Establishing Multi-level Security in Mobile Data Access

Muhammad Mukaram Khan[1] And Constantinos Papadopoulos*,[2]

[1]*School of Computer Science, The University of Manchester, UK*

[2]*General Secretariat for Information Systems, Ministry of Economy and Finance, Greece*

Abstract: Transaction processing over mobile networks faces new challenges due to limitations in bandwidth and available power, as well as due to intermittent connectivity that causes loss of data and transaction aborts. Besides, the possibility of security breach increases substantially due to the frequent motion of clients across cells, which gives rise to novel forms of covert channels. In this paper, we first investigate to what degree this breach may occur and we also assess the suitability of existing protocols for avoiding the appearance of covert channels in mobile database access. Based on the discovery of certain vulnerabilities in these protocols, we propose an optional multi-granularity locking protocol that ensures secure access to shared data in mobile environments without compromising their consistency or the atomicity of transactions.

Keywords: Mobile networks, multi-level database security, covert channels, mobile transactions, locking protocols, multiple granularity locking.

1. INTRODUCTION

Database transaction processing conforms for several years now to the criteria of atomicity, consistency, isolation and durability (ACID). Techniques like two-phase commit (2PC) and locking, in turn, are used by almost every transaction to achieve the atomicity and isolation properties and preserve the consistency of shared data. The application of these techniques to centralized and distributed database systems has been thoroughly discussed in [1]. 2PC ensures that a transaction either commits (if all its update operations have been successfully completed) or aborts and its intermediate effects on all affected sites are obliterated as if that transaction had never executed. 2PC comprises two phases, i.e., voting and commit. Even if only one participant votes '*no*', the whole transaction is aborted in favour of consistency and atomicity. In distributed computing, factors like message complexity due to network delays, log overloading due to I/O delays, and time complexity of a transaction with regard to reaching a decision may lead to a substantially long execution time, causing thereby problems in preserving the atomicity property [2]. Two-phase locking (2PL), in turn, is a standard technique for coordinating concurrent operations that maintains the integrity and consistency of data by prohibiting conflicting updates from occurring simultaneously [1, 3].

Various types of locks and different levels of lock granularity can be used to ensure the isolation property and preserve data consistency, while improving at the same time transaction throughput. For example, multiple-granularity locking [4, 5] allows a single transaction to access different parts of a database at the same time by setting multiple locks on these parts simultaneously, with each lock covering objects of different sizes. This is achieved with the use of both actual and *intentional* locks, which cover an object along with some (or all) of its descendants (i.e., components) in the object hierarchy. Multiple-granularity locking can dramatically reduce the access time of transactions, as it is shown in the literature by a number of simulation experiments (more details on this issue are presented in [5]). Optional locking [3], on the other hand, allows a transaction to set a *tentative* lock on an object and start updating the object without having requested a permission for that. If another transaction wants then to update the same object (while the first transaction is still running), the locking protocol checks its priority and, if it is higher than the priority of the currently executing transaction, the latter is suspended and the results it has produced till then are *undone*.

1.1. Mobile Transaction Processing

In mobile computing environments, transaction processing faces new challenges due to typical characteristics of wireless networks such as low bandwidth, frequent disconnections by mobile hosts (MH), very low processing power as well as limited storage capacity of the mobile devices. A survey of database operations in mobile environments points out that disconnections are frequent when clients are roaming, as they disconnect from a cell to connect to another [6]. In this paper, we assume that during the execution of a transaction a MH experiences short intermittent disconnections (like the above) rather than long ones, which occur instead when a client switches off. Moreover, we adopt the assumption of [7] that handoff delays pose a severe challenge for database transactions, hence we recognize the need for a novel transaction model to counter their effects. In addition, the mobile devices that are used today operate as I/O and communication devices primarily with low processing capabilities and battery life, while they rely on proxies working on their behalf and residing at their mobile-support station (MSS) of the current cell. A novel model for transaction execution in such environments may not use the traditional techniques of 2PC and 2PL, as transactions would only get a small fraction of useful work done due to frequent aborts which owe to network disconnections. An effort towards this direction defines such a model (so called Kangaroo Transac-

*Address correspondence to this author at the General Secretariat for Information Systems, Ministry of Economy and Finance, Greece; E-mail: k.papadopoulos@gsis.gr

1874-1339/07

tions [8]) by building upon the concepts of split and global transactions, which ensures the successful execution of transactions despite the occurrence of handoffs. Following this model, a number of solutions have been proposed by other authors [9, 10, 11] that address issues related to roaming, disconnections, data availability and transaction throughput.

1.2. Secure Transaction Processing in Mobile Environments

Transaction processing in safety-critical installations (like military and government agencies) requires support of multi-level database security (MLDBS) so as to enforce security at various levels of classification and user clearances, especially in a real-time environment which is crucial to the success of such installations. Typically, the security aspect is looked after by the Bell-LaPadula (BLP) security model [12, 13, 14, 15]. Yet this model is not sufficient to protect from covert channels, which are indirect means whereby a high-security transaction may transfer information to a low-security one. In fact, due to the way that 2PC and 2PL operate, it is possible to allow a malicious user (or a Trojan horse [14]) to seamlessly pass on information from a high level of classification to lower one(s). The chances for establishing covert channels increase when transactions execute in a mobile environment, due to intermittent disconnections and the movement of mobile devices across different cells (where they re-establish connection using seamless handoff methods), as well as due to the weaknesses of the 2PC and 2PL techniques in maintaining atomicity and consistency. The authors of [12] and [14] for example, provide solutions to ward off covert channels in fixed networks, but these solutions are unsuitable for establishing MLDBS in mobile networks. In an effort to address this deficiency, we incorporate in this paper an optional form of multiple-granularity locking into an MLDBS protocol so as to eliminate overt and covert channels from mobile transactions.

1.3. Outline of Paper

While combining the requirements of security and mobility, we are also concerned with other relevant issues like concurrency and performance (of an individual transaction and an entire system too). Fig. (**1**) below shows the effect of these factors on each other within the context of a mobile transaction. The dotted arrows denote undefined effects for which different views can be presented (we keep this discussion out of the scope of this paper). Though concerned with the security of shared data during the execution of mobile transactions, we are conscious not to do this at the cost of reduced concurrency and degraded performance; rather, our solution aims at improving these two factors through the use of optional locking at multiple data granularities, which can also eliminate the chances of covert channel establishment.

In Section 2 we review existing work on mobile transaction processing that is relevant to our own work (a more elaborate analysis can be found in [16], which presents several details that fall beyond the scope of our paper). In that section we also consider certain aspects of database security which depend on locking mechanisms and concurrency control. Based on the conclusions drawn in this review, we consider in Section 3 the possibilities of establishing covert channels in mobile environments, while in Section 4 we propose a novel protocol for avoiding them which is based on locking. Although locking is in general considered inappropriate for mobile data management [10], its *optional* character in the proposed protocol does not degrade transaction throughput but increases security and improves usability in database access. Section 5 evaluates this protocol and points out its strengths through a real scenario, while Section 6 summarizes the paper's results and presents an outlook of our future objectives.

