient commit protocols.

In this paper, we comprehensively consider and classify the perturbations of the wireless infrastructure-based mobile environment according to their impact on the outcome of commit protocols and on the resource blocking times. For each identified perturbation class a commit solution is provided. Consolidating these sub-solutions, we develop a family of fault-tolerant atomic commit protocols that are tunable to meet the desired perturbation needs and provide minimized resource blocking times and optimized transaction commit rates. The framework is also evaluated using simulations and an actual testbed deployment.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed systems—*distributed applications, distributed databases*; H.2.4 [**Database Management**]: Systems—*transaction processing*

General Terms: Transaction design and reliability

Additional Key Words and Phrases: Mobile database systems, dependability, infrastructure-based wireless networks

1. INTRODUCTION

The pervasiveness and functionality of interacting mobile computing devices is increasing given the progress in wireless technologies. These mobile devices also interact with fixed devices/servers in realizing conventional applications such as e-mail, and also in enabling new applications such as mobile commerce (m-commerce), mobile inventory driving the need for mobile transactions.

For distributed systems, and especially distributed databases, a transaction is a set of operations that satisfies the following condition: either all operations of the transaction are completely performed or they have no effects. This all-or-nothing feature is known as the atomicity property. Commit protocols ensure atomicity and thus constitute a key issue in the execution of transactions. Commit protocols typically rely on a central entity (i.e., the coordinator) to take the final decision either to commit or abort the transaction. A transaction forms a logical unit of work, such as money transfer from one bank account to another which is composed of different operations but should be interpreted as a single operation. Transactions are also required for many mobile applications such as mobile auctions, mo-

Author's address: Brahim Ayari, Department of Computer Science, Technische Universität Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany.

bile inventory and mobile healthcare where data consistency is vital for the application and where multiple mobile devices act/operate as full transaction participants besides fixed ones. We depict two example scenarios to highlight the need for data consistency through atomic transactions in emerging mobile applications:

- Users connect increasingly often to the Internet through their mobile devices, browse online catalogs and buy goods from online shops. A sale transaction can involve bank and online shop servers in fixed wired networks in addition to the wireless devices. Atomicity guarantees in this scenario that *(1)* the customer gets the items and services he/she paid for, and *(2)* that the vendor is not reserving the goods for long periods of time waiting for the customer to pay for these goods. This might happen if atomicity is not supported and the customer declares his interest in buying some items and never appears to complete the initiated transaction.
- Several medical doctors use mobile devices to remotely access, monitor and update medical records of aged people living alone. Typically, the medical data is composed of health indicators and medicine prescriptions. Therefore, atomicity guarantees in this scenario the consistency of the data modified by the different doctors based on which the doctors are taking their decisions.

In the scenarios above, we assume a typical communication infrastructure where mobile devices are equipped with one or more wireless network interfaces to access a wired backbone. We refer to these established communication platforms as infrastructure-based mobile environments. These environments are characterized by a variety of perturbations (environmental constraints and failures) such as *(a)* the scarcity of processing and energy resources of mobile devices, *(b)* the continuously varying properties of wireless channels, which often lead to network disconnection, and *(c)* intermittent node failures. These perturbations make commit protocols that are designed for fixed networks (such as the traditional Two-Phase Commit (2PC) protocol [Gray 1978], Three-Phase Commit (3PC) protocol [Skeen and Stonebraker 1983] or the Paxos Commit protocol [Gray and Lamport 2006]) unsuitable for mobile environments. While 2PC is widely applicable in fixed networks because the cases in which the protocol blocks are rare, its applicability in mobile environments is limited as blocking is actually a part of the normal behavior of the system due to frequent failures. 3PC and Paxos Commit solve the blocking problem of 2PC by adding a considerable message overhead which is also not viable in mobile environments due to the considerably higher costs in terms of bandwidth, power consumption, and charges of using wireless links.

1.1 Our Contributions

The main contribution of this paper is the development of a comprehensive framework of perturbation-resilient techniques to cope with the set of identified perturbation in infrastructure-based mobile environments. Based on this framework, we develop a family of atomic commit protocols that provide perturbation-resilient transaction commit for emerging infrastructure-based mobile environments. We identify the perturbations in the investigated system model and comprehensively classify them into environmental constraints and failures. The comprehensive perturbation classification is intended to simplify the design of perturbation-resilient transaction commit protocols in mobile environments by supporting a modular and hierarchical approach. This classification also allows to systematically tackle the problem of atomic commit and present appropriate design techniques to

provide resilience to each of the identified perturbation classes without sacrificing performance. The main drivers of defining these new design techniques are *(1)* the minimization of transaction aborts in presence of perturbations and *(2)* the minimization of blocking time of critical resources especially of the fixed participants (e.g., bank servers) as they represent critical resources.

Different infrastructure-based mobile environments often entail varied commit perturbation levels. Naively, a solution that covers all perturbation classes would provide the desired perturbation-resilience in all considered environment classes. However, each resilience enhancement (perturbation counter-measure) usually leads to additional performance overheads. Therefore, we stress the modularity of our framework, which simplifies, at the design stage, the careful selection of the required building blocks implementing only the necessary fault-tolerance and recovery techniques. Such a preselection provides for sufficient perturbation-resilience while minimizing resilience-related performance drain. Overall, we combine the developed building blocks into a family of atomic transaction commit protocols for the most common infrastructure-based mobile environment classes.

Based on the classified perturbations our protocol family is composed of three protocols:

(1) Pre-Phase Transaction Commit (PPTC), a protocol designed to only cope with environmental constraints. Consequently, PPTC is suitable for infrastructure-based mobile environments where network and node failures are controllable.

(2) Fault-Tolerant PPTC (FT-PPTC), a protocol based on the fault-tolerant pre-phase transaction commit presented in [Ayari et al. 2006]. FT-PPTC implements tolerance to network failures in addition to resilience to environmental constraints. Accordingly, FT-PPTC is suitable for infrastructure-based mobile environments where only node failures are controllable but not network failures.

(3) Fault-Tolerant and Recovery PPTC (FT-PPTC-Rec), a protocol intended to tolerate all the identified perturbation classes. FT-PPTC-Rec represents a powerful solution for harsh infrastructure-based mobile environments with frequent and arbitrary commit perturbations.

The developed protocols are comparatively evaluated to highlight the functionality/performance tradeoffs of the different identified building blocks.

1.2 Paper Organization

The paper is organized as follows. Related work is discussed in Section 2. Section 3 describes the underlying system and perturbation models. Section 4 lists the major design requirements and issues of resilient atomic commit protocols for mobile transactions in the considered mobile environment. Based on these requirements, Section 5 describes the perturbation-resilient transaction commit framework. The family of commit protocols representing possible solutions for the identified problems are introduced along with their correctness proofs in Section 6. A comprehensive comparison to related work and evaluation of the different introduced protocols is described in Section 7. Section 8 concludes the paper and presents issues for future work.

2. RELATED WORK

Given the need for correct data management in infrastructure-based mobile environments, mobile transactions have increasingly become the focus of extensive ongoing research.

A variety of transaction models have been proposed such as [Alonso and Korth 1993; Chrysanthis 1993; Yeo and Zaslavsky 1994; Pitoura and Bhargava 1995; Walborn and Chrysanthis 1995; Dunham et al. 1997; Gray et al. 1998; Pitoura and Bhargava 1999; Walborn and Chrysanthis 1999; Ku and Kim 2000; Madria and Bhargava 2001] with an excellent survey appearing in [Serrano-Alvarado et al. 2004]. Some transaction models such as [Serrano-Alvarado et al. 2005; Nouali-Taboudjemat and Drias 2008; Karlsen 2003] propose adaptability to different mobile environments, constraints and applications by relaxing the traditional ACID properties (Atomicity, Consistency, Isolation and Durability) which leads to temporary inconsistent states. Some commit protocols have recently started addressing the problem of atomic commit in mobile infrastructure-based environments [Bobineau et al. 2000; Kumar et al. 2002; Serrano-Alvarado 2004; Nouali et al. 2005]. Other works have been conducted to address the atomic transaction commit problem in infrastructure-less mobile environments, i.e., mobile ad-hoc networks [Xie 2005; Böttcher et al. 2007; Obermeier et al. 2008].

In this work, we focus on transaction commit in infrastructure-based mobile environments such as for mobile commerce [Popovici and Alonso 2002] where ACID properties should not be relaxed. We consider a fully distributed scenario where the execution of mobile transactions is distributed among several mobile and fixed nodes. Only a few commit protocols have addressed this commit problem [Bobineau et al. 2000; Kumar et al. 2002; Serrano-Alvarado 2004; Nouali et al. 2005; Nouali-Taboudjemat et al. 2007; Bobineau et al. 2004].

The work on Unilateral Commit for Mobile (*UCM*) [Bobineau et al. 2000] provides support for disconnections and off-line executions on mobile devices. UCM is a one-phase protocol where the voting phase of 2PC [Gray 1978] is eliminated by enforcing some properties on the participant's behavior during the transaction execution. The elimination of the voting phase of 2PC results in reducing the wireless message complexity. UCM reduces the atomic commit protocol to a single phase that consists in broadcasting the coordinator's decision to all participants. In other words, the coordinator acts as a "dictator" imposing its decision on all participants. UCM guarantees atomicity. UCM has been specifically designed for mobile environments. To guarantee atomicity, transaction operations and their acknowledgments are continuously logged. If a problem arises the global transaction is immediately aborted. However, UCM is based on strict and hard assumptions such as local pessimistic concurrency control (strict two-phase locking [Bernstein et al. 1987]) which is required for all participants, as well as immediate integrity control and homogeneity of participating database systems (we refer to the 1PC assumptions provided in [Bobineau et al. 2000]). These assumptions restrict also the applicability of UCM to only a subset of the possible applications in mobile infrastructure-based environments. Similar to 2PC, an UCM coordinator blocks in the waiting state if at least one acknowledgement message is missing.

Transaction Commit On Timeout (*TCOT*) [Kumar et al. 2002] uses timeouts to provide a non-blocking protocol that limits the amount of communication between the participants in the execution of the protocol. Instead of exchanging messages to reach a Commit or Abort decision, the coordinator waits for timeouts to expire. In TCOT, if the coordinator node does not receive a failure message from a participant within a predefined timeout period, then it commits the transaction. While processing its transaction operations, if a participant finds out that its operations will execute longer than estimated, then it extends

its timeout and informs the coordinator. Overall, TCOT provides only semantic atomicity as defined in [Garcia-Molina 1983]. Semantic atomicity requires the existence of a compensating transaction for every initiated mobile transaction which is not possible for every transaction. Compensating transactions undo semantically the transaction effects. This type of atomicity is weaker than the strict atomicity [Haerder and Reuter 1994] needed for transactions in general. Semantic atomicity limits the applicability of TCOT to a narrow class of applications.

The CO2PC protocol [Serrano-Alvarado et al. 2004; Serrano-Alvarado 2004] combines an optimistic approach with 2PC. Like TCOT, the objective of the CO2PC protocol is to provide semantic atomicity for execution alternatives by allowing participants to perform either optimistic local commit (locally committed results are shared) or non-optimistic commit. Semantic atomicity limits the applicability of the protocol only for a restricted set of applications. The authors relax strict atomicity in this work (by guaranteeing only semantic atomicity) to increase the flexibility of participants (particularly mobile nodes). Hence, they distinguish between compensable transactions that are committed locally in an optimistic manner and non-compensable ones that have to wait for the global decision.

The objective of the Mobile Two-Phase Commit (*M-2PC*) protocol [Nouali et al. 2005] is to globally commit a distributed transaction in a mobile environment. M-2PC lets the mobile node delegate its commit duties to the coordinator which is assumed to be always available during the protocol execution. The mobile node sends the request for commit to the coordinator along with its logs. The mobile node can then disconnect. The coordinator sends vote request messages to all participants and decides on whether to commit or abort according to the classical 2PC semantics. Once the coordinator receives the acknowledgements of all participants, it informs the initiator about the result. The coordinator waits for the client acknowledgement before forgetting about the transaction (by releasing all resources acquired by the transaction). M-2PC assumes that all mobile participants are connected at transaction initiation and that network disconnections are allowed only after the mobile node delegates its commitment duties to the corresponding agent. Unfortunately, these assumptions are hard to fulfill in mobile environments, which leads to high abort rates when M-2PC is used. We will compare our protocol family with M-2PC, TCOT, CO2PC and UCM qualitatively in Section 7 and quantitatively whenever the protocol provides strict atomicity and implementation details are available to us.

As each evolving mobile environment necessitates new commit constraints, the current approaches geared towards dedicated scenarios, often do not provide comprehensive, generalized, and perturbation-resilient commit capabilities. Thus the need to develop generic and evolvable commit drives our research.

3. SYSTEM AND PERTURBATION MODELS

The perturbation-resilient atomic commit framework developed in this paper is based on the system and perturbation models presented in this section. The system model describes the different components of the considered infrastructure-based mobile environment. The perturbation model covers the various environmental constraints and failure modes that can be encountered in the stated system model. These models and especially the perturbation model are comprehensively taken into consideration in the design issues presented in the next sections.

3.1 System Model

We consider a generalized *infrastructure-based* mobile distributed environment consisting of sets of battery-powered mobile nodes (MNs) and fixed nodes (FNs). In our work, we do not consider extremely resource constrained devices such as wireless sensor motes and smart cards. The architecture of the mobile environment is illustrated in Fig. 1. MNs might be carried by vehicles, pedestrians etc. and intermittently connect to the wired network through base stations (BSs) via wireless channels. MNs can communicate with each other or with fixed entities only using the services provided by BSs. We refer to the set of MNs and FNs as $M = \{MN_1, \ldots, MN_m\}$ and $S = \{FN_1, \ldots, FN_s\}$ respectively, where m and s are the number of MNs and FNs respectively. The nodes usually entail varied hardware and software. In particular, MNs can range from cell phones with restricted storage and processing capabilities to laptops with considerably higher capabilities. For FNs, we do not place any restriction on the computation and storage capabilities and assume that all of them have a stable storage. Furthermore, MNs may use different wireless interfaces for communication ranging from low bandwidth and costly links (e.g., GPRS) to high bandwidth and free links (e.g., WLAN). Summarizing, we are dealing with highly *heterogeneous* nodes and links.