2. RELATED WORK

2.1. Mobile Transaction Processing

Dunham et al. [8] introduced in the nineties the aforementioned concept of Kangaroo Trans-actions for multi-hop mobile environments, which was the first that incorporated into transaction scheduling both client and data movements. In that concept, there is a Data Access Agent (DAA) at the base station which creates a mobile transaction and, as a mobile client hops from one cell to another, the control of the

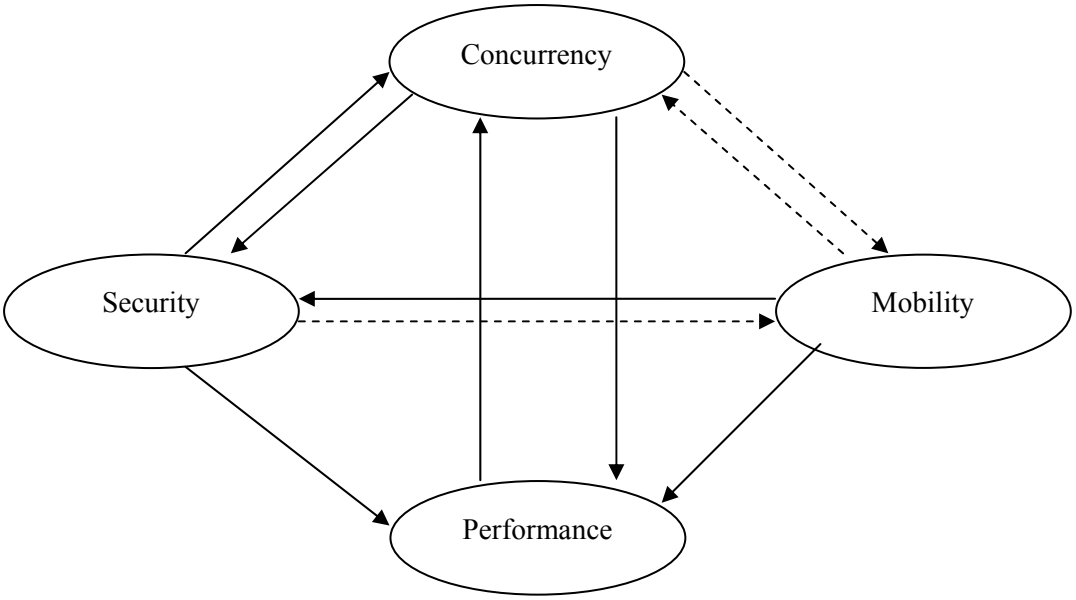

Fig. (1). Interdependence among various factors in mobile transaction processing.

KT is passed to the DAA at the base station of the new cell, where it creates a new subtransaction called 'Joey Transaction' (JT). This subtransaction is considered as the unit of transaction execution at each base station. Old JTs commit independently of new ones in the same station. At any time though, the failure of a JT may cause the entire KT to be aborted. A KT can work in compensating mode or in split mode. In the first one, the failure of any JT causes the currently executing JT as well as all the preceding and the following JTs to be undone (that is, previously committed JTs will have to be compensated for). The problems with KT are that serialization is not always guaranteed, global isolation is not enforced (hence global consistency cannot be attained), while disconnection due to failures and interference is not considered at all.

The semantics of transaction processing (and the preservation of the above properties too) had been considered in an earlier work [11], in which Walborn and Chrysanthis had extended semantic-based models to mobile environments in order to increase the concurrency among operations and to leverage transaction performance. Specifically, the exploitation of commutative operations and the maintenance of multiple database replicas along with the fragmentation of data objects in mobile environments were shown sufficient to achieve serialization and preserve isolation. Yet each fragmented object must be cached independently and manipulated synchronously. According to Walborn and Chrysanthis [11], that scheme performs well in situations where data objects are fragmented into stacks or queues.

The preservation of data consistency in mobile environments was dealt with at the same time by Lu and Satyaranayanan as well, who presented a model based on Isolation-Only Transactions (IOT) [17]. That model was influenced by Optimistic Concurrency Control (OCC) schemes that use client caches for local transaction execution. IOT include a sequence of access operations to the database and they conform to a set of properties that are specifically tailored to disconnections in mobile environments. Moreover, they perform automatic detection of read/write conflicts based on certain serializability constraints. A variety of conflict-resolution mechanisms are supported, including a mechanism employing application semantics as well as an invocation mechanism that executes transactions by itself or through an Application Specific Resolver (ASR). Unlike regular transactions that preserve the ACID properties, IOT do not guarantee the atomicity of updates and can preserve durability under certain conditions only. Besides, the relevant model is exclusively tailored to the Unix file system to access data.

A radically different approach to mobile transaction processing was proposed one year later by Gray *et al.* [18], which uses 2-tier replication to allow MH to execute transactions in disconnected mode. The database is replicated at both the MH and the base nodes, and the MH maintains both a local and a master version. The master version of each object may coexist with several replicated versions. Two types of transactions (i.e., 'base' and 'tentative') are supported. During disconnection, the MH may perform tentative updates to objects owned by other nodes in the network, while on reconnection, the MH forwards these updates to the owner nodes, where they are re-executed in order to commit or to be rejected (they may also commute with other transactions to improve the chances of commitment). After this exchange, the base transactions execute by accessing master versions, whereas the tentative ones execute by accessing tentative versions (i.e., local copies). Moreover, on reconnection the various base stations (BS) re-execute tentative transactions as base ones in order to achieve global consistency and make local updates persistent.

In the model proposed by Chrysanthis [19], in turn, mobile transactions are represented as sets of open nested subtransactions using the notions of 'reporting' and 'co-transaction'. As shown in [19], this model guarantees the atomicity and non-compensability of mobile transactions. While in execution, a 'global' mobile transaction can share partial results on MH and partially maintain its state on BS. Each subtransaction is atomic but does not guarantee the atomicity of the global one. Reporting transactions and co-transactions share their partial results, retain their state and can follow their associated transaction which executes on a MH (by relocating from one MSS to another along the path to the MH). This model allows transactions to be executed even in periods of disconnection and it supports the unilateral commit of subtransactions and compensating transactions. Yet not all the operations of a transaction are compensated, since compensation is costly in mobile environments.

To increase the availability of data in mobile and stationary hosts, Madria and Bhargava [20] introduced at the same time the concept of *pre-write* operations that are supposed to execute before regular write operations. A pre-write operation makes visible the value of a data object after the commit of the transaction. Once all the pre-writes have been processed, the mobile transaction 'pre-commits' on a mobile host. The results of a pre-committed transaction are visible at mobile and stationary hosts before the final commit. This minimizes the blocking probability for other transactions and increases also concurrency. The transaction continues its execution on the mobile host by announcing pre-write values and by delegating the resource-consuming part of the execution (i.e., updates on the disk) to the stationary host (reducing thereby the computation cost on the former). As shown in [20], a pre-committed transaction is guaranteed to commit. This feature avoids operation *undo* or transaction compensation, which are quite costly in mobile environments. A 'pre-read' operation returns a 'pre-write' value, whereas a read one returns a write value. Transactions are serialized based on their pre-commit order, dealing thus efficiently with the limitation of resources in mobile environments.