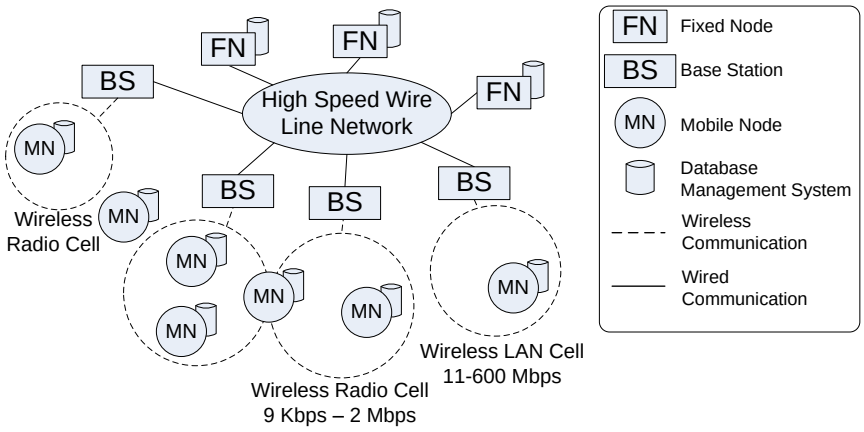

Fig. 1. Illustration of the considered mobile environment

We assume that each MN has a Mobile Database Server (MDBS) installed on it, and that a Database Server (DBS) is attached to each FN. Database servers are needed on both fixed and mobile nodes to support basic transaction operations such as Read, Write, Commit and Abort. The nodes are typically equipped with different *database management systems (DBMS)*. For instance mobile phones employ embedded DBMS such as Oracle Database Lite [Oracle Database Lite 2009] and IBM DB2 Everyplace [IBM DB2 Everyplace 2009] while laptops utilize standard DBMS such as Oracle Database Standard Edition [Oracle Database Standard Edition 2009] and IBM DB2 [IBM DB2 2009].

We consider applications, which run on either MNs or FNs and access data stored on both *mobile* and *fixed* nodes. Subsequently, a transaction can originate from any node in $M \cup S$, and the participants in its execution can be any set $P \subseteq M \cup S$. We focus on distributed transactions issued by either MNs or FNs and involving some FNs besides MNs

as participants. A distributed transaction where at least one MN participates in its execution is a *Mobile Transaction (MT)*. Commonly, a MT T_i is defined as a set of "execution fragments" distributed among a set of locations (also sites) in $M \cup S$ [Kumar and Dunham 1998; Kumar 2000]. However, this set of locations should contain at least one site in M. The participating node, where T_i is initiated, is termed as *Initiator*. The initiator node can be either a MN or a FN. If the initiator node is a MN, it is termed as *initiator MN (I-MN)*. The *commit set* consists of all FNs and MNs participating in execution and commit of T_i including the initiator node. FNs and MNs in the commit set are called *participant FNs (P-FNs)* and *participant MNs (P-MNs)* respectively.

We consider all distributed database system components (in $M \cup S$) to be *autonomous*, i.e., every component must take the decision to either commit or abort the transaction independently from other components in the network. Components are also able to decide which information to share with the global system and how to manage their local data. The data of the MN may be replicated on a backup database server in the fixed network. The synchronization of the data between the MN and its corresponding backup server is done periodically by the user.

We assume the existence of a *coordinator (CO)* in the fixed network (CO is a FN) responsible for coordinating the execution of the corresponding transaction. For different transactions, different nodes may play the CO role. The CO is responsible for storing information concerning the state of the transaction execution. Based on the information collected from the participants of the transaction, the CO takes the decision to commit or abort the transaction and informs all participants about its decision.

In this work, we assume the traditional asynchronous message passing communication between the different entities participating in the execution and coordination of the transaction. Asynchrony implies a lack of bounds on the time needed by a message to reach its destination or to be processed on a node.

3.2 Perturbation Model

Designing perturbation-resilient transaction commit protocols essentially requires the identification of relevant perturbations, i.e., environmental constraints and failure modes that can occur in the considered environment and disturb commit functionality. The following sections classify and enumerate these aspects.

3.2.1 *Environmental Constraints.* The considered mobile environment is constrained mainly by the characteristics of *MNs* and *wireless links*. MNs (ranging from laptops, personal digital assistants (PDAs) to cell phones) intuitively possess less computational resources than FNs, for instance processor speed and storage capacity. Especially, some MNs possess limited memory space which restricts the amount of data storable on them. These resource constraints increase the time MNs need to execute transaction fragments or may even lead to execution failures. Furthermore, MNs have no stable storage since they are carried by users, incur operational wear and tear and can also be easily lost or stolen. Additionally, data replication strategies typically used in the infrastructure-based mobile environment have limited capabilities. Most replication strategies proposed in the literature like [Pradhan et al. 1996] rely on BSs to replicate data of the MNs. BSs belong to a third party service provider and it is not evident that these providers want to contribute to achieve such goals. Another issue is the cost of storing the data on these BSs. Due to these issues the memory storage on a MN cannot be considered as stable.

MNs rely on finite amount of *energy* provided (e.g., by batteries) which implies that they can run out of energy anytime thus losing information about the status of execution stored on their volatile storages. Two of the most important sources of power consumption are transmissions and memory accesses.

Wireless network characteristics also change more frequently than those of wired links. For example, the effective available bandwidth is highly dynamic. This depends on the wireless technology (GSM, UMTS, WLAN, satellite, ...), access coverage, and number of MNs that have to share the wireless medium. Other key characteristics of the wireless links are latency and communication costs. These characteristics lead to considerably varied reliability/availability and connectivity of MNs.

The limitations and characteristics listed above outline the variety of environmental constraints for the mobile environment being different from those in fixed networks. These environmental constraints also complicate the design of efficient commit protocols.

3.2.2 *Failure Modes.* We now outline the relevant failure modes classified into the primary classes of communication and node failures.

3.2.2.1 *Communication Failures.* These constitute the majority of failures in the mobile environment. We distinguish between two types of communication failures:

a) Message loss: Messages exchanged between the MN and the BS are highly vulnerable to loss due to the high bit error rate of wireless links, network congestion and collisions. Message loss is much more probable to occur in mobile environments than in fixed networks and needs to be explicitly taken into consideration in the design of mobile systems.

b) Network disconnection (or link disruption): Given its mobile nature, a MN can enter a geographical area out of coverage of any BS so that it loses its connection to the network. The MN is said to be *disconnected* from the rest of the network. While disconnected from the network, the MN is not able to send or receive messages. As network disconnection is a common occurrence in mobile scenarios, it needs to be explicitly considered in the system design. Network disconnections can be either transient or permanent.

3.2.2.2 *Node Failures.* We distinguish between MN and FN failures. For MNs, we identify two main failure classes, i.e., *transient* and *permanent* failures. We do not consider malicious failures such as Byzantine faults or intrusions and leave them for future work.

a) Transient MN failures: These occur mainly due to either software or hardware faults and usually disappear if the MN reboots. A further cause of transient failures is the lack of battery power to sustain the operation of the mobile device. Transient failures are the most probable failures of MNs in the mobile environment. Opposite to network disconnection, in the case of a transient MN failure the content of the volatile storage of the MN and consequently the state of its recent computations is lost.

b) Permanent MN failures: These are irreparable failures such as loss, theft or physical damage of the MN itself or its non-volatile storage where the data and logs are stored (media failure). Consequently, all the data stored in the MN is lost. Although permanent MN failures are rather rare in mobile environments when compared to transient MN failures, this failure mode of MNs does occur because of the mobility and size of MNs.

c) FN and CO failures: We assume that if a FN (which can be a CO) crashes, then it stops receiving, sending and processing messages until it recovers after a finite but unbounded amount of time. This is known as the crash-recovery model.

4. DESIGN REQUIREMENTS AND ISSUES

Before presenting a family of commit protocols that tolerate the discussed perturbations, we present the design requirements on resilient atomic commit protocols for mobile transactions. A basic question is the need for new design requirements for atomic commit protocols in mobile environments. The difference between mobile and fixed environments is that perturbations in mobile environments are not the exception but often the normal case. Thus, we need to define the boundaries of our framework in terms of design requirements. We identify the following main requirements and design issues:

4.1 Fault-tolerance and recovery

To build resilient atomic commit protocols, it is essential to define a comprehensive categorization of perturbations and a set of techniques to cope with environmental constraints and recover from failures. The categorization of perturbations assists the protocol designer in identifying the main concerns and developing appropriate solutions. The overall objective for fault-tolerance (FT) is to maximize the commit rate. A naive approach to provide for fault-tolerance is to abort the MT each time a failure occurs and to restart it (e.g., after a back-off time or after the failure disappears). This simplistic approach introduces a large overhead for the successful participants (due to frequent re-execution of the fragments) and requires some external intelligence (either from the user or from the ability of the system to detect failures). Therefore, we introduce the delay-tolerance design requirement for MT.

4.2 Delay-tolerance

Masking latent faults such as long disconnections imposes that the MT execution time can be delayed till local Commit/Abort decisions can be collected. This implies that a MT can last for minutes or even hours. Thus, the interest is on developing *delay-tolerant transactions* where users can sacrifice latency for atomicity. For example, international bank transactions can last days due to heterogeneous processing and regulatory issues across the countries. We can easily expect that the application/user is able to specify an appropriate (tolerable) *lifetime* for each initiated MT. It is noteworthy that the lifetime is a timeout defined by the application to take a final Commit/Abort decision, and that final decisions actually are taken before the expiration of the lifetime. The selection of a longer lifetime should increase the probability to successfully execute the transaction without sacrificing the efficiency of the transactional service.

4.3 Efficiency

The efficiency of commit protocols is measured in terms of messages and resource blocking time. The classical approach to improve the efficiency of such protocols is to reduce the communication overhead (number and size of messages) and to minimize the resource blocking time. The rationale behind minimizing resource blocking time is that transactions, especially those executing on FNs, often lock expensive resources. Transactions are isolated from each other by locking all relevant data needed by them. As long as the locks are held, no other transaction can access the data, i.e., data or resources are *blocked*. The more transactions per time unit an application can process, the better the system's throughput. If

resources are blocked, transactions using them are delayed waiting for the resources to be unlocked resulting in reduced throughput. For this reason the blocking time, especially of FN resources as they are frequently much more loaded than MNs, should be minimized.

4.4 Scalability

Commit protocols are considered to be scalable if they can support growing numbers of participants without sacrificing efficiency. Resource blocking time as well as capabilities of the CO to handle more transactions per time unit are the main factors that determine the scalability of commit protocols.

The efficiency and scalability design requirements are orthogonal to the delay-tolerance requirement and imply a key challenge for the generalized commit framework targeted in this work.

5. PERTURBATION-RESILIENT TRANSACTION COMMIT FRAMEWORK: DESIGN CONCEPTS

Our primarily goal is to offer perturbation-resilience optimizing the performance of commit protocols in mobile environments. We focus on transparent FT-techniques which do not require any intervention from the user and mask the perturbations cited in our perturbation model. We also discuss different techniques to mask the environmental constraints in order to optimize the overall system performance and resilience.

5.1 Approach

We first classify the perturbations into a set of discrete classes to help simplify the solutions by supporting a modular and hierarchical approach. Next, we present FT design techniques relevant to each class driven by performance optimization requirements. As several contemporary efforts for fault-tolerant commit protocols exist, we review the existing strategies and maximize their reuse. Especially, we aim at reusing the results of the mature work existing for fixed networks.

5.2 Classification of Perturbations

We recognize the following two main classes of perturbations: environmental constraints and failures (Fig. 2). We classify the environmental constraints relevant to commit protocols into heterogeneity (of nodes and links), unstable storage and energy constraints. Failures of the mobile environment are classified into communication and node failures. Communication failures are due to message loss and network disconnections. We divide network disconnections into transient predictable, transient unpredictable and permanent. Examples of transient predictable network disconnections include the class of planned disconnections (like put-off or reboot) and situations in which the network coverage degradation is predictable such as when the wireless signal is continuously getting weaker. Another situation in which the MN can predict its disconnection is when the battery charge is low (disconnection due to node failure). Node failures are either MN or FN failures. MN failures are in turn either transient or permanent. Node failures might lead to network disconnections.

The comprehensive classification of perturbations in mobile environments helps simplify the identification of the main building blocks to guarantee perturbation-resilience for common classes of the considered mobile environment. These common mobile system

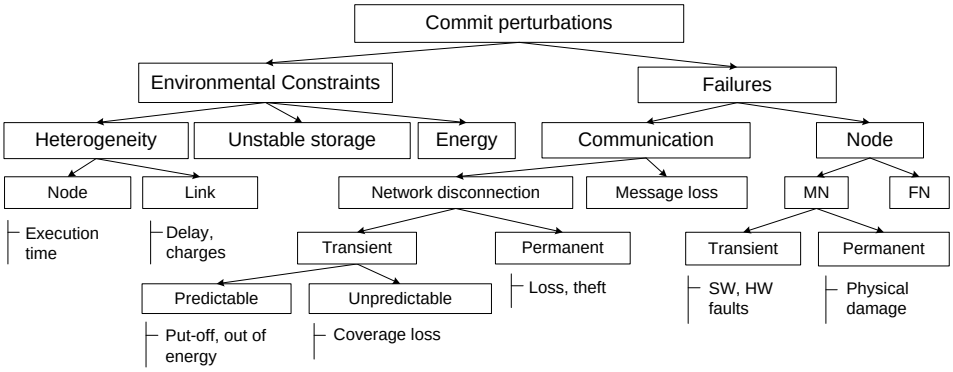

Fig. 2. Classification of perturbations

classes with their corresponding building blocks build the basis for the protocol family that we present in this work.