As concerns the *Pro-motion* technique that was proposed by Walborn and Chrysanthis in [21], it supports transaction execution in disconnected mode through the so called 'compacts', which enable local executions at MH. The necessary information to manage a compact is encapsulated in it. Compacts are the basic unit of caching and control. They consider the entire mobile system as a large long-lived transaction executing on the server. The management of compacts is performed by joint collaboration of a 'compact manager' at the database server, a 'compact agent' at the MH and a 'mobility manager' at the BS. The compact manager constructs the compacts and it acts as a front-end for the database server, appearing thereof as an ordinary database client. On each MH, the compact agent is responsible for cache man-

agement, transaction processing, concurrency control, logging and recovery, while the transactions generated by MH are executed locally. The mobility manager, instead, is in charge of the communication among the agents.

Another technique, i.e., *preserialization* (PS), aims to facilitate transaction processing in mobile multi-database systems (MMDBS) [22]. In this technique, all operations of a global transaction accessing the same site constitute a single-site transaction (analogous to a Joey Transaction) and they are executed as a single (local) transaction. Site transactions are executed as independent transactions by the local database system (LDBS), while the global MMDBS cannot prevent the LDBS from executing local transactions intermediately among site transactions. As all site transactions are compensable, they are committed at the LDBS prior to the decision to commit the global transaction (which consists of all site transactions), releasing thus resources in a timely fashion. In addition, all site transactions are categorized as either *vital* or *non-vital*. If neither atomicity nor isolation (A/I) have been violated, the transaction commits else it is aborted. Serializability of a global transaction can be verified by constructing a 'Partial Global Serialization Graph' (PGSG) from the serialization information of local transactions. In general, this technique places emphasis on explicitly ensuring the A/I properties of a transaction, while it assumes that the LDMSs maintain individual consistency and durability. However, A/I properties can only be ensured if all site transactions are declared as vital.

Contrary to the above schemes, Gruenwald and Banik presented in [23] a power-aware scheme for Mobile Ad-hoc Networks (MANET) which allows real-time transaction processing. Transactions have deadlines and are classified on that basis as *firm* or *soft*. Firm transactions must be aborted if they miss their deadlines and their value becomes zero. Soft transactions have two deadlines; they can still be executed after the first deadline expires, but their value decreases after the first deadline and becomes zero after the second deadline. The model is based on the concept of Large Mobile Hosts (LMH), i.e., a MH has sufficient resources while a Small Mobile Host (SMH) has few ones. A global transaction may consist of a number of subtransactions running on LMH. A subtransaction can be vital or non vital. All the vital subtransactions must succeed in order for their global transaction to succeed. The model also provides a balance in energy consumption by executing soft transactions at the LMH with the highest energy level. To address the real-time requirement, it reduces the number of transactions that miss deadlines by executing firm ones at the nearest LMH, employing two deadlines for soft transactions, and scheduling both the soft and firm ones using a real-time energy-efficient transaction scheduling algorithm. To address disconnections and migrations that cause prolonged transaction executions, the model incorporates the concepts of *toggled* and *suspended* transactions which detect the violation of the A/I properties as soon as the vital subtransactions of a transaction are completed. Moreover, toggled and suspended transactions allow disconnected transactions to remain in the system and be re-executed at a later instance, unless they obstruct the execution of other transactions.

Dynamic object clustering, finally, has been proposed in [10] to facilitate the processing of mobile transactions through weak-read, weak-write, strict-read and strict-write operations. Strict-read and strict-write have the same semantics as normal read and write operations that are invoked by transactions satisfying the ACID properties. A weak-read returns the value of a locally cached object which is written by a strict-write or a weak-write. A weak-write operation only updates a locally cached object and can become permanent upon cluster merging if the weak-write does not conflict with any strict-read or strict-write operation. The weak transactions use local and global commits. The 'local commit' is the same as 'pre-commit' while 'global commit' is the same as the 'final commit' proposed in [20]. However, a weak transaction can abort after local commit and is compensated. A weak transaction's updates are visible to other weak transactions, whereas pre-writes are visible to all transactions.

2.2. Secure Transaction Processing

As we mentioned in the introduction, the BLP model used for MDBS does not guarantee security against covert channels. Several approaches have been proposed to ward off overt channel appearance in such systems. In the following, we review some of these approaches.

Kogan and Jajodia [24], for example, have presented a concurrency control mechanism for MLSDBMS which is based on a replicated architecture. This mechanism favors the use of that architecture for leveraging performance and employs a technique for controlling concurrency, which ensures 1-copy-serializability but hides from the transactions all aspects of data replication. Moreover, this mechanism provides security against covert channels (since the information only flows from lower levels of security to higher ones) and incorporates also the techniques of 'update-projection' and 'update-report' which provide coordination among DBMS's at various levels. Specifically, no voting is done by this mechanism (unlike the 2PC protocol) but the chances of covert channel establishment are eliminated by having a transaction committing at its own database and then being passed as an update-projection onto other databases of different security levels, where it is guaranteed to commit in the same manner. Yet this mechanism runs the risk that higher-class transactions might be forced to read arbitrarily old values [24].

Ammann and Jajodia [25], on the other hand, proposed a multiversion algorithm for secure servicing of transaction read which can maintain (up to) three versions of a modified data item. Each version corresponds to the state of the data item at the end of an (externally defined) version period. Thereby, covert channels and starvations of high security-level transactions are avoided. The algorithm also allows Long-Lived Transactions (LLT) of any security level to access data by avoiding conflicts. Moreover, the algorithm ensures 1-copy serializability but it may present outdated views of several portions of a database. Besides, it has the drawback that transactions at a higher-access class are forced to read arbitrarily old values from the database due to the assignment of timestamps to data versions (this problem can be especially serious if most of the lower-level transactions are long-running ones).

The *Secure 2PL* (S2PL) technique proposed in [26] prohibits the blocking of low-security transactions by high-security ones, eliminating thus the chances of covert channel

establishment. Specifically, the authors of [26] propose the use of the so-called 'virtual lock' by low-security transactions in order to avoid conflicts with high-security ones. When a conflicting high-security transaction commits and releases a data item, the virtual lock is upgraded to a real one and the operation of the low-security transaction is allowed to execute. Another technique that was proposed in [14] improved S2PL by using its modified model for inter-level concurrency control in combination with the OPT-WAIT technique [21] for intra-level security (in fact, could not eliminate interference in all circumstances). This improvement leverages performance and enhances security, since the use of a relative slow approach (i.e., S2PL) for intra-level security is not a good idea when we are not concerned with covert channels. The problem with this approach, however, is that it cannot avoid altogether the starvation of high-level transactions.

Kim *et al.*, in turn, proposed in [27] a (transaction) length-sensitive protocol for MLSDBMS which is based on altruistic locking. This protocol introduces a new primitive called 'donate', which uses *lock* and *unlock* to signal an interested transaction that no access to a particular data item is required any more by the 'donating' transaction (making therefore this item ready for access by others and avoiding its lazy release). The protocol is meant to ensure serializability, eliminate covert channels and reduce the likelihood of starvation for Short-Lived Transactions (SLT). The work presented in [28] has extended the above protocol towards mobile transactions and enjoys a better performance compared to simple 2PL/MLS. Moreover, it improves the degree of concurrency in mobile environments (where LLT may coexist with SLT).

Finally, the nested transaction model for MLSDBMS proposed in [12] considers concurrent transactions as nested trees and provides application-level recovery along with notification-based locking protocols that ensure serializability, avoid starvation, allow concurrent execution and prohibit the appearance of timing covert channels. The model uses one more primitive 'signal lock' to allow a high-level transaction to seamlessly read a low-level data item, hence a low-level transaction can update that item even in the presence of a signal lock and remove any chances of covert channel establishment (this concept extends previous work on multilevel file storage [13]). The nested transaction model is further supported by notification functions that signal all concerned high-level transactions of a data-item's update. Besides the avoidance of starvation, the model also avoids transaction abort by reading old data values during re-execution. However, it is computationally expensive and has slow performance.

2.3. Conclusions

Existing work on mobile transaction processing has relied primarily on cached data at MH as well as on local execution and update (or commit) at these hosts. Moreover, the various models presented in the previous section deal with single security-level databases and do not account for the effects of transaction execution on multiple security levels (if these are necessary). As we pointed out earlier, these models are not suitable for all kinds of mobile devices (especially for those characterized by low computing power, short battery life and limited storage capacity). Local execution and storage is not only unsuitable for small-sized MH, but is also detrimental to data security (hence unsuitable for MLSDBS). For this reason, we advocate in this paper a transaction model that is independent of the computational power and storage capacity of MH, provides multilevel security in mobile environments and preserves also the basic properties of atomicity and consistency. Moreover, this model guarantees satisfactory performance in the presence of network disconnections.

On the other hand, the work on MLSDBMS (except that presented in [28]) is concerned with the security of data across static networks and has concentrated primarily on the elimination of timing covert channels, being thus unsuitable for mobile transactions. In fact, the latter do not run the risk of timing covert channels very often, due to the difficulty of synchronization between MHs during transaction processing. Other security concerns, however, which owe to the mobility of MHs (like, for example, *motion covert channels* that are defined below), are of prime importance to mobile transactions. For example, the frequent relocations or disconnections of mobile clients (either intentional or caused by Trojan horses) may give rise to novel forms of security breach, since they may cause variable delays in addition to the delays resulted from the intermittent connectivity of the network. Besides, the adoption of replicated architectures and the related techniques of caching and cache invalidation [6] may cause a plethora of security breaches since, for example, a low-security record in a local cache may have a different classification in the central server and incur thus an information leak.

The likelihood of the above scenarios may be reduced by the use of optional locking schemes, whereby the update of database contents can be made provisional and comply with security policies. Because this may degrade performance, however, we intend to apply optional locks at multiple levels of granularity so as to better support multiple security levels in databases (i.e., different database objects will be locked at different granules, enabling thus the assignment of a different security classification to each granule). This idea is further analyzed below.

3. COVERT CHANNELS IN MOBILE TRANSACTIONS

Security analysis of any system must take into account any violation of security policies by overt or covert channels [29]. Overt channels (i.e., those established using buffers, files or I/O devices) rely on system-protected data to directly pass on information among users or processes against the system's security policy, whereas covert channels imply the use of other entities (e.g. scheduling locks, disk arm movement, system/memory-created shared clocks or device-busy flags, etc) to pass on such information. Covert channels can be classified in two categories based on the technique they use for this malicious activity, i.e., timing covert channels and storage covert channels. A potential covert channel is a storage channel if the scenario of its use "involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of that storage location by another process" [15]. A potential covert channel is a timing channel if the scenario of its use involves a process that "signals information to another by modulating its own use of

system resources (e.g., CPU time, memory) in such a way that this modulation affects the real response time observed by the second process" [15]. Further details on the definition and the situations that establish these channels can be found in [15].

In this paper we identify another type of covert channel that does not fall in any of the above categories and has its basis on mobile transactions, due to the peculiar execution of the latter in wireless networks. We name it "motion covert channel" as its creation depends on client movements, network disconnections and the execution characteristics of mobile transactions. A motion covert channel is the type of covert channel in which some information of higher-security level (or about a higher-security subject) is passed over to a lower-security level subject who is not supposed to obtain it in a mandatory access control (MAC) system (violating thus the security policy of the whole system). The scenario of its use involves signaling of higher-security information by exploiting certain factors that affect transaction execution in a mobile environment (i.e., handoff, movement pattern of clients, temporary network disconnection, as well as local transaction execution).

Major types of information that can be passed this way are: the higher security-level data in a database (as in MLSDBS), passwords to access data with higher-level privileges, the location of a higher-level subject that may reveal (by inference) critical information about some operation or intention, the movement pattern of some higher-level subject (or a group of related subjects) that may reveal critical information on an ongoing operation, or even the location of (central or distributed) databases which, in turn, can give out the location of headquarters or centers of critical activity. The main causes of the establishment of such channels in existing mobile transaction models are: the *local cache on MH* (used for boosting performance and supporting disconnected operations), *local execution* (for supporting disconnected execution and preventing traffic congestion), the *extraction of cell information* (especially during handoff), as well as the *maintenance of status tables and logs at MH or BS*. To further elucidate the concept, we present below some scenarios:

a. In the Pre-write mobile transaction model [20] and the Kangaroo transaction model [8], while a MH moves from one cell to another, its state (i.e., the values of data objects and the location information) moves along with it. This feature can be used for establishing a motion covert channel in a way that a high-level subject can pass on information to a low-level subject by adopting a predefined path and making relevant entries in logs (or status tables) that are shared by all subjects. A low-level subject can read this information and "get the signal". One way that a stream of bits can be passed by a higher-level subject to a lower-level one is by adopting a *to-and-from movement* between two adjacent cells. To do that, a MH with high-level subjects can position itself on the boundary between two cells. We assume that the two subjects of different security levels have already synchronized on time and have decided also on fixed time intervals (say 10 to 15 minutes), during which the MH with the high-level subject (i.e., the one on the boundary of the two cells) moves to the other cell in order to create a new site transaction (i.e., JT), signaling thereby bit '1'. Otherwise (i.e., if it doesn't move) it signals bit '0'. This way, a to-and-from movement of a MH between two adjacent cells and the ensuing update of its state/location information can establish a covert channel which, in turn, can be utilized to pass on secret information (such as passwords, etc).

b. In the Kangaroo transaction model [8], once a MH requests a transaction, a KT entry is created by DAA at the corresponding BS that has a unique KID (made up of the ID of the BS and some unique number at that station). As the MH moves through different BS, new JTs are created at each BS along its route, which (i.e., the JTs) have unique IDs consisting of a KID and a unique sequence number. In this model, the information about handoff is passed over to the MH to help the creation of JTs. The DAA uses a *transaction status table* as well as a *log record* that are shared by all concurrent transactions in a BS. A malicious process of higher-security level can use the entries in these tables and/or logs (or the sequence numbers used in the creation of the KT and JT) as it moves from preselected cells to pass on secret information to a lower-level subject.

c. The Kangaroo transaction model [8] and the reporting mobile transaction model [19] are based on the principle of 'open nested transaction'. According to that principle, the component transactions (i.e., the JTs in Kangaroo transactions and compensable reporting transactions) commit independently without waiting for the global transaction to commit. In the case of abort, however, a compensating transaction cancels the effects. These partial results are available to other mobile transactions in order to increase the availability of information and upgrade the transactions' performance. This behavior can be used by a malicious higher-level transaction to pass on information covertly. If, for example, a transaction updates a data item (e.g., it makes it '1' from '0') and then that data item is locked for reading by another, higher-level transaction, bit '1' is passed. Else (i.e., if the lower-level transaction is not locked) bit '0' is passed. Then the lower-level transaction aborts and forces a compensating transaction to run. The compensating transaction will successfully execute within a shorter time period if no lock has been placed on the updated data by another (lower-level) transaction; this time difference will be noticed however by the other transaction.