5.3 Coping with the Environmental Constraints

We begin with discussing the main techniques to provide resilience to the identified environmental constraints, i.e., node/link heterogeneity, unstable storage and energy.

5.3.1 *Heterogeneity of Nodes and Links.* We start from a scenario where only homogeneous fixed nodes communicate through high speed wired networks of homogeneous links using standard commit protocols like 2PC or 3PC. Finding a timeout (TO) value after which the CO can abort the transaction if it does not receive the votes of all participants is straightforward since the costs involved with restarting the transaction are not high. This timeout value should only mask the slightly oscillating execution times and communication delays due to varying node and network loads. These timeout values do not usually exceed the range of a few seconds.

Next, we consider the same scenario but with heterogeneous nodes, i.e., the participants can have different capabilities for CPU processing, memory, etc. Finding an appropriate timeout value now becomes challenging since the time needed to execute different fragments of a transaction can vary considerably from one participating node to another. As usually restarting the transaction with a higher timeout value is not costly in wired static environments, this would be a suitable decision. It becomes challenging when some of the heterogeneous devices participating in the transaction are mobile and use not only high speed networks but also, like in our system model, costly wireless communication links. Subsequently, setting an appropriate timeout value by the CO in this new scenario can reduce the costs of restarting the transaction. Also modifying the commit protocol to use less messages can further reduce these costs and save the limited bandwidth.

In [Kumar et al. 2002], the authors presented a timeout based approach to deal with node and link heterogeneity. Each P-MN computes an *execution timeout (E_t)* - an estimated upper bound for the time to complete the execution of its transaction fragment - and a *shipping timeout (S_t)* - an estimated upper bound for the time to compose updates and to send them to the CO. The P-MN sends both timeouts to the CO. Both timeouts have to account for the environmental constraints related to the P-MN and the wireless link it uses.

These timeouts can be extended if needed. The CO of the MT sets its timeout according to the time needed by the participants to execute their fragments and to send their votes to either commit or abort the transaction. Details on how this timeout is set are not provided in [Kumar et al. 2002]. However, this approach will always abort the MT if one of the participants is considerably slower than the initiator as illustrated in Fig. 3. This figure shows that in case one of the participants is considerably slower than the initiator, the timeouts of the slow participant are received after the expiration of the CO timeout defined in this case by the initiator, i.e., after the abortion of the MT. Therefore, we extend this approach to an advanced timeout handling as follows.

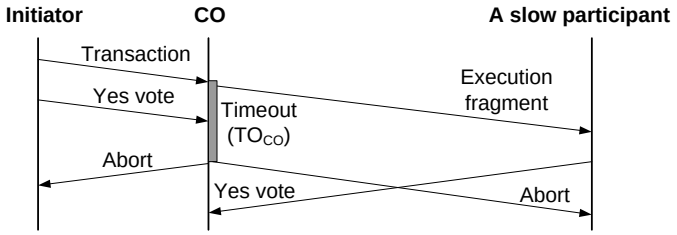

Fig. 3. Timeout selection in a heterogeneous scenario

We let the initiator inform the CO about the estimated/desired *lifetime* of the MT. The lifetime of a transaction provides the maximal timeout the CO should wait to take the decision about the outcome of the MT. It takes into consideration the environmental constraints including heterogeneity of MNs and wireless links. The lifetime can either be set by the application or estimated by the initiator or CO based on previous experiences and observations. As usually P-MNs are the slowest among all participating nodes and as we want to maximize the reuse of existing commit protocols in the wired static network, the CO requests only from P-MNs their estimated timeouts $TO_{P\text{-}MN_i}$. If the CO receives a lifetime from the initiator, it sets its own timeout $TO_{CO} = lifetime$. If the lifetime is undefined for any reason, the CO updates its own timeout every time it receives a timeout estimation from one P-MN by setting it to the maximum of all received $TO_{P\text{-}MN_i}$ ($TO_{CO} = \max\{TO_{P\text{-}MN_i}\}$). If TO_{CO} expires before receiving a timeout from one P-MN, the CO aborts the MT. In this case the CO additionally needs to identify the slowest P-MN and to estimate the time needed to receive its timeouts (Fig. 3). We present a detailed discussion of the scenario where the lifetime is undefined in Section 5.4.

As described above, for heterogeneity of nodes and links, an enhanced timeout handling is proposed. The two remaining environmental constraints, i.e. unstable storage and energy impact mainly the selection of the CO and will be used in the following to justify the choice of the CO as a FN.

5.3.2 *Unstable Storage.* We consider the constraint of unstable storage of MNs and derive its impact on the design of commit protocols. As mentioned before in Section 3.2, the storage of MNs is usually not stable, making them even less suitable for a CO role since it is a key element of atomic commit protocols. Due to its unstable storage, if a MN plays the role of a CO, it can lose all the information related to a transaction and consequently will not be able to take a decision. This is similar to a permanent crash of the CO. Furthermore, if the CO runs out of memory and is not able to store required

information about the state of the transaction, it has to abort the transaction even after receiving "Yes" votes from all participants.

5.3.3 *Energy.* The CO sends and receives relatively high number of messages (compared to other participants) in order to take a decision on the outcome of the transaction and also to ensure that every participant is informed about this outcome. Sending and receiving this relatively high number of messages consumes a lot of energy on energy-limited mobile devices making them less suitable for CO role.

Unstable storage and energy constraints make a MN unsuitable for playing the CO role. The CO role requires a powerful node in terms of computation and memory capabilities having a stable storage and not relying on a finite energy source. This substantiates the selection of a FN to play the CO role.

Due to the focus on failure resilient commit, we now consider failures in detail in the following subsections covering *(1)* network disconnections, *(2)* message loss, and *(3)* node failures.

5.4 Tolerating Network Disconnections

Traditional commit protocols trigger Abort if at initiation one or more participants are not connected. These protocols tolerate network disconnection durations of some seconds (using timeouts) and abort the transaction if no connection is observed during this short timeout. These solutions are typically not suitable for mobile environments since the probability that all (mobile) participants are connected at initiation time is usually low. This can be explained by the fact that disconnection time of MNs can range from some minutes to hours or days, or become even permanent due to physical damage, theft or loss. In the following, we provide design principles for tolerating three specific classes of network disconnection, i.e., transient predictable, transient unpredictable and permanent (Fig. 2).

A possible solution to mitigate network disconnection has been proposed by *UCM* [Bobineau et al. 2000]. UCM provides support for disconnections and off-line executions on mobile devices. UCM is a one-phase protocol where the voting phase of the 2PC is eliminated by enforcing some properties on the participant's behavior during the transaction execution. UCM reduces the atomic commit protocol to a single phase that consists in broadcasting the coordinator's decision to all participants. In other words, the coordinator acts as a "dictator" imposing its decision on all participants. UCM has been specifically designed for mobile environments. To guarantee atomicity, transaction operations and their acknowledgments are continuously logged. UCM guarantees strict atomicity and reduces the wireless message complexity. However, UCM is based on strict and hard assumptions such as local pessimistic concurrency control (strict two-phase locking) which is required for all participants, as well as immediate integrity control and homogeneity of participating database systems (we refer to the 1PC assumptions in [Bobineau et al. 2000]). These assumptions restrict also the applicability of UCM to only a subset of the possible applications in mobile infrastructure-based environments.

Since UCM does not represent a generic solution for tolerating network disconnections, we propose to create representatives/proxies of MNs in the fixed part of the network. Introducing representatives is inspired by the M-2PC protocol [Nouali et al. 2005] and the work presented in [Serrano-Alvarado et al. 2003]. In M-2PC the representatives take part in the transaction execution when P-MNs delegate their commitment duties to them. We extend the role of these representatives which can act on behalf of MNs from the begin-

ning and mask their disconnections. This is similar to a lightweight replication of commit data and commit state of the MN in the fixed part of the network, i.e, these representatives will store required information concerning the state of execution of the MT and also the message traffic from and to the corresponding P-MN to be able to act on its behalf in case perturbations occur. These representatives are henceforth termed as mobile node agents (MN-Ag).

MN-Ags are implemented on FNs and are provided as a service to the P-MN by its service provider (the service provider provides infrastructure-based communication facilities to the P-MNs). We consider a MN-Ag per MN and per all the transactions involving the MN. The MN-Ag can play the CO role if the corresponding P-MN is the initiator (I-MN) of the transaction. Fig. 4 shows a scenario where the MN-Ags act on behalf of their corresponding P-MNs in case of a network disconnection. In this scenario, the MN-Ags buffer the messages received from the CO and forward them to the corresponding P-MNs when they reconnect.

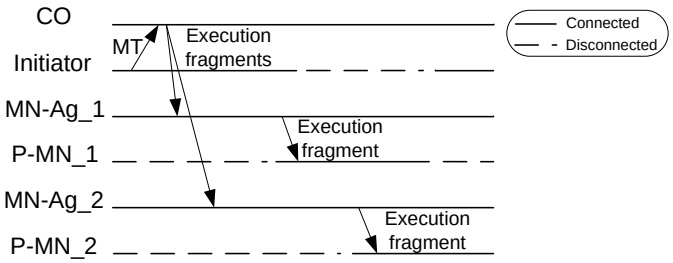

Fig. 4. Tolerating network disconnection – Agent concept

Besides disturbing the Commit/Abort decision process, a network disconnection substantially impacts the *blocking time* of FN resources. Using a classical transaction commit protocol like 2PC, disconnections of P-MNs can block the valuable resources of FNs for an intolerable long time period. Thus, it is crucial to minimize this blocking period. We suggest decoupling the commit of P-MNs from that of P-FNs. The transaction execution is consequently split into *two phases* (Fig. 5). We call the first phase *pre-commit phase* where "sufficient" information is collected from P-MNs after finishing the execution of their corresponding fragments. The second phase called *core phase* involves only P-FNs. Therefore, any classical transaction commit protocol established in fixed networks such as 2PC or 3PC can be used. If the first phase fails, i.e., in case the CO receives at least one "No" vote or TO_{CO} expires before the CO receives all votes, then it is useless to progress with the second phase which avoids blocking the resources of P-FNs. Otherwise, the blocking time of resources on P-FNs is determined by the blocking time of the core phase protocol.

Accordingly, the decoupling allows preventing network disconnections, mainly caused by P-MNs, from affecting P-FNs. For the different network disconnection types different precautions to design the pre-commit phase are discussed in the following.

5.4.1 *Transient Predictable Network Disconnection.* For the considered classes of network disconnections, we assume that only the initiator is connected at the time when the transaction is issued. P-MNs are not required to be connected. A transient predictable

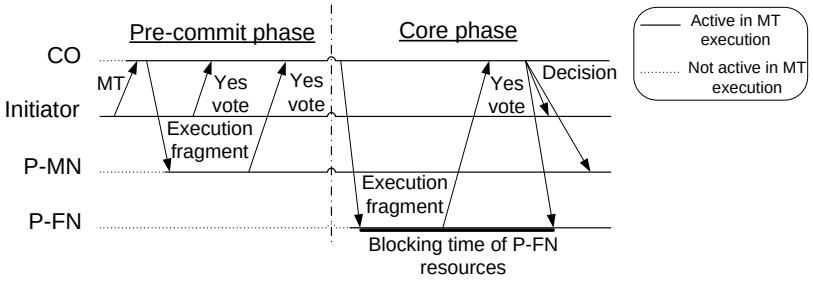

Fig. 5. Tolerating network disconnection – Decoupling concept

network disconnection allows the nodes to share some messages with other entities participating in the execution or coordination of the transaction just before it disconnects. Thus, the node can likely notify other nodes about its expected disconnection (Fig. 6). One solution to this problem could be to wait (the CO) for a predefined amount of time (TO_{CO}) and abort the transaction. This solution increases the number of transaction Aborts in this environment and also the costs associated with the re-initiation of aborted transactions in terms of messages and energy. The aborted transaction should be also delayed to be executed in the near future which can affect other dependent/related transactions.

For transient predictable network disconnection the P-MN knows approximately when it will disconnect. If it additionally knows how long it will disconnect, then a timeout extension ($TO_{ext}(P\text{-}MN)$) is easy to determine for the P-MN. $TO_{ext}(P\text{-}MN)$ is an update of $TO_{P\text{-}MN}$ which should be sent to the CO to update its timeout TO_{CO}. If the P-MN lacks this information, a table of possible scenarios for network disconnection and the corresponding estimated disconnection times can be used. As a representative of the P-MN, the MN-Ag can take decisions about $TO_{ext}(P\text{-}MN)$ on the P-MN's behalf when the P-MN is disconnected (Fig. 6). This decision should be taken in case of a transient predictable network disconnection based on the information sent by the corresponding P-MN before disconnecting.

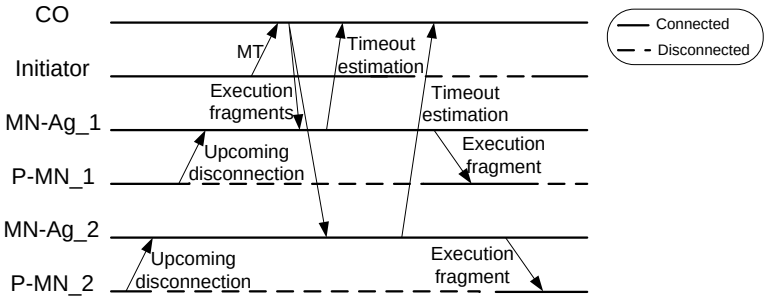

Fig. 6. Tolerating predictable network disconnection

For delay-tolerant transactions $TO_{ext}(P\text{-}MN)$ can be evaluated based on the P-MN's behavior in the past. A history of previous disconnections can assist in estimating an appropriate value for the $TO_{ext}(P\text{-}MN)$ for future disconnections.