4. PROPOSED PROTOCOL FOR SECURE MOBILE TRANSACTIONS

4.1. Development Rationale

As discussed in Section 2, existing models for mobile transactions are based primarily on local execution at MHs to increase performance. Moreover, these models have been designed to support conventional DBMS in mobile environments. There is a need to develop a technique that supports MLSDBS-based transaction processing, be independent of the processing power and storage capability of the MHs, does not depend on network bandwidth and is robust against

disconnections (i.e., previous work should not be wasted due to disconnection and a transaction should continue its execution despite several short disconnections). The support of MLSDBS in mobile environments has not been considered so far seriously (except in [28]). Instead, the solutions proposed so far for MLSDBS are intended for static centralized systems and thus may not adequately support mobile transactions. On the other hand, existing models for mobile transactions do not fit into MLSDBS models. In fact, previous research on MLSDBS has focused on eliminating covert channels, while in the case of mobile transactions the major concerns are movement and disconnection management. Moreover, none of the existing MLSDBS models has considered the novel challenges arising in mobile networks (like the establishment of motion covert channels).

In this section we propose a protocol that is based on a nested transaction model for mobile transactions and utilizes an optional multiple-granularity locking scheme, which assist in leveraging the performance and ensure the security of mobile transactions by eliminating any chances of timing and motion covert channels. Within the frames of this protocol we assume the existence of a distributed database system with a central repository managed by a MLSDBS, as well as a local MLSDBS at each BS that can handle cached data, manage the local execution of mobile transactions and maintain logs and status tables about each transaction. We propose moreover the use of a data cache at the BS that starts a mobile transaction; this cache holds the data acquired from the multi-granularity locking protocol on the central database. The MLSDBS at the BS maintains the cached data throughout the execution of a mobile transaction, preserving thus the security level of each one of the different transactions that execute concurrently on the same BS. Upon completion of a transaction's execution, the data is updated in the central repository and removed from the local MLSDBS.

4.2. Nested Transaction Model for Mobile Transactions that Supports MLSDBS

Since mobile transactions are characterized by frequent transitory disconnections which, in turn, may lead to aborts, we propose the use of a nested transaction model that has been used also in the Kangaroo Transaction model [8], the Reporting Transaction model [19], as well as in the OPT-WAIT technique [21] for partial redoing (i.e., redoing a transaction's portion that has been lost due to a catastrophic disconnection). By utilizing nested transactions we assume a transaction model for MLSDBS in mobile environments that is similar to the model proposed in [12]. The beauty of this model is that each subtransaction can commit independently from the others and its results can be shared among other subtransactions of the same global transaction (if those subtransactions are of the same security level). The local MLSDBS performs the local commit (as part of the DAA) for each subtransaction and the results can be shared by the subtransaction's siblings.

Contrary to the 'pre-write' approach [20], however, all the processing on the BS is done at the MLSDBS while the MH only contains a query processor and a browser (like a thin client) with no sensitive data onboard. This is done because, if a MH is captured or stolen, a minimum amount of secure information will be passed on to the enemy. In both the Kangaroo Transaction [8] and the Reporting Transaction [19] models, instead, the processing is done at the BS with subtransactions committing independently. However, there are obvious problems with these models as we mentioned earlier.

In the following we propose a locking for the nested transactions of our protocol, which makes them more robust than Kangaroo Transactions against catastrophic failures. Specifically, with this technique the global transaction does not need to abort in case of a subtransaction's (JT) failure and, moreover, while no extra traffic is incurred on the network (as it is done with Reporting Transactions).

4.3. Optional Multiple Granularity Locking

As explained in the previous section, we want to make subtransactions as independent as possible so as to minimize effort duplication (through the repetition of cancelled work) in the phase of disconnections. For this reason, subtransactions should be executing with minimum amounts of data in order to complete portions of a global transaction. Fine granularity is the best option to achieve this purpose and increase concurrency among multiple users, but it may degrade performance when a transaction requires large portions of data. To achieve both goals, we propose the use of multiple granularities at which data may be locked. Moreover, we propose the use of optional locking, as many performance and security-related issues (especially covert channels) depend on the way that locking is performed.

Specifically, we extend the multiple-granularity locking protocol presented in [5] which achieves syntactic and semantic consistency (i.e., interface and data consistency) in synchronous CSCW. This protocol can be used in mobile transactions too, as it improves their performance and guarantees the consistency of data without incurring extra network overhead (this is because MHs do not need to wait for decisions on their locking requests). Hence instead of locking with intention locks all the instances of an object class in a database or all the components of an object (or even the whole hierarchy of data objects), we may assign a *scope* to each intention lock in order to extend it to a specific component only. Multiple-granularity locking also reduces the chances of establishing covert channels, as the size of the object granule that is being read by a higher-level transaction and the granule that is being updated by a lower-level transaction may not be the same all the time. With optional locking we can further reduce the chances of establishing timing covert channels, as a request for a tentative lock by a higher-level transaction (in order to read a lower-level data item) can be granted immediately. Similarly, a tentative lock may be granted to a lower-level transaction in order to update a data item that is locked for reading by a higher-level transaction. The lower-level transaction can update of course that item, but the resulting inconsistency will be notified to all the other transactions concerned (i.e., the higher-level transactions which read that item).

However, instead of leaving the locking option at the discretion of users (as suggested in [3]), we propose a mechanism based on the 'Raise_Signal' and 'Get_Signal' primitives (which are like those presented in [12]). The only difference between the signal locking technique in [12] and the optional locking technique we propose here is that, in the former, notifications are only made to higher-level transac-

tions that set a signal lock, whereas in our own technique notifications are sent to all the affected subjects, irrespective of their security classification. However, since only the subjects of the same security level can update a data item of a particular classification (but, on the other hand, this item can be read by any higher-level security subject), notifications will be sent to the subjects of the same security level that were updating that item as well as to all higher-level subjects that are reading it. Hence since no notifications will be sent from higher-level to lower-level subjects, there are no chances to establish covert channels.

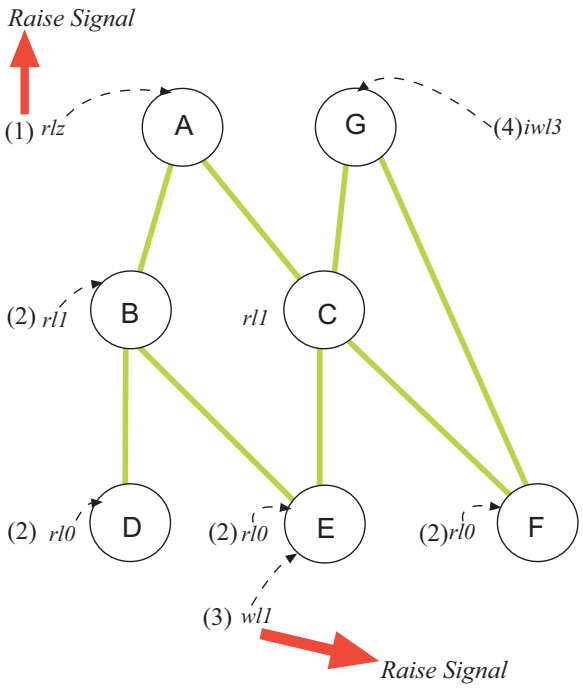

Fig. (2). Example scenario of multi-granularity locking.