5.4.2 *Transient Unpredictable Network Disconnections.* This type of network disconnections refers to the case when the P-MN disconnects sans any other entity participating in the execution of the transaction is updated on the state of this disconnection. This occurs if the P-MN was not able to communicate with its MN-Ag or with any other participant to share its actual status before disconnection. When a transient network disconnection occurs a timeout extension can only be triggered by either the CO or the MN-Ag, where the timeout selection may be less suitable. For this, we suggest that each P-MN specifies for its MN-Ag a *default timeout extension value* for the case the network disconnection is unpredictable. The MN-Ag should also have the possibility to extend the timeouts of its corresponding P-MN when needed. We suggest here that the MN-Ag develops an experience log for transient unpredictable network disconnection tailored to its corresponding P-MN. For example, to first estimate the disconnection cases of short duration (e.g., tunnel (GSM), handover (GSM-UMTS-WLAN), SW transient failures) and subsequently of medium length (e.g., hardware transient failures) and for long-duration reasons (e.g., discharged battery) and in worst case long-duration network disconnections may become permanent network disconnections due to damage or loss.

5.4.3 *Permanent Network Disconnections.* When a certain P-MN is lost, stolen or damaged, we consider that a permanent network disconnection has occurred. In this case, the transaction should be aborted. If the P-MN has voted with "Yes" before crashing permanently the transaction could be committed if the changes done by the P-MN on his local database are available to its MN-Ag or to the CO. These changes can be propagated to the main copy of the data located, e.g., on a backup server on the wired network, or to the user. In case of permanent network disconnections, the MN-Ag can be used to store these changes and will be responsible for its propagation to the backup server or to the user whenever needed.

5.5 Tolerating Message Loss

Message loss is a common occurrence over wireless links. A message loss can be tolerated by using *acknowledgments (Acks)* and *timers* with appropriate timeout values. Acks constitute an overhead in term of messages for the considered mobile environment. Therefore, the number of Acks exchanged during the execution of atomic commit protocols should be kept minimal. Timers can also be used to detect loss of messages by setting an appropriate timeout value after which the message is assumed to be lost. Trade-off can be obtained when to use Acks or timers or a combination of both.

5.6 Tolerating Node Failures

As mentioned before, we classify node failures into MN and FN failures. We note that a node failure implies a network disconnection besides data loss, which we investigate in the following subsections. MN failures can either be transient or permanent. To cope with transient failures local recovery is used. Permanent failures can be tolerated using replication.

5.6.1 *Transient MN Failures.* Local recovery is the set of operations that must be performed locally by a node after a transient failure to recover to a correct, consistent and failure-free state. For this purpose, a set of precautions have to be conducted during the failure-free operations. To identify these precautions, it is needed to categorize the situations where recovery is needed. For example if failures occur they should not result in loss

of commit data or commit state on a node or a subset of nodes. We classify these situations into isolated node failures and combined node failures.

Isolated node failures consist of scenarios where only one P-MN is affected by a transient failure (e.g., battery depletion) during the execution of a MT. Such failures result in the loss or corruption of the content of volatile storage. A common solution is the *logging* of all needed operations related to the execution of the transaction commit protocol onto *stable storage*. A log represents a durable record or history of the significant events such as Write, Commit and Abort that have occurred locally at one site. When a failure occurs the recovery is performed based on the logged information. But which data should we log? How frequent should logging be performed? And where to log this data if the MNs are not assumed to have a stable storage? Answering these questions requires a detailed knowledge about the transaction commit protocol and the interaction of the participants. We will detail this issue while presenting our protocol family in Section 6.

Combined node failures include scenarios when more than one node at a time observe a transient failure and these failures can be either similar or different in nature. In these situations recovery may need global information about the state of execution of the transaction depending on the transaction commit protocol that defines which information needs to be exchanged between the different nodes either participating in the transaction or responsible for its coordination. Usually, this global information is stored and managed by the CO.

5.6.2 *Permanent MN Failures.* This type of failures can be tolerated if the data or the logs related to the execution of MTs on MNs are replicated (e.g., on MN-Ags) before committing the transaction. Logs must not be replicated if the transaction is aborted since the MT should not have any effect on the data stored on the MN in this case. If the final decision is Abort, the logs are only needed locally.

5.6.3 *FN Failures.* Decoupling, as discussed in Section 5.4, allows for easy reuse of existing techniques to handle FN failures. For the sake of completeness, we provide a short overview of the basic FT recovery mechanisms. For a detailed survey we refer the reader to [Elnozahy et al. 2002]. An important FT mechanism useful to tolerate FN failures is *checkpointing*. A checkpoint is a record of a consistent state that existed on a node at some time in the past. Checkpointing is usually used along with logging. The log contains the history of significant events since the start of the transaction or since the last checkpoint. If a failure occurs the node needs to rollback to a consistent state. Recovery strategies can be divided into roll-backward recovery, roll-forward recovery or a combination of both. *Roll-backward recovery* brings the system back to a previous correct and consistent state. Checkpoints are made periodically during normal operations by recording (on stable storage) the current state. This represents a big performance overhead for on-going transactions. After the occurrence of a failure, the state can be restored from the checkpoint information. *Roll-forward recovery* brings the system to a new correct state after a crash. This may involve asking another site what the current state is if data is replicated. *The combination of both* (often used in databases) uses both checkpoint and recovery logs and proceeds as follows: *(a)* take a checkpoint and delete old log and start new log, *(b)* log all significant messages, transactions, etc, up to the next checkpoint, and *(c)* when recovering from failure, restore the checkpoint state then replay the log and re-do these operations.

Because the CO is a FN, we adopt the fault-tolerance and recovery strategies discussed above for FN failures. In addition, we investigate which information related to the exe-

cution and coordination of the MT should be stored on a stable storage to allow the CO to recover from node failures. Beyond storing the information related to the MT such as the commit set, the corresponding execution fragments and the identity of the CO, the CO needs to maintain information about the status of execution of every execution fragment. We distinguish between the following states: (a) *idle*, (b) *active*, (c) *pre-committed* (only for MNs and if decoupling is used), (d) *committed* and (e) *aborted*. Being in the "idle" state means that the participant has not started executing its fragment yet. After starting the execution the state becomes "active". If decoupling is implemented and its first phase succeeds, the state of P-MN is updated to "pre-committed", otherwise the state is "aborted". The state "committed" is reached when the whole MT is committed, otherwise the final state is set to "aborted".

6. PROTOCOL FAMILY FOR MOBILE TRANSACTION COMMIT

In the perturbation-resilient transaction commit framework described in Section 5, we *(a)* investigated and classified the perturbations in the considered mobile environment, and *(b)* presented appropriate design techniques to provide resilience to each identified class under consideration of the design requirements and issues listed in Section 4. Now, we combine the set of presented design techniques into a family of transaction atomic commit protocols. Since different application scenarios show different perturbation classes, we integrate only the necessary building blocks for the most common mobile environment classes implementing the main identified fault-tolerance and recovery techniques. For example in a mobile system where failures are not frequent, using a protocol with sophisticated FT techniques will only add unnecessary overhead to the system decreasing its efficiency. It is noteworthy that our family of protocols does not provide adaptivity in the sense of [Serrano-Alvarado et al. 2005; Nouali-Taboudjemat and Drias 2008; Karlsen 2003], but it represents customized solutions to pre-defined classes of mobile infrastructure-based environments based on the nature of perturbations characterizing every considered class and that our strengths reside in the modularity of our approach.

6.1 Overview

We construct a family of atomic commit protocols for three common classes of mobile infrastructure-based environments:

(1) Failure-free environments with environmental constraints on nodes and links as described in Section 5.3. An industrial plant scenario (inventory, smart fabric etc.) is an example of this infrastructure-based environment class. Mobile participants are very heterogeneous in this scenario, ranging from PDAs to laptops on mobile autonomous vehicles with a well maintained reliability of mobile devices and wireless networks.

(2) Environments where additional to arbitrary environmental constraints frequent network disconnections and message losses occur (network disconnections and message losses are investigated in Section 5.4 and Section 5.5 respectively). A representative scenario for this class is a business scenario where mobile participants are relatively robust laptops that are carried by employees commuting between home, office and customers. Typically, the laptops are equipped with WLAN network interfaces and are only sporadically connected depending on the base station coverage (hotspots).

(3) Environments where arbitrary node failures (explored in Section 5.6) may occur in addition to the previous class *(2)*. A flea market scenario is an appropriate example for this environment class. In this scenario, the heterogeneity and robustness of mobile participants

and wireless networks cannot be controlled. Customers and merchants use their mobile devices for searching, offering and buying items.

We propose a protocol for each of the outlined mobile environment classes. Accordingly, we distinguish across the set of proposed protocols (Table I):

(1) Pre-Phase Transaction Commit (PPTC), a basic protocol derived from the fault-tolerant pre-phase transaction commit presented in [Ayari et al. 2006] and suitable for failure-free environments described above. This version includes a minimal set of the main design techniques a transaction commit protocol should include to cope with environmental constraints. PPTC implements mainly the concepts and techniques presented in Section 5.3 and which deal with environmental constraints, i.e., heterogeneity of nodes and links, unstable storage and energy.

(2) Fault-Tolerant PPTC (FT-PPTC), a protocol which implements fault-tolerance in addition to resilience to environmental constraints. FT-PPTC is based on the fault-tolerant pre-phase transaction commit presented in [Ayari et al. 2006] and is customized to scenarios such as the business scenario presented above. Therefore, FT-PPTC implements in addition to PPTC the concepts and techniques presented in Sections 5.4 and 5.5 to cope with the different types of network disconnections and also with message losses.

(3) Fault-Tolerant and Recovery PPTC (FT-PPTC-Rec), a protocol which enriches FT-PPTC with the necessary mechanisms for recovery in case of node failures, i.e., FT-PPTC-Rec implements additionally to FT-PPTC the concepts and techniques presented in Section 5.6 that provide resilience to different types of nodes failures. FT-PPTC represents an appropriate solution to environments with arbitrary perturbations introduced above.

Table I summarizes the building blocks of each protocol. These protocols are then compared in Section 7 to emphasize the impact of the various building blocks. We emphasize here the modularity of our framework as further combinations of the building blocks to construct alternate protocols or protocol adaptations are possible.

Table I. Perturbation-resilience of the proposed protocol family

	Protocol		
Building Block	PPTC (Section 6.2)	FT-PPTC (Section 6.3)	FT-PPTC-Rec (Section 6.4)
Tolerating environmental Constraints (Section 5.3)	+	+	+
Tolerating Network disconnections (Section 5.4)	-	+	+
Tolerating Message Loss (Section 5.5)	-	+	+
Tolerating Nodes Failures (Section 5.6)	-	-	+

6.2 Base Protocol: PPTC

The PPTC protocol is our basic step towards perturbation-resilient atomic commit protocols for mobile environments. PPTC implements the necessary techniques to cope with the main environmental constraints described in Section 5.3. We will refer to the used techniques and covered environmental constraints while describing the protocol.

6.2.1 *Protocol Overview.* As mobile participants may need an arbitrary long time to execute their fragments, and as very few assumptions can be made regarding the performance of their wireless links, resources of fixed participants may potentially be blocked for an undefined period of time. Therefore, PPTC uses our decoupling strategy to *decouple*

the commit of mobile participants from that of fixed participants. In the pre-commit phase (Fig. 7), PPTC collects the votes of mobile participants to be able to reduce the commit set to a set of entities in the fixed network. The core phase involves only FNs and can be completed by any atomic commit protocol for wired networks, such as the traditional 2PC protocol. Consequently, we term this as the *core 2PC PPTC phase* as we select the established 2PC protocol to implement it. 2PC was arbitrarily chosen for this phase of PPTC especially because it is widely used in wired networks. This is not a restriction since any other established commit protocols in fixed networks can be used in this phase.

The pre-commit phase involves only P-MNs (see Fig. 7). As discussed in Section 5.3, a timeout-based concept is exploited to reach a provisional Commit decision at the end of the pre-commit phase (Fig. 7). The only difference to the timeout-based concept used in this protocol is that S_t represents an estimated upper bound for the time needed to send a vote to the CO. The CO waits for the expiration of TO_{CO} and finalizes the pre-commit phase by a provisional Commit or an Abort decision. The CO proceeds to the second phase of PPTC, only if it receives "Yes" votes from all P-MNs within the specified time-limit (TO_{CO}). The transaction is aborted as soon as one P-MN aborts the transaction or TO_{CO} expires at the CO before receiving all the votes of P-MNs.

As a result of the pre-commit phase, the P-MNs delegate the CO to execute the 2PC protocol on their behalf. The second phase of the protocol begins when the CO sends the execution fragments of P-FNs to their corresponding FNs and the 2PC protocol is executed to collect their votes. If all P-FNs vote for committing the MT, the CO decides to commit, otherwise it decides to abort the mobile transaction.

6.2.2 *Detailed Protocol Description.* Fig. 7 illustrates the execution of the PPTC protocol. The activities of each participant are outlined below. Specifically, we detail the activities of the initiator, CO and P-MN later in this chapter in Algorithm 1, Algorithm 2 and Algorithm 3 respectively. Without loss of generality, we consider in the following the initiator to be a MN (I-MN) since this is the more interesting case to investigate.