Fig. (**2**) below presents a simple scenario of multi-granularity locking. A read lock of scope 2 is first applied to object *A* in the hierarchy. This locking operation has the immediate effects of locking descendant objects *B* and *C* with read locks of scope 1 and objects *D*, *E* and *F* with read locks of scope 0, respectively (*phase 2* of locking). According to our mobile transaction protocol, the locking operation upon object *A* entails a 'Raise_Signal' notification. At about the same time, a write lock of scope 1 is applied to object *E* at the bottom of the hierarchy (entailing therefore another 'Raise_Signal' notification). Following the rules of multi-granularity locking [5], appropriate intention locks should be placed then to all the "parents" above that object. However, only object *G* acquires an intention write lock of scope 3, because all other parent-objects bear read locks from the previous locking operation and disallow thus the simultaneous placement of write locks on them (since read locks are incompatible with write ones).

We deem it important to point out here that the use of optional locking in mobile transactions was necessitated by the frequent interference and disconnections in wireless mobile networks, since more than one lock may be applied concurrently to the same data item (as in Fig. **2**, for example) and the acknowledgement of these locking operations may not reach all the interested users. Another argument that justifies the use of optional locking is that, when a data object is already locked by some user, other (concurrently executing) transactions will not need to abort nor will they be restarted again (and degrade thereby performance). Besides, we propose the caching of locked data in the BS that initiates the Kangaroo transaction. While this will increase performance (as the data is readily available there), it will also reduce the chances of establishing covert channels because all the execution will take place on the cached data. In fact, since a mobile transaction can only be initiated by one user and utilizes only one cache, there is no chance for two users to establish a storage or timing covert channel (since the users work on different caches).

4.4. Protocol Rules

As mentioned earlier, our protocol relies on a nested model to support mobile transactions and assumes the existence of a distributed MLSDBS consisting of a central repository and a local MLSDBS (which, in turn, reside on BS and handle cached data via multiple logs and status tables, each one corresponding to a different security level). Moreover, our protocol takes advantage of an optional multi-granularity locking technique to reduce the chances of covert channel establishment. That technique improves moreover transaction performance. Finally, we assume a nested MLSDBS to handle the central repository.

In general, the development of our protocol has been influenced by the nested transaction model for MLSDBS [12], the Kangaroo Transaction model [8], the Pro-Motion transaction model [21], the pre-serialization technique [22], as well as the multi-granularity locking protocol proposed in [5]. The execution phases of our protocol are as follows:

4.4.1. Transaction Initiation

a. A mobile transaction works in a thin-client mode without a data cache or any kind of local execution at the MH; instead, all the execution takes place at the MLSDBS (central or at the BS). A mobile user gets involved in this process using a specific application at her/his device or by selecting in her/his browser a link to a particular server.

b. Upon connecting to that server, the user at the MH initiates the mobile transaction by issuing a combination of read/write operations on his/her authorized data (according to his/her security clearance).

c. The transaction manager (TM) at the BS will make an entry into its particular status table (based on the security clearance of the mobile user) in order to form a mobile transaction as well as the first site-transaction at that BS (which will be part of the mobile transaction). The TM then enters corresponding records in its particular status table, as it is done in the *BTKT* (*B*egin *T*ransaction of *K*angaroo *T*ransaction) and the *BTJT* (*B*egin *T*ransaction of *J*oey *T*ransaction) [8].

d. The local TM issues a lock request to the central lock manager (LM), which grants or denies it (i.e., the request) based on the availability of the relevant data. The central LM tries to grant to this mobile transaction a lock with as coarser a granularity as possible (in or-

der to increase its performance) by issuing intention-lock requests of a suitable granularity level.

e. If the requested lock is granted, the data is cached onto the DBMS of the BS where the mobile transaction was initiated from, while all the subtransactions (ST) of that mobile transaction are executed on the cached data of this BS.

f. All subtransactions (ST) generated at one BS become part of one site-transaction. Site-transactions are created as a result of a split operation which is meant to split the ongoing mobile transaction while the MH moves towards another BS. The data is cached only at the first BS where the mobile transaction started from and, if the required data is not already in that cache, an "expansion" request is forwarded to the central database. All the site-transactions under a mobile transaction are sent to the DBMS of the BS that holds the cache. All STs below a particular site-transaction are local transactions (LT), since transaction execution only takes place upon cached data.

g. Based on the request made by the MH, the local TM creates some STs to complement the current site-transaction and adds their entries to the respective status table of the current JT.

h. The TM makes also appropriate entries in the status tables and logs of each subtransaction (ST) that is saved on the stable storage, so as to support Long-Lived-Transaction execution in the phase of disconnections and be able to run compensating transactions for recovery. The creation of nested STs under each site-transaction (as well as under one mobile transaction) conforms to the nested transaction model proposed by Bertino *et al.* [12].

4.4.2. Transaction Execution

a. The execution of ST also conforms to the nested transaction model for MLSDBS ([12]), according to which execution only takes place on the leaves of the transaction hierarchy and the child transactions hand over all acquired locks to their parents. Siblings can share the updated records of each other, since a ST can deal with data of the same security level. However the transaction hierarchy (tree) will have one site-transaction below the tree's root, while all the other STs will be hung from it. We propose sequential execution at the site-transaction level only (hence transaction blocking will be done according to the nested transaction model in [12]) in order to achieve better serializability (as discussed by Bertino *et al.*). In that case, the next site-transaction cannot start before the previous one has completed.

However, the STs under one site-transaction can be executed either in blocking or in non-blocking mode (i.e., the children can be executed in parallel with their parents in the case of non-blocking to avoid performance degradation), without affecting serializability that is anyway maintained by the split operation [30]). The results of a ST will become available to its parent after commit, as well as to the outside world after commit of the root transaction. The abort of a transaction's parent will hold the transaction in order to restart it later from the same place or to let the user reinitiate, change or abort it. Similarly, abort of a child transaction will leave to the transaction's parent the options of restarting or aborting that child (depending on the prevailing situation).

b. STs can view the partial results of their parents. Yet when two siblings are executing concurrently, they cannot share the results of each other.

c. Instead of using a 'signal lock' (as suggested by Bertino *et al.* [12]), we propose the use of optional locking (as suggested by Sun [3]) to eliminate the chances of timing covert channel appearance and also to increase transaction performance. Based on this, lower-level transaction(s) can update a data item that is already locked for reading by a higher-level transaction. However, this update will be notified to all the transactions concerned, which will take then appropriate actions to preserve consistency. Instead of leaving this to the users (as proposed by Sun), we propose a mechanism based on *notification* and *action* primitives, namely a 'Raise_Signal' and a 'Get_Signal'. The lower-security level data that is locked for reading by the higher-level transaction is also brought to the local cache and is stored on the local MLSDBS, based on its security classification.

d. Unlike the KT model [8], every mobile transaction in our protocol runs in compensating mode in order to maintain data consistency and guarantee recovery.