Activities of the I-MN. The I-MN initiates the mobile transaction T_i, extracts its execution fragment e_i(*I-MN*), computes its E_t, S_t and $lifetime$ of the initiated transaction and sends them along with the rest of the MT $T_i - e_i$(*I-MN*) to the CO (Algorithm 1). The I-MN begins the processing of e_i(*I-MN*). Whenever the I-MN needs to extend its E_t and/or S_t, it sends a message to the CO with the new timeout value(s). The I-MN sends a "No" vote to the CO whenever it decides to abort the MT. If the I-MN successfully completes the execution of its fragment, it sends a "Yes" vote to the CO. After receiving the final decision the I-MN (like the other P-MNs) is not supposed to send an "Ack" message to the CO since the PPTC protocol is not designed to tolerate message losses. P-FNs acknowledge the CO upon receiving the final decision as part of the 2PC protocol which is adopted for the core phase in PPTC.

Activities of the CO. In PPTC, the CO is the MN-Ag of the I-MN. Upon receiving $T_i - e_i$(*I-MN*) from the I-MN, the CO extracts the execution fragments of the P-MNs and sends each fragment to its corresponding P-MN. The CO computes also the timeout of the MT (Algorithm 2, lines 5-9). If the lifetime received from the I-MN is not undefined, the timeout of the MT is set to $lifetime$. Otherwise, if the CO receives E_t and/or S_t from any P-MN, it updates its timeout (lines 11-13). The CO waits for the expiration of TO_{CO_i}. If it receives a "Yes" vote from each P-MN within this time, it distributes the

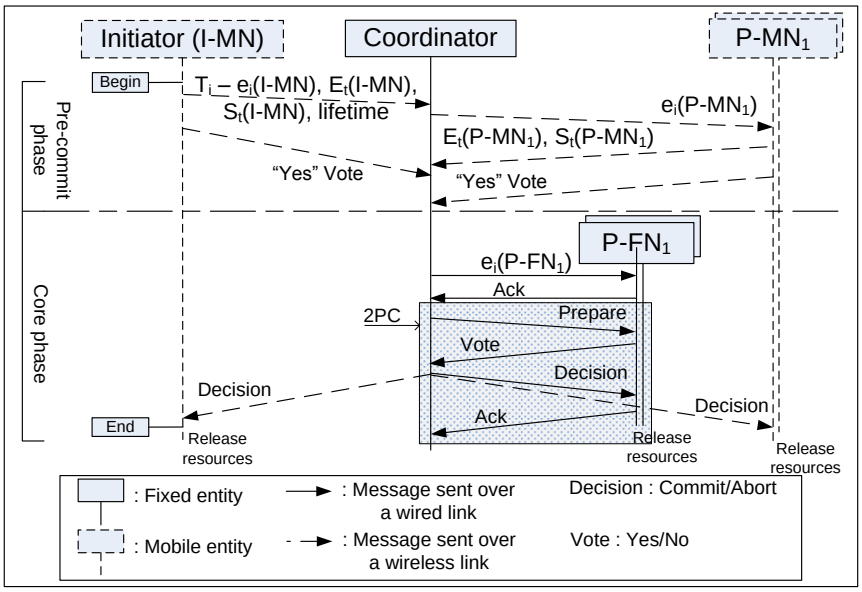

Fig. 7. Scenario execution of the MT T_i using the PPTC protocol

execution fragments of the P-FNs along with the vote request ("Prepare" message). If the CO receives an "Abort" message from any of the P-MNs before the expiration of TO_{CO_i} or if it doesn't receive the vote of at least one P-MN within this timeout, it sends an "Abort" message to the rest of the P-MNs and the whole transaction is aborted. After sending the execution fragments of the P-FNs, the CO starts a 2PC protocol session to collect the votes from them. If the CO receives a "Yes" vote from all the P-FNs, it decides to commit the transaction and sends Commit decision to all the participants. If it receives at least one "No" vote (or no reply) it decides to abort the transaction and sends Abort decision to all participants.

Activities of a P-MN. Upon receiving its execution fragment, the P-MN computes E_t and S_t, and sends them to the CO (Algorithm 3). The P-MN behaves then exactly like the I-MN.

Activities of a P-FN. P-FNs behave according to the established 2PC protocol, i.e., a P-FN executes its fragment, waits for the "Prepare" message, sends its vote and waits for the decision. Upon receiving the decision, the P-FN acknowledges the CO. Note that 2PC is not a restriction here and any further established protocol such as 3PC or Paxos Commit can be used.

6.2.3 *Perturbation Coverage and Application Scenarios.* By using a timeout-based strategy and by choosing a FN to play the CO role, the PPTC protocol copes with a wide range of environmental constraints (i.e., heterogeneity of nodes and links, unstable storage and energy) as additionally confirmed by our evaluation results presented in Section 7. Therefore, PPTC suits well for closed mobile environments or those with high coverage and availability (such as commit between participants possessing GPRS or UMTS connec-

Algorithm 1: I-MN's Algorithm (PPTC)

```
1  Initialize T_i;
2  Extract execution fragment e_i(I-MN);
3  Compute E_t(I-MN), S_t(I-MN) and lifetime;
4  send T_i − e_i(I-MN), E_t(I-MN), S_t(I-MN) and lifetime to CO;
5  while processing e_i(I-MN) do
6      if E_t(I-MN) and/or S_t(I-MN) need to be extended then
7          compute new timeout value(s);
8          send new value(s) to CO;
9      end
10 end
11 if I-MN decides to abort T_i then
12     abort T_i;
13     send "No" vote to CO;
14     return;
15 else                                   /* I-MN decides to commit T_i */
16     send "Yes" vote to CO;
17     wait for decision from CO
18     if decision is Commit then
19         commit T_i;
20         return;
21     else                                /* decision is Abort */
22         abort T_i;
23         return;
24     end
25 end
```

tivity), where failures can be controlled and are rare. Using more sophisticated protocols (such as FT-PPTC or FT-PPTC-Rec that we will present next) in this environment class will result in a non-beneficial performance overhead introduced by these protocols.

6.3 FT Coverage Protocol: FT-PPTC

FT-PPTC extends the PPTC protocol by adding fundamental fault-tolerance techniques presented in our framework in Sections 5.4 and 5.5 (Table I). FT-PPTC tolerates in addition to environmental constraints network disconnections and message losses.

6.3.1 *Protocol Overview.* To support and improve the decoupling strategy in FT-PPTC, a MN-Ag is assigned to each P-MN as discussed in Section 5.4. Fig. 8 shows that the MN-Ag acts as an intermediate between the CO and its corresponding P-MN. The MN-Ag in FT-PPTC is also responsible for executing the 2PC protocol on behalf of its corresponding P-MN as illustrated in Fig. 8. The MN-Ag can also take the CO role if the corresponding P-MN is the initiator of the transaction, i.e., the I-MN. As described in Section 5.4, introducing the agents simplifies the handling of the different types of communication failures (network disconnection and message loss).

6.3.2 *Detailed Protocol Description.* Fig. 8 shows a scenario where P-MN$_1$ observes a predictable transient disconnection. Before disconnecting from the network, it informs its MN-Ag about its extended timeouts which are forwarded to the CO. The CO updates

Algorithm 2: CO's Algorithm (PPTC)

wait for $T_i - e_i$(*I-MN*), E_t(*I-MN*), S_t(*I-MN*) and $lifetime$;
extract and send execution fragments to the P-MNs;
initialize all the timeouts of the P-MNs to 0;
let $P_m = \{$*P-MN*$_1, \ldots,$ *P-MN*$_m\}$ the set of all the P-MNs;
if *lifetime is undefined* **then**
    $TO_{CO_i} \leftarrow \max(E_t($*P-MN*$_1) + S_t($*P-MN*$_1), \ldots, E_t($*P-MN*$_m) + S_t($*P-MN*$_m))$;
else
    $TO_{CO_i} \leftarrow lifetime$;
end
while *waiting for* TO_{CO_i} *to expire* **do**
    if *lifetime is undefined AND value of* E_t *and/or* S_t *(initial or extended values) of one of the MNs in* P_m *is received* **then**
        recompute TO_{CO_i};
    end
    if Abort *message is received from one of the MNs in* P_m **then**
        send Abort to **all** members of P_m;
        return;
    end
end
if *Yes Vote is received from each P-MN* **then**
    start a 2PC protocol to collect the votes from **all** P-FNs;
    if *all votes were Yes* **then**
        send Commit message to **all** members of the commit set;
        return;
    else /* at least one of the votes is No */
        send Abort to **all** members of the commit set;
        return;
    end
else /* at least one Vote not received */
    send Abort to **all** members of the commit set;
    return;
end

Algorithm 3: P-MN's Algorithm (PPTC)

wait for receiving the corresponding execution fragment;
Compute E_t(*Par-MN*) and S_t(*Par-MN*);
send E_t(*Par-MN*) and S_t(*Par-MN*) to CO;
/* continue with step 5 to step 25 of Algorithm 1 substituting I-MN with P-MN */

its own timeout. After reconnecting to the network, P-MN$_1$ sends its "Yes" vote (in this scenario) to the CO by the intermediate of its MN-Ag. The CO decides to proceed to the core phase because its received "Yes" votes from all P-MNs before the expiration of its timeout.

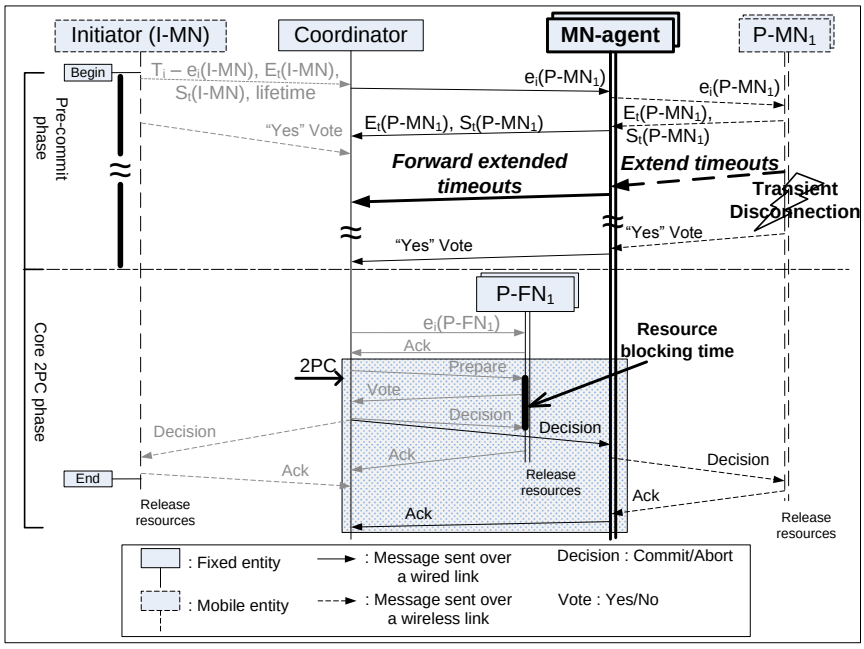

Fig. 8. Failure-prone execution of the MT T_i using the FT-PPTC protocol (for simplicity only transient disconnections are illustrated)

Activities of MN-Ag. Upon receiving the execution fragment of its corresponding P-MN from the CO, the MN-Ag forwards it to the corresponding P-MN (Algorithm 4, line 3). After receiving E_t and S_t from the P-MN, the MN-Ag forwards this information to the CO. After receiving a "Yes" or "No" vote from the P-MN, the MN-Ag forwards the vote to the CO. Upon receiving the decision from the CO, the MN-Ag forwards it to the P-MN as soon as it is available (connected to the network). After receiving the "Ack" for decision reception from the P-MN, the MN-Ag acknowledges the CO. It is key to mention that the MN-Ag is not an active participant in the execution of the MT, since it does not have to know any information about the application and does not need to process any part or fragment of the MT.

The MN-Ag can take some decisions on behalf of his corresponding P-MN which are discussed next. These decisions include the extension of the timeouts of the P-MN in case of a transient (predictable or unpredictable) disconnection. The MN-Ag is also given the responsibility to send an estimation of the timeouts of the corresponding P-MN direct after receiving the execution fragment of this P-MN (line 2). This estimation can be corrected after receiving new timeout values (E_t and S_t) from the P-MN.

The extension of FT-PPTC with respect to PPTC affects the I-MN and the P-MN only by requiring them to send an "Ack" message to their corresponding MN-Ags when they receive the final decision (Fig. 8). This allows FT-PPTC to tolerate additional message losses compared to PPTC. Moreover, the CO communicates in FT-PPTC with the P-MNs through their corresponding MN-Ags.

Algorithm 4: MN-Ag's Algorithm (FT-PPTC)

```
1  wait for receiving execution fragment e_i(P-MN) of the corresponding P-MN from CO;
2  send an estimation of the timeouts of corresponding P-MN to the CO;
3  forward e_i(P-MN) to corresponding P-MN as soon as it is available;
4  for any received message do
5      if message contains possible disconnection of the corresponding P-MN and its reasons
       then
6          recompute the timeouts based on disconnection reasons;
7          update the token with this information;
8          send extended timeouts to the CO;
9      else if message is sent by the CO then
10         forward message to the corresponding P-MN as soon as it is available;
11     else if message is sent by P-MN then
12         forward message to CO;
13
14 end
```

6.3.3 *Perturbation Coverage and Application Scenarios.* By deploying MN-Ags, using an adaptive timeout extension scheme and utilizing positive acknowledgements, FT-PPTC handles a wide range of network failures additionally to environmental constraints. Compared to PPTC, FT-PPTC is tailored to atomic commit in environments using wireless technologies with limited coverage (such as WLAN) and where communication failures (i.e., network disconnections and message loss) are arbitrary due to the openness of the system and the number of participants joining and leaving the system. FT-PPTC tolerates that P-MNs intentionally disconnect during the execution of their fragments (e.g., to save energy). The disconnection time can also be adapted to the application needs and communicated to the corresponding MN-Ag that accordingly informs the MT CO. In order to tolerate network disconnections and message losses, FT-PPTC adds an additional overhead in terms of message complexity as compared to PPTC (as investigated in Section 7.1.2). Consequently, PPTC will outperform FT-PPTC in environments where only constraints constitute the dominating commit perturbation. However, PPTC will lead to high abort rates in scenarios where communication failures are frequent.