4.4.3. Handoff Procedure

a. During handoff, split-transactions operate as in the KT model [8], requiring first some entries in respective transaction-status tables. In our protocol, these tables are maintained in stable storage under the MLSDBS and conform to the security clearance of mobile users. A new site-transaction is created at the destination cell with its own ID, while its various operations are generated by the MH as STs under this site-transaction. An expansion in the cache may be done at this stage if the requested data is not already there. As concerns the previous BS, if the STs of the previous site-transaction continue their execution, the log record is flushed to the stable storage so as to facilitate the execution of the compensating transaction.

b. The TM at the destination BS runs its portion of the split-transaction and makes appropriate entries (like the *Continue Transaction KT* in [8]) into relevant status tables, which (i.e., the entries) hold the state of the mobile transaction. In addition, the TM creates another site-transaction (as before) so as to manage the new STs requested by the MH in the destination cell.

4.4.4. Ending Transaction

Commit

a. Each site-transaction is handled independently and its commit is notified to all the TMs involved in the mobile transaction. These TMs make transaction-ending entries into respective status tables (like *End-*

Transaction ST and *End-Transaction JT* [8]). The STs will commit locally at the MLSDBS holding the cached data (i.e., at the first BS where the KT started from). On successful termination, each ST notifies to its parent(s) and to the site-transaction and, at the end, to the mobile transaction too.

b. Upon commit, each ST makes its results available to its parent and to other sibling transactions (if these are of the same security level in one mobile transaction). The locks held by that ST are retained by its parent.

c. The results become available to the outside world only if the mobile transaction has committed and the database has resynchronized with the central repository.

d. As every transaction is running in compensating mode, all logs are kept on stable storage until the mobile transaction commits (or aborts), in which case these logs are removed.

e. The mobile transaction's status table at the last BS will keep the current state of the transaction. If there is no live site-transaction and the MH sends eventually an end-transaction request, the mobile transaction commits. The end-mobile-transaction entry (like the ETKT in [8]) is written into the respective log and all the records in the BS status tables are freed, so the mobile transaction ends. Following through this, the cached data is updated into the central repository and, upon successful update, this data and all the entries related to the mobile transaction are removed from all the BS's to preserve security.

Abort

a. If a subtransaction aborts, the MH is notified of this effect and can decide to abort the whole (or part of that) transaction, retry some portion of the ST, or re-initiate the whole transaction according to the prevailing situation at that time (contrary to the KT model).

b. If some ST aborts due to a fault, the MH can still carry on with the remaining site-transactions and the mobile transaction. This matter will be reported however to the client who can continue without this ST, or send another ST, or re-initiate automatically the same ST using entries in the log (according to the nested-transaction model).

c. When a ST aborts the mobile transaction itself does not abort (thanks to the employed models of nested and split-transaction). Instead, the client is notified of the ST's abort and can possibly re-initiate that ST by himself or can request its restart (the restart can be done using the log entries of the last operations of the transaction). Alternatively, the client may ask for abort if the site-transaction is not required any more. In that case the site-transaction will not be removed but it will be an empty site-transaction without any ST under it, while the entries in the BS status table and the log will be used for running a compensating transaction (bring thus the database to a consistent state).

d. Upon receiving an abort request from a client, an end-mobile-transaction entry (like the ETKT in [8]) is written onto the log, while all entries in all BS status tables are freed. Then a compensating transaction runs in order to cancel the effects of the failed transaction.

e. A transaction-level or root-level rollback/abort will result in the global transaction's rollback with the help of a compensating transaction, regardless of the results produced in the meantime.

4.4.5. Locking

a. Locking follows the multi-granularity locking protocol in [5], the optional locking scheme in [3] and the nested transaction model in [12], except that multiversion locking (that is supported in [5]) is not supported here so as to ensure 1-copy serializability.

b. The granularity of locking operations is decided by the LM at the central DBMS, so as to facilitate the operations of the mobile transaction which have been requested as part of the first site-transaction. The decision is essentially a compromise between *coarse granularity* (i.e., maximum-size cache that yields high efficiency by keeping local executions at the BS) and *fine granularity* (i.e., minimum-size cache that provides maximum concurrency with other mobile transactions executing at the MHs). Intention locks are assigned implicitly to the parents of explicitly locked items (as well as to the parents above them) and the depth of each intention lock is set by the LM.

c. In addition to multi-granularity levels in locking we also propose the use of optional (tentative) locks (like the ones proposed in [3]), which enable low-level transactions to perform updates even if the targeted data items have been locked for reading by other, higher-level transactions. As we explained in Section 4.3, tentative locks prevent the appearance of covert channels and increase also transaction throughput.

5. SECURITY AGAINST COVERT CHANNELS IN THE PROPOSED PROTOCOL

In this section we discuss how the proposed protocol provides security against covert channels (especially motion covert channels). In fact, our protocol allows for:

a. *No Data Cache in MH*. This feature prevents on one hand the compromising of secure data when a MH is or stolen and, on the other hand, it prevents a malicious high-level subject (or a Trojan Horse) on the mobile device from sending secret pieces of information to low-level subjects (and establish thus covert channels).

b. *No Local Execution at MH.* This feature prevents the establishment of covert channels that would allow the pass of secret information to lower-level subjects, since local execution requires the manipulation of data to be done at the local host. If local execution were allowed at the MH, this could be exploited by a malicious user for establishing covert channels.

c. *Use of MLSDBS at BS.* The data cached at the BS is handled *via* a MLSDBS so as to prevent the sharing of the same data (or resources) by subjects of different levels. As mobile transactions consist of update operations related to only one security classification, the cached information consists of data of one classifica-

tion only (except the data items belonging to lower classifications, which are brought in by the MLSDBS and stored in read-only mode on respective data files, based on their security level).

d. *Use of different Logs and Status Tables based on the Security Clearance of Subjects.* One major cause of the establishment of motion covert channels is the way that the movement behavior of clients is captured at base stations by status tables and logs. As these tables and logs are shared by all transactions that execute on a BS, they offer a good chance to establish covert channels. In our model, the status tables and logs are maintained within the MLSDBS based on their security level. Thereby the establishment of motion covert channels is prevented, since mobile transactions of the same security level utilize one status table and log.

e. *Multiple Granularity Locking.* In this technique, the granule size is never fixed, hindering thus the synchronization between two subjects of different security clearances. Therefore covert channels are hard to be created, because a storage or timing channel requires two subjects of different security clearances to synchronize on a common data item or system resource (e.g., time).

f. *Optional Locking.* The use of optional locks allows a high-level subject to read lower-level data without delaying a lower-level subject from updating that data. This prevents the establishment of timing covert channels (as we explained in Section 4.3).

g. *Use of Data Caches at BS*. This eliminates the chances of coordination between two subjects and prevents thus the establishment of timing or motion covert channels. As we mentioned earlier, one cache belongs to only one mobile transaction or to the mobile transactions of only one security level and also it is resident at the first BS where the mobile transaction was initiated from. This implies that no sharing of data or resources can be done by two subjects of different levels.