6.4 FT and Recovery Protocol: FT-PPTC-Rec

FT-PPTC-Rec extends FT-PPTC to tolerate node failures. The CO and MN-Ags use a combination of logging and checkpointing scheme (Section 5.6) to recover if a failure occurs during the execution of the mobile transaction.

6.4.1 *Detailed Protocol Description.* Upon receiving $T_i - e_i$(*I-MN*) from the I-MN, the CO creates a *Token* for T_i, which includes one entry for each e_i(*P-MN*) and contains the identity of the CO and the commit set (Algorithm 5, lines 1-2). Each entry includes the state of processing of e_i(*P-MN*) as discussed in Section 5.6 (idle, active, pre-committed, committed and aborted states). The state of e_i(*I-MN*) is set to "active" and the state of the rest of the execution fragments is set to "idle". The entries for the execution fragments of the P-MNs additionally include their corresponding E_t and S_t. After receiving E_t and S_t from one P-MN, the CO sets the state of its corresponding fragment to "active" and updates its timeout values (E_t and S_t). If a new E_t and/or S_t (extended values) is received either

Algorithm 5: CO's Algorithm (FT-PPTC-Rec)

1 **wait for** $T_i - e_i$(*I-MN*), E_t(*I-MN*), S_t(*I-MN*) and $lifetime$;
2 create a *Token* for T_i;
3 **set** the state of the MT to "active" in T_i's Token;
4 extract and send execution fragments of P-MNs to their corresponding MN-Agents;
5 initialize all the timeouts of the P-MNs with 0;
6 let $P_m = \{$*P-MN*$_1, \ldots,$ *P-MN*$_m\}$ the set of all the P-MNs;
7 **if** *lifetime is undefined* **then**
8 $\quad TO_{CO_i} \leftarrow \max(E_t($*P-MN*$_1) + S_t($*P-MN*$_1), \ldots, E_t($*P-MN*$_m) + S_t($*P-MN*$_m))$;
9 **else**
10 $\quad TO_{CO_i} \leftarrow lifetime$;
11 **end**
12 **while** *waiting for* TO_{CO_i} *to expire* **do**
13 $\quad$ **if** *lifetime is undefined AND value of* E_t *and/or* S_t *(initial or extended values) of one of the MNs in* P_m *is received* **then**
14 $\quad\quad$ recompute TO_{CO_i};
15 $\quad\quad$ update the Token of T_i with the received value(s);
16 $\quad$ **end**
17 $\quad$ **if Abort** *message is received from one of the MNs in* P_m **then**
18 $\quad\quad$ **set** the state of the MT to "aborted" in T_i's Token;
19 $\quad\quad$ **send Abort** to **all** members of P_m;
20 $\quad\quad$ **return**;
21 $\quad$ **end**
22 **end**
23 **if** *updates are received from I-MN and a "Yes" vote from each MN-Ag* **then**
24 $\quad$ **write** received updates to the corresponding Token;
25 $\quad$ **set** the state of the MT to "pre-committed" in T_i's Token;
26 $\quad$ start a 2PC protocol to collect the votes from **all** P-FNs;
27 $\quad$ **if** *all votes were "Yes"* **then**
28 $\quad\quad$ **set** the state of the MT to "committed" in T_i's Token;
29 $\quad\quad$ **send Commit** message to **all** members of the commit set;
30 $\quad\quad$ **return**;
31 $\quad$ **else** `/* at least one of the votes is No */`
32 $\quad\quad$ **set** the state of the MT to "aborted" in T_i's Token;
33 $\quad\quad$ **send Abort** to **all** members of the commit set;
34 $\quad\quad$ **return**;
35 $\quad$ **end**
36 **else** `/* either the updates of the I-MN or at least one vote from the MN-Ag of a P-MN in` P_m `- I-MN are not received */`
37 $\quad$ **set** the state of the MT to "aborted" in T_i's Token;
38 $\quad$ **send Abort** to **all** members of the commit set;
39 $\quad$ **return**;
40 **end**

from I-MN or from a P-MN, then the CO also updates the Token. If it receives the updates from the I-MN (updates are the local DBMS logs of the operations belonging to the mobile transaction) and a "Yes" vote from every MN-Ag involved in the MT within its computed timeout TO_{CO_i} (Algorithm 5, line 23), it stores them in the corresponding Token and sets

Algorithm 6: MN-Ag's Algorithm (FT-PPTC-Rec)

```
1  wait for receiving execution fragment e_i(P-MN) of the corresponding P-MN from CO;
2  create a Token for e_i(P-MN);
3  set the state of e_i(P-MN) to "idle" in T_i's Token;
4  send an estimation of the timeouts of corresponding P-MN to the CO;
5  for any received message do
6      if message contains the timeouts of the P-MN then
7          update T_i's Token with the received timeouts;
8          set the state of the e_i(P-MN) to "active" in T_i's Token;
9          forward the timeouts to the CO;
10     else if message contains the updates of the corresponding P-MN then
11         update the Token with the received updates;
12         send "Yes" vote to CO;
13     else if message contains possible disconnection of the corresponding P-MN and its
       reasons then
14         recompute the timeouts based on disconnection reasons;
15         update the token with this information;
16         send extended timeouts to the CO;
17     else if message is sent by the CO then
18         update the Token with the received message;
19         send the received message to the corresponding P-MN as soon as it is available;
20     else if message is sent by P-MN then
21         update the Token with the received message;
22         send the received message to CO;
23
24 end
```

the state of the MT to "pre-committed". Similar to the CO, MN-Ags implement a logging and checkpointing scheme to recover from failures (Algorithm 6). MN-Ags log all the messages sent to and received from their corresponding P-MNs and also store the local DBMS logs sent by the P-MN if it votes for committing the MT. The DBMS logs contain all the changes made by the transaction to the P-MN local data stored on its DBMS such as Write, Commit and Abort events. It is noteworthy to mention that all P-MNs including the I-MN are required to force-write (flush updates to their storage) their votes and composed logs before sending this information to their corresponding MN-Ags in order to tolerate transient MN failures.

6.4.2 *Perturbation Coverage and Application Scenarios.* By using a logging and checkpointing scheme and replicating the local database logs on the MN-Ags, FT-PPTC copes with wide range of node failures in addition to network failures and environmental constraints. Compared to FT-PPTC, FT-PPTC-Rec is well suited to environments where node failures cannot be controlled. Similar to FT-PPTC, FT-PPTC-Rec copes with network disconnections, message loss and environmental constraints covering all the spectrum of perturbations taken into consideration in this work. From the message complexity point of view, FT-PPTC-Rec does not add any message overhead as compared to FT-PPTC but more data needs to be transferred from the P-MN to its corresponding MN-Ag such as the local database logs. Additionally, there is an overhead in the amount of data that needs

to be stored on the different nodes participating in the execution and coordination of the MT. This overhead is due to the logging and checkpointing scheme used. Consequently, PPTC and FT-PPTC will outperform FT-PPTC-Rec in environments where either only environmental constraints or network failures and environmental constraints constitute the dominating commit perturbation(s). However, PPTC and FT-PPTC will lead to high abort rates and loss of execution state in scenarios where node failures are frequent.

6.5 Correctness Basis

According to [Bernstein et al. 1987], an atomic commit protocol has to satisfy the following five *atomicity properties or conditions*:

- *Stability:* A process cannot reverse its decision after it has reached one.
- *Consistency:* All processes that reach a decision reach the same one.
- *Validity:* The Commit decision can only be reached if *all* processes voted "Yes".
- *Non-triviality:* If no failure occurs and all processes voted "Yes", then the final decision should be commit.
- *Termination:* At any point in execution, if all existing failures are repaired and no new failures occur for sufficiently long time, then all processes will eventually reach a decision.

To show the correctness of the proposed family of solutions, we need to prove that each protocol satisfies the five properties listed above. It is important to notice that FT-PPTC-Rec protocol (as an extension of FT-PPTC and PPTC protocols) does not add additional functionality to FT-PPTC or PPTC but represents an improvement of some extra-functional properties of these protocols. Therefore, we only need to prove that FT-PPTC-Rec satisfies the atomicity properties. It follows directly from the specification of the FT-PPTC-Rec protocol in Section 6.4 that it satisfies the stability and the non-triviality properties. The consistency property is satisfied due to the fact that only the CO decides about the outcome of the transaction and distributes the same final decision to each participant in the commit set. Therefore, we need to prove that it satisfies the validity and termination properties.

Validity: We assume that the CO decides to commit the transaction when at least one of the participants has not decided yet. If this participant is a MN then the CO sends the rest of the transaction to the FNs before receiving a "Yes" vote from its corresponding MN-Ag or before receiving the updates if this MN is the I-MN. Obviously, this is in contradiction with the protocol specification. If this participant is a FN, then the 2PC protocol decides to commit the transaction before receiving all the votes from the P-FNs, which again contradicts the specification of the 2PC protocol. In the case that at least one of the participants decides to abort the transaction, the CO cannot decide to commit the whole transaction because this decision will violate the protocol specification. Hence, the Commit decision can only be reached if all processes voted "*Yes*", i.e., decided to commit the transaction.

Termination: We consider any execution containing the failures listed in the fault-model detailed in Section 3.2. For this proof we need to consider two aspects. The first aspect is whether the protocol can block at any time making all the participants waiting for an undefined amount of time for the final decision. Since our protocol is based on timeout for coordinating the execution of the fragments of P-MNs, the CO cannot block waiting for messages from these participants. The participants also cannot block waiting for a message from the CO (which is the decision) since they are required to acknowledge this message

and thus the CO is able to detect a message loss and re-send the message. The second aspect considers whether the participants are able to reach any decision after recovery or not. The P-MNs can reach a decision after recovery by asking either the CO or the corresponding MN-Ag (where the necessary state of execution of the transaction is stored) about the outcome of the transaction. For the P-FNs this is guaranteed by the recovery protocols applied there. We note that keeping the state of the execution of the MT on a stable storage allows the continuation of the execution after recovery and eventually reaching a decision.

7. PERFORMANCE EVALUATION AND DISCUSSION

To evaluate the efficiency of the family of protocols developed in this work, we first qualitatively compare FT-PPTC-Rec and FT-PPTC with existing commit protocols M-2PC [Nouali et al. 2005], CO2PC [Serrano-Alvarado et al. 2004; Serrano-Alvarado 2004], TCOT [Kumar et al. 2002] and UCM [Bobineau et al. 2000], regarding their perturbation resilience and message complexity. Next, we use both simulations and experiments to investigate the performance of the PPTC protocol with respect to the throughput and resource blocking time. The perturbation resilience of FT-PPTC is also investigated using our simulation model.

7.1 Comparison to other Existing Approaches

We now compare the FT-PPTC-Rec protocol - which provides maximal perturbation resilience among all other members of the protocol family - to the M-2PC, CO2PC, TCOT and UCM protocols regarding the environmental constraints and failures considered in each protocol. Next, the message complexity, i.e., the number of messages exchanged during execution of these protocols, is investigated.

Table II. Coverage of perturbations (++: Comprehensive, +: Basic, -: No coverage)

Protocol	Heterogeneity	Transient MN Failure	Permanent MN Failure	FN & CO Failure	Message Loss	Network Disconnection
FT-PPTC-Rec	++	++	++	++	++	++
FT-PPTC	++	+	+	+	++	++
M-2PC	+	+	-	-	++	+
CO2PC	+	+	-	-	-	+
TCOT	+	++	++	-	-	-
UCM	-	+	-	-	++	+

7.1.1 *Perturbations Discussed in Different Protocols.* In Table II, we compare these protocols with respect to their investigation and coverage of the perturbations classified in our framework. In this table, (++) indicates a comprehensive coverage, (+) a basic coverage and (-) no coverage of the identified perturbations. In the case of comprehensive coverage, the perturbation is explicitly investigated and detailed by the work. Basic coverage implies that the perturbation is either cursorily investigated or we assume that the authors implicitly considered it in their protocols (e.g., through using some mechanisms that can be extended to tolerate this perturbation). Table II summarizes the perturbations covered by each protocol and the qualitative level of each protocol related to considering

and handling these perturbations. The two first rows follow from the detailed description of the FT-PPTC and FT-PPTC-Rec in Sections 6.3 and 6.4. For the M-2PC, CO2PC, TCOT and UCM protocols, we followed an optimistic evaluation since none of these protocols comprehensively tackle the perturbations in the mobile environment apart from network disconnections barely investigated by part of them as shown in this table. FT-PPTC-Rec is the only protocol designed to handle CO failures. The comprehensive handling of heterogeneity, communication and node failures distinguishes FT-PPTC-Rec and demonstrates its superiority for perturbation-resilience.

7.1.2 *Message Complexity.* We denote by e the number of timeout extensions of P-MNs and by e_{all} the number of all timeout extensions (including those of P-FNs in case of TCOT). We denote by c_a the additional costs in terms of messages for the execution of compensating transactions in case such transactions are initiated. m_p represents the number of P-MNs and f_p the number of P-FNs. We denote by n_{ext} the number of messages sent by the MH-Ag to extend the timeout of its corresponding P-MN in case of predictable and unpredictable network disconnection as discussed in Section 5.4. We compute the message complexity of M-2PC, TCOT and UCM based on the message complexity analysis done in [Nouali et al. 2005; Bobineau et al. 2000]. We refer to Fig. 7 and Fig. 8 to compute the message complexity of PPTC and FT-PPTC respectively. It follows that each P-MN sends two messages to the CO and receives one message from it starting from the point in time, where the votes are sent (only the I-MN sends one message and receives one). Whenever a P-MN needs to extend its timeouts (E_t and S_t) it sends an extra message to the CO. Table III does not include the message complexity of FT-PPTC-Rec since it is equal to that of FT-PPTC.

Table III. Message complexity

Protocol	**Atomicity**	**Phases**	**Wireless message complexity**	**Overall message complexity**
PPTC	*Strict*	*2*	$(2+1)*m_p-1+e$	$\{3m_p-1+e\}+\{4f_p\}$
FT-PPTC	*Strict*	*2*	$(3+1)*m_p-1+e$	$\{4m_p-1+e\}+4f_p+n_{ext}$
M-2PC	Strict	2	$(2+2)*m_p-1$	$\{4m_p-1\}+\{4f_p\}$
CO2PC	Semantic	2	$(2+1)*m_p-1$	$\{3m_p-1\}+\{4f_p+c_a\}$
TCOT	Semantic	1	$2*m_p-1+e$	$\{2m_p-1\}+\{2f_p+e_{all}+c_a\}$
UCM	Strict	1	$(1+1)*m_p$	$\{2m_p\}+\{2f_p\}$
2PC	Strict	2	$4*m_p$	$\{4m_p\}+\{4f_p\}$

Table III details the efficiency of PPTC compared to other protocols while providing strict atomicity. We emphasize that PPTC's efficiency is comparable to classical protocols, even though it allows for fully mobile participants. Because of its importance in mobile environments we will investigate the wireless message complexity of PPTC and FT-PPTC using simulations in the next section. This investigation aims at showing the impact of the number of timeout extensions e on the overall message complexity of the evaluated protocols.

7.2 Simulations and Experiments

We present our simulation results for the common mobile system classes identified in Section 6. We further provide experimental results for some failure-free scenarios as imple-

mented using a setup of multiple PDA's connected to a base laptop. The experimental results were used majorally to calibrate our simulator. Experiments in failure-prone scenarios are planned for future work. We compare the performance of the PPTC protocol to that of M-2PC and the conventional 2PC. A comparison to CO2PC, TCOT and UCM is omitted as TCOT and CO2PC do not guarantee strict atomicity and due to the lack of implementation details of UCM. Additionally, we highlight the fault-tolerance improvements provided by our various building blocks implementing the developed FT techniques.

7.2.1 *Simulation Settings.* For simulation studies, we have used SimJava [SimJava 2004], a discrete event-based simulator. We conducted our simulations using confidence intervals of 95%. For the evaluation of our protocols, we consider a wide range of parameter values to highlight the generality of our results. Table IV summarizes our simulation parameters. Disconnection rate is the ratio of time where the P-MN is disconnected from the network to the total simulation time. In some of our simulations the CO does not receive an estimation of the lifetime of the MT from the initiator. In this case we say lifetime is undefined (we denote it by "Lifetime=UNDEF" in the Figures). For execution times and transmission delays we used uniform distributions. The mobility of MNs is implicitly considered in our simulations through considering its impact of disconnection and reconnection of MNs from/to base stations. The correlation between mobility and disconnection rate is given as we are investigating only infrastructure-based mobile environments.

Table IV. Simulation parameters and settings

Parameter	**Value(s)**
#P-MNs	[1,10]
#P-FNs	[1,4]
Heterogeneity of P-MNs	Laptop, PDA, Cell phone
Fragment execution time (P-MN)	[0.3,0.4], [0.5,0.6], [0.6,0.7] s
Fragment execution time (P-FN)	[0.1,0.3] s
#Fragments per MT	n ∈ [3,15]
Heterogeneity of links	WLAN, UMTS, GSM
Transmission delay (wireless link)	[0.2,0.4], [0.4,0.7], [0.6,1.0] s
Transmission delay (wired link)	[0.01,0.03] s
Disconnection rate	0%-90%

We generate transactions as follows. We assume all transactions are of different lengths and are composed of n fragments. We distribute n uniformly across all MTs. We let some MNs initiate one transaction each at the beginning of the simulation. The number and nature of P-MNs and P-FNs are randomly selected modeling arbitrary heterogeneity. Additionally, we implemented all the timeout strategies we developed in our framework not only for the proposed family of protocols but also for all the protocols we compared our protocols with. Our goal was to enhance the competitiveness of these protocols compared to the new ones developed in this work.

7.2.2 *Simulation and Experimental Results in Failure-free Scenarios.* For the performance analysis of transactional systems, throughput and resource blocking time are the commonly used performance metrics. We define *throughput* as the number of successfully committed MTs per time unit, and the *resource blocking time* as the time interval, where the resources at the *fixed participants* remain locked. Resource blocking time starts when

the participant sends its vote to the CO and ends when the participant receives the final decision about the outcome of the MT. Resource blocking times at mobile participants are not evaluated since the data stored on them is not critical and we do not assume that it is queried frequently as compared to the data stored on fixed participant nodes.

Throughput. We first compare the throughput of PPTC, M-2PC, and 2PC. Fig. 9 shows the throughput over the number of initiated transactions. We observe that throughput of PPTC is comparable to 2PC and M-2PC despite the fact that PPTC decouples the execution of fragments of P-MNs and P-FNs, whereas these fragments are executed in parallel in 2PC and M-2PC. This observation can be explained by the fact that the execution time of transaction fragments and communication delays in wired networks are considerably smaller than those in wireless networks. The time needed to execute 2PC in wired networks can be almost neglected as compared to the time needed to collect votes from P-MNs. This applies almost for any other transaction commit protocol used instead of 2PC in the wired network.

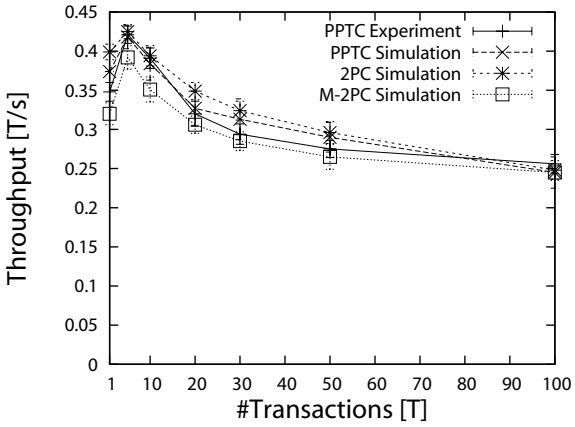

Fig. 9. Throughput

We implemented the PPTC protocol on a testbed consisting of multiple PDAs and one laptop, which use WLAN to communicate with the BS. The throughput is comparable to the results obtained from simulations because we used our prototype implementation to calibrate our simulator. Overall PPTC displays a stable performance behavior that is similar to the behavior of the traditional 2PC protocol. This is significant given that the effect of mobile nodes is shown to be minimal for the commit operations. It also validates the effectiveness of our split two-phase approach, where the impact of the decoupling in PPTC on the performance is minimal. The matching traces of Fig. 9 highlight that the throughput of PPTC (both in simulations and experiments) and of M-2PC is competitive to 2PC. This is intuitive as our simulations here were for the failure-free scenario which is similar in some extent to a wired scenario where 2PC is actually best suited for. This also illustrates the negligible overheads of PPTC and M-2PC (as compared to 2PC) for comparable failure-free cases.

Resource Blocking Time. Fig. 10 depicts the blocking time over the number of initiated transactions. PPTC shows a significantly lower blocking time of the resources at P-FNs due to the decoupling of the commit of P-MNs from that of P-FNs. This decoupling makes the resource blocking time in PPTC dependent *only* on the time needed by P-FNs to execute their corresponding fragments, which is considerably shorter than the time needed by P-MNs. The results of our simulation model are also validated by the conducted experiments.

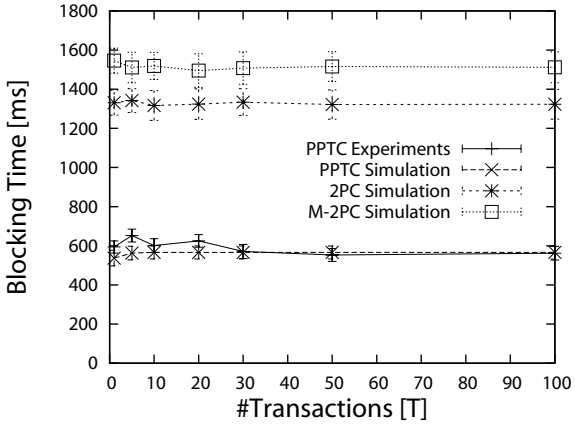

Fig. 10. Resource blocking time at FNs

7.2.3 *Heterogeneity.* To evaluate the impact of heterogeneity of nodes and links on PPTC, the impact of the choice of timeouts on the commit rate is investigated.

Impact of P-MNs Timeouts on Commit Rate. We conducted simulations and experiments with PPTC varying heterogeneity of links to observe how the choice of the timeout impacts the commit rate in scenarios where lifetime is undefined. Fig. 11 depicts the results of the experiments we conducted to select an optimal timeout and those of simulations. These results show the existence of such a value for a certain given scenario.

In the case of a homogeneous network (see the curves for GSM, UMTS and WLAN networks in Fig. 11), there is a threshold after which the commit rate is 100%. This shows a deterministic behavior. In the case of a heterogeneous network (Fig. 11) we observe an oscillation before stabilizing. This oscillation is due to the difference of delays in the different networks and also due to the difference of computing capabilities of the heterogeneous P-MNs which is modeled by different fragment execution times in this simulated scenarios (Table IV). If the fastest P-MN initiates the transaction and the rest of the P-MNs are considerably slower, the timeout sent to the CO by the initiator will be short in comparison to the time needed by the timeouts of the rest of the participants to reach the CO. The timeout set by the CO therefore expires before receiving any timeout from the rest of the participants and the CO aborts the transaction only due to lack of timeout estimation (Fig. 3). Simulation results show that if an appropriate lifetime is defined, no oscillations result. The choice of the lifetime is discussed further for the failure-prone scenario when the impact of lifetime on commit rate is investigated.

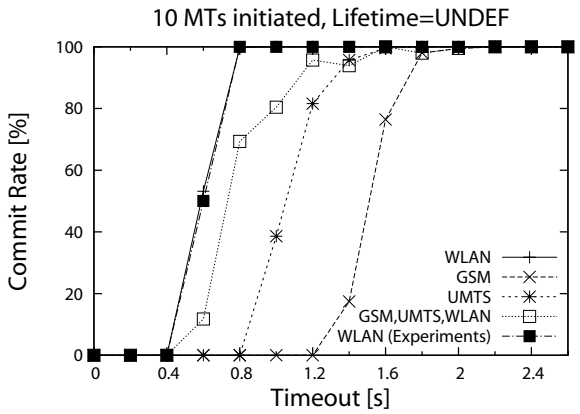

Fig. 11. Optimal timeout selection

7.2.4 *Failure-Prone Scenarios.* We analyze failure-prone scenarios using FT-PPTC and concentrate on four aspects: *(1)* the impact of P-MNs connectivity on commit rate, *(2)* impact of lifetime on commit rate, *(3)* the impact of P-MNs connectivity on P-FNs resource blocking times, and *(4)* the impact of P-MNs connectivity on MT execution time.

Impact of P-MNs Connectivity on Commit Rate. Fig. 12 shows that a disconnection rate of 20% leads to the abort of almost 65% of the initiated transactions if we use PPTC (or M-2PC) which is intolerable in the considered type of environments. To improve the resilience of the protocol to such failures, FT-PPTC deploys MN-Ags which are able to take some decisions on behalf of their corresponding P-MNs like extension of timeouts if the P-MN gets disconnected. This shows a considerable improvement in the commit rate where the commit rate goes below 90% only if the disconnection rate is more than 80%. This considerable improvement comes with only a small cost of additional wireless message overhead as shown in Fig. 13.

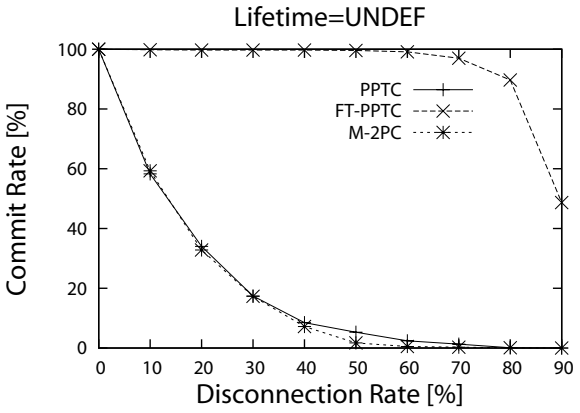

Fig. 12. Impact of connectivity on commit rate

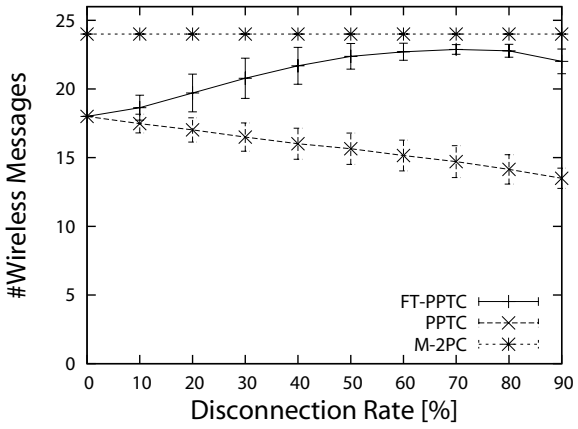

Fig. 13. Wireless messages overhead

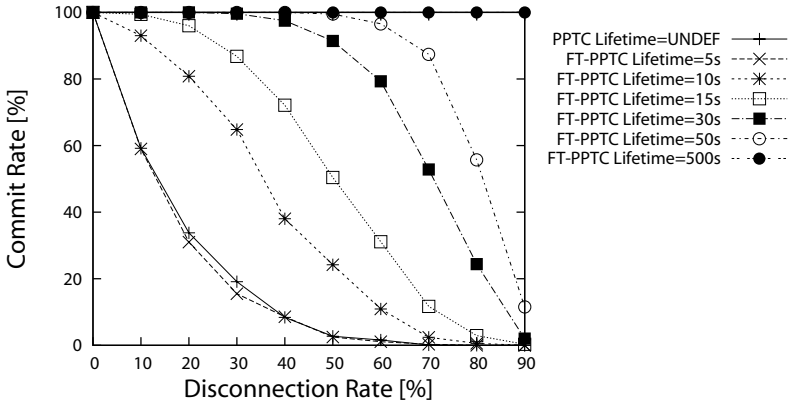

Fig. 14. Impact of lifetime on the commit rate

Impact of Lifetime on Commit Rate. Fig. 14 illustrates the impact of increasing the lifetime of the MT on the commit rate. When the lifetime value is small and is in the range of the timeouts of P-MNs, FT-PPTC shows a similar behavior to PPTC. When the lifetime increases the commit rate also increases since the P-MNs are now more likely to be connected within the lifetime and therefore able to successfully execute their fragments and to inform the MN-Ags.

Impact of P-MNs Connectivity on P-FNs Blocking Time. Fig. 15 shows that the blocking time of P-FNs in FT-PPTC is not affected by the disconnection rate of P-MNs. However, in M-2PC the disconnection rate influences the blocking time of P-FNs since P-FNs have to wait for P-MNs as all participants execute their fragments in parallel. This demonstrates that FT-PPTC is highly resilient to disconnections of P-MNs as resource blocking times

of P-FNs remain constant. We emphasize that resource blocking time of FT-PPTC is independent from the number of mobile participants. This highlights the scalability of our approach.

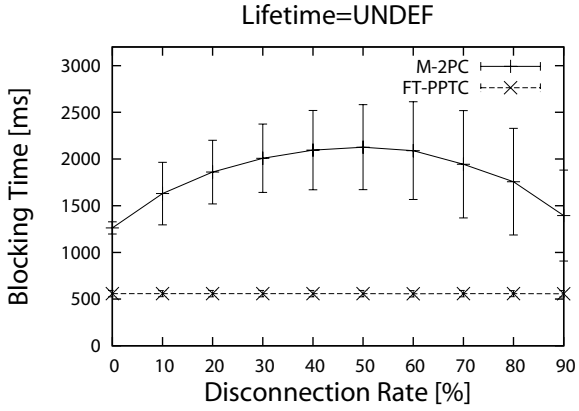

Fig. 15. Impact of connectivity on blocking time of P-FNs

Impact of P-MNs Connectivity on MT Execution Time. We also investigated in our simulations the MT execution time, which is the time interval between the initiation of the MT and the time when the final decision is received by the last participant. Fig. 16 shows that the decoupling does not affect the execution time considerably because the time needed to execute MT fragments on MNs and to exchange wireless messages is considerably larger than the time needed to execute MT fragments on FNs and to exchange wired messages. We conclude that the overhead in terms of MT execution time due to decoupling is minor in comparison to the gains obtained by using this strategy in terms of perturbation resilience and reduction of resource blocking time of P-FNs. We observe in Fig. 16 that the MT execution time of M-2PC is affected by the disconnection of P-MNs similarly to FT-PPTC. This is due to the fact that we implemented all the timeout strategies for both FT-PPTC and M-2PC.

7.2.5 *Discussion.* We showed in our studies the utility of the different building blocks we developed in our framework. We demonstrated how these blocks can be combined during design time to adapt the atomic commit protocol to the characteristics and perturbations of the considered mobile environment. Our studies highlighted also the modularity of our approach which can be extended to satisfy the requirements of other mobile environments by identifying new buildings blocks and integrating them to the framework. The protocols developed in this work outperform existing solutions with respect to efficiency and fault-tolerance. This is achieved by adding a tolerable overhead in terms of message complexity and transaction execution times. The prototype implementation of the protocol [Ayari et al. 2008] helped us in calibrating and validating our simulation environment.

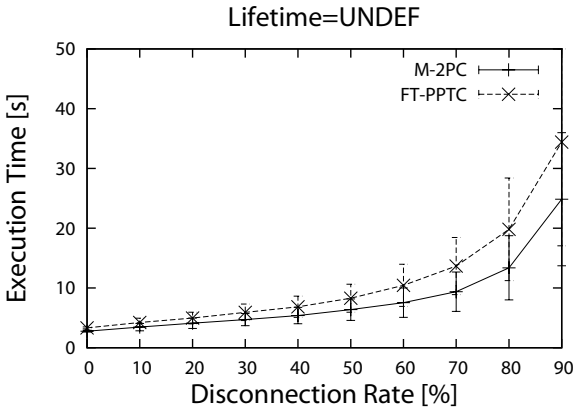

Fig. 16. MT execution time

8. CONCLUSION AND FUTURE WORK

In this paper, we have presented a modular framework for perturbation-resilient atomic commit protocols for mobile transactions in infrastructure-based mobile environments. We provided a comprehensive classification of the commit perturbations in mobile environments which simplifies the design of atomic commit protocols. For each identified class of perturbation, we designed a set of efficient techniques. Next a family of progressive mobile transaction commit protocols (PPTC, FT-PPTC, FT-PPTC-Rec) for three common classes of mobile systems was presented. This family of solutions is extendable and adaptable in its nature allowing the designer to customize protocol resilience to a subset of all possible perturbations based on the overall costs involved such as message complexity and charges for bandwidth utilization. Using simulations and experiments, we have also demonstrated the efficiency, scalability and level of perturbation-resilience added by using our proposed FT techniques.

Our future plan is to enhance the communication model by allowing for ad-hoc communication between the mobile devices, and to accordingly extend the commit framework.

REFERENCES

ALONSO, R. AND KORTH, H. F. 1993. Database system issues in nomadic computing. In *Proceedings of ACM SIGMOD International Conference on Management of Data*. 388–392.

AYARI, B., KHELIL, A., AND SURI, N. 2006. FT-PPTC: An efficient and fault-tolerant commit protocol for mobile environments. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS)*. 96–105.

AYARI, B., KHELIL, A., SURI, N., AND BLEIM, E. 2008. Implementation and evaluation of delay-aware and fault-tolerant mobile transactions. In *Proceedings of the the 2nd International Conference on E-Medical Systems (E-MediSys)*.

BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

BOBINEAU, C., LABBÉ, C., RONCANCIO, C. L., AND SERRANO-ALVARADO, P. 2004. Comparing transaction commit protocols for mobile environments. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications*. 673–677.

BOBINEAU, C., PUCHERAL, P., AND ABDALLAH, M. 2000. A unilateral commit protocol for mobile and

disconnected computing. In *Proceedings of the 12th International Conference on Parallel and Distributed Computing and Systems (PDCS)*.

BÖTTCHER, S., GRUENWALD, L., AND OBERMEIER, S. 2007. A failure tolerating atomic commit protocol for mobile environments. In *Proceedings of the 8th International Conference on Mobile Data Management (MDM)*. 158–165.

CHRYSANTHIS, P. K. 1993. Transaction processing in a mobile computing environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems (APADS)*. 77–82.

DUNHAM, M. H., HELAL, A., AND BALAKRISHNAN, S. 1997. A mobile transaction model that captures both the data and movement behavior. *Mobile Networks and Applications 2,* 2, 149–162.

ELNOZAHY, E. N. M., ALVISI, L., WANG, Y.-M., AND JOHNSON, D. B. 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys 34,* 3, 375–408.

GARCIA-MOLINA, H. 1983. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems 8,* 2, 186–213.

GRAY, J. 1978. Notes on data base operating systems. In *Operating Systems, An Advanced Course*. 393–481.

GRAY, J., HELLAND, P., O'NEIL, P., AND SHASHA, D. 1998. The dangers of replication and a solution. 372–381.

GRAY, J. AND LAMPORT, L. 2006. Consensus on transaction commit. *ACM Transactions on Database Systems 31,* 1 (March), 133–160.

HAERDER, T. AND REUTER, A. 1994. *Principles of transaction-oriented database recovery*. Morgan Kaufmann Publishers Inc.

IBM DB2 2009. IBM Software Products, DB2 Product Family. http://www-01.ibm.com/software/data/db2/.

IBM DB2 Everyplace 2009. IBM Software Products, DB2 Everyplace. http://www-01.ibm.com/software/data/db2/everyplace/.

KARLSEN, R. 2003. An adaptive transactional system - framework and service synchronization. In *Proceedings of the 5th International Symposium on Distributed Objects and Applications (DOA)*. 1208–1225.

KU, K.-I. AND KIM, Y.-S. 2000. Moflex transaction model for mobile heterogeneous multidatabase systems. In *Proceedings of the 10th International Workshop on Research Issues in Data Engineering (RIDE)*. 39.

KUMAR, V. 2000. A timeout-based mobile transaction commitment protocol. In *Proceedings of the East-European Conference on Advances in Databases and Information Systems*. 339–345.

KUMAR, V. AND DUNHAM, M. H. 1998. Defining location data dependency, transaction mobility and commitment. TR 98-CSE-1, Southern Methodist Univ. February.

KUMAR, V., PRABHU, N., DUNHAM, M. H., AND SEYDIM, A. Y. 2002. TCOT-A timeout-based mobile transaction commitment protocol. *IEEE Transactions on Computers 51,* 10, 1212–1218.

MADRIA, S. K. AND BHARGAVA, B. 2001. A transaction model to improve data availability in mobile computing. *Distributed Parallel Databases 10,* 2, 127–160.

NOUALI, N., DOUCET, A., AND DRIAS, H. 2005. A two-phase commit protocol for mobile wireless environment. In *Proceedings of the 16th Australasian Database Conference (ADC)*. 135–143.

NOUALI-TABOUDJEMAT, N., BOUKANTAR, L., AND DRIAS, H. 2007. Performance evaluation of atomic commit protocols for mobile transactions. *International Journal of Intelligent Information and Database Systems 1,* 2, 122–155.

NOUALI-TABOUDJEMAT, N. AND DRIAS, H. 2008. A policy-based context-aware approach for the commitment of mobile transactions. In *Proceedings of the 8th international conference on New technologies in distributed systems (NOTERE)*. 1–11.

OBERMEIER, S., BÖTTCHER, S., AND KLEINE, D. 2008. CLCP-A distributed cross-layer commit protocol for mobile ad hoc networks. In *Proceedings of the 6th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. 361–370.

Oracle Database Lite 2009. Oracle Corporation, Oracle Database Lite 10g: The Internet Platform For Mobile Computing. http://www.oracle.com/technology/products/lite/index.html.

Oracle Database Standard Edition 2009. Oracle Corporation, Oracle Database 11g Standard Edition. http://www.oracle.com/database/standard_edition.html.

PITOURA, E. AND BHARGAVA, B. 1995. Maintaining consistency of data in mobile distributed environments. In *Proceedings* 15^{th} *ICDCS*. 404–413.

PITOURA, E. AND BHARGAVA, B. 1999. Data consistency in intermittently connected distributed systems. *IEEE Transactions on Knowledge and Data Engineering 11,* 6, 896–915.

POPOVICI, A. AND ALONSO, G. 2002. Ad-hoc transactions for mobile services. In *Proceedings of the 3rd International Workshop on Technologies for E-Services*. 118–130.

PRADHAN, D., KRISHNA, P., AND VAIDYA, N. 1996. Recovery in mobile wireless environment: Design and trade-off analysis. In *Proceedings of the 26th International Symposium on Fault-Tolerant Computing (FTCS)*. 16–25.

SERRANO-ALVARADO, P. 2004. Transactions adaptables pour les environnements mobiles. Ph.D. thesis, Joseph-Fourier University, Grenoble, France.

SERRANO-ALVARADO, P., RONCANCIO, C., AND ADIBA, M. 2004. A survey of mobile transactions. *Distributed Parallel Databases 16,* 2, 193–230.

SERRANO-ALVARADO, P., RONCANCIO, C., ADIBA, M., AND LABBÉ, C. 2003. Adaptable mobile transactions and environment awareness. In *Proceedings of the 19èmes Journées Bases de Données Avancées (BDA)*.

SERRANO-ALVARADO, P., RONCANCIO, C., ADIBA, M., AND LABBÉ, C. 2004. Context aware mobile transactions. In *Proceedings of the 5th International Conference on Mobile Data Management (MDM)*. 167.

SERRANO-ALVARADO, P., RONCANCIO, C. L., ADIBA, M., AND LABBÉ, C. 2005. An Adaptable Mobile Transaction Model for Mobile Environments. *International Journal Computer Systems Science and Engineering – Special issue on Mobile Databases, 20, 3*.

SimJava 2004. The SimJava discrete event-based simulator. http://www.dcs.ed.ac.uk/home/hase/simjava.

SKEEN, D. AND STONEBRAKER, M. 1983. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering 9,* 3, 219–228.

WALBORN, G. AND CHRYSANTHIS, P. 1995. Supporting semantics-based transaction processing in mobile database applications. In *Proceedings of 14th Symposium on Reliable Distributed Systems (SRDS)*. 31–40.

WALBORN, G. D. AND CHRYSANTHIS, P. K. 1999. Transaction processing in pro-motion. In *Proceedings of ACM symposium on Applied computing (SAC)*. 389–398.

XIE, W. 2005. Supporting distributed transaction processing over mobile and heterogeneous platforms. Ph.D. thesis, Georgia Institute of Technology, Georgia, USA.

YEO, L. H. AND ZASLAVSKY, A. 1994. Submission of transactions from mobile workstations in a cooperative multidatabase processing environment. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)*. 372 – 379.