Table **1** below summarizes these features along with the kind(s) of covert channel (or other security breach) that each

Table 1. Operational Features of our Protocol and Security Services it Provides

No Data Cache in Mobile Hosts	Prevents timing covert channels	This service is also supported by the protocols described in Ref. [12], [14], [24], [25], [26], [27] and [28].
	Prohibits data compromising in the case of disaster or theft	Not guaranteed by any other protocol in the research literature.
No Local Execution at Mobile Hosts	Prevents motion covert channels	Not supported by any other protocol in the research literature.
Use of MLSDBS at Base Stations	Prevents unauthorized access to classified data	This service is also supported by the protocols described in Ref. [12], [14], [25], [26] and [27].
Use of Different Logs and Status Tables based on the Security Clearance of each Subject	Prevents motion covert channels	Not supported by any other protocol in the research literature.
Multiple Granularity Locking	Prevents timing and storage covert channels	This service is also supported by the protocols described in Ref. [12], [14], [24], [25], [26], [27] and [28].
Optional Locking	Prevents timing covert channels	This service is also supported by the protocols described in Ref. [12], [14], [24], [25], [26], [27] and [28].
	Increases transaction performance	This service is also supported by the protocols described in Ref. [14], [21] and [28].
Use of Data Cache at Base Stations	Prevents timing and motion covert channels	Not supported by any other protocol in the research literature.
	Prevents unauthorized access to classified data	This service is also supported by the protocols described in Ref. [12], [14], [25], [26] and [27].
Support of Nested Transactions	Allows for multiple levels of security in mobile databases	This service is also supported by the protocols described in Ref. [12], [26], [27] and [28].
Support of Compensating Transaction Mode	Guarantees transaction recovery	This service is also supported by the protocols described in Ref. [12] and [27].

feature is meant to eliminate. Based on this information, it compares our protocol against earlier ones in the research literature and shows its superiority with regard to the security it affords in mobile data access.

6. SUMMARY AND FUTURE RESEARCH

In this paper we have examined the possibility of establishing multiple levels of data security over mobile networks, and we have discovered a number of drawbacks in existing protocols for fixed networks as well as in protocols for mobile transactions. Based on this discovery, we proposed a novel transaction model that supports multilevel database security in mobile environments by eliminating the chances of covert channel appearance, especially the appearance of motion covert channels that we have identified as a new kind of security threat in mobile environments. Specifically, our model incorporates nested transaction techniques along with multi-granularity and optional locking protocols to ward off any type of covert channels and make transaction execution faster. In addition, it assumes the existence of distributed data caches at base stations in order to leverage the performance of mobile transactions and prevent the synchronization of malicious processes with different security classifications. On the other hand, local caches have been removed from mobile hosts to ensure the absence of security breaches. Overall, our protocol prevails over similar ones regarding the security it provides in mobile data access (as shown in Table **1** above). In fact, no other protocol allows for multiple levels of security in mobile environments and prevents simultaneously the establishment of *all* kinds of covert channels (i.e., timing, storage and motion).

We intend henceforth to extend our research towards verifying the above protocol with model checking tools. Upon completing the verification process, we intend to perform simulation experiments in order to evaluate the protocol's performance in mobile environments. To this end, we are about to develop prototype components (such as MTM and LM) based on the ideas proposed in this paper. Moreover, we want to explore meticulously the security effects of optional locking, because this technique was initially proposed for collaborative editing rather than for database security. For example, utilizing optional locking to overwrite a transaction's effects may cause losses of multimillion dollars or result in the leakage of sensitive information (related for instance to national security). The aforementioned use of model checking will be extremely useful in that case, as optional locking can cause numerous event interleavings [3] (and generate thus many different states in the operation of a system).

REFERENCES

[1] J. Gray, and A. Reuter. *Transction Processing: Concepts and Techniques*. San Mateo, CA: Morgan Kaufmann, 1993.

[2] G. Samaras, G.K. Kyrou, and P.K. Chrysanthis. "Two-phase commit processing with restructured commit tree," in *Proc. Nat'l. Greek Conf. on Inform.*, pp. 82-99, *LNCS 2563*, 2003.

[3] C. Sun. "Optional and responsive fine-grain locking in internet-based collaborative systems." *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, pp. 994-1008, Sept. 2002.

[4] J.N. Gray, R.A. Lorie, G.R. Putzolu, and I.L. Traiger. "Granularity of locks and degree of con-sistency in a shared database." San Jose, CA: IBM Research Lab, 1975.

[5] C. Papadopoulos. "A multiple granularity locking protocol for CSCW." *Int. J. Cooper. Inf. Syst.*, vol. 11, pp. 21-50, June 2002.

[6] D. Barbará. "Mobile computing and databases - a survey." *IEEE Trans. Knowl. Data Eng.*, vol. 11, pp. 108-117, Jan./Febr. 1999.

[7] T. Imielinski, and B.R. Badrinath. "Mobile wireless computing." *Commun. ACM*, vol. 37, pp. 18-28, Oct. 1994.

[8] M.H. Dunham, A. Helal, and S. Balakrishnan. "A mobile transaction model that captures both the data and movement behaviour." *Mobile Netw. Appl.*, vol. 2, pp. 149-162, June 1997.

[9] S.A. Patricia, C.L. Roncancio, and M. Adiba. "Analyzing mobile transactions support for DBMS," in *Proc. DEXA*, pp. 595-600. *LNCS 2113*, 2001.

[10] E. Pitoura, and B. Bhargava. "Data consistency in intermittently connected distributed systems." *IEEE Trans. Knowl. Data Eng.*, vol. 11, pp. 896-915, Nov. 1999.

[11] G.D. Walborn, and P.K. Chrysanthis. "Supporting semantics-based transaction processing in mobile database applications," in *Proc. IEEE SRDS*, pp. 31-40, 1995.

[12] E. Bertino, B. Catania, and E. Ferrari. "A nested transaction model for multi-level secure database management systems." *ACM Trans. Inform. Syst. Security*, vol. 4, pp. 321-370, Nov. 2001.

[13] E. Bertino, S. Jajodia, L. Mancini, and I. Ray. "Advanced transaction processing in multilevel secure file stores." *IEEE Trans. Knowl. Data Eng.*, vol. 10, pp. 120-135, Jan./Febr. 1998.

[14] B. George, and J.R. Haritsa. "Secure concurrency control in firm real-time database systems." *Distrib. Parallel Dat.*, vol. 8, pp. 41-83, Jan. 2000.

[15] National Computer Security Center. "A guide to understanding covert channel analysis of trusted systems." NCSC-TG-030 (Library no. 5-240-572), 1993.

[16] P. Serrano-Alvarado, C.L. Roncancio, and M. Adiba. "A survey of mobile transactions." *Distrib. Parallel Dat.*, vol. 16, pp. 193-230, Sept. 2004.

[17] Q. Lu, and M. Satyanaraynan. "Improving data consistency in mobile computing using isolation-only transactions," in *Proc. IEEE HotOS-V*, 1995, pp. 124-128.

[18] J.N. Gray, P. Helland, P. O'Neil, and D. Shasha. "The dangers of replication and a solution." *ACM SIGMOD Record*, vol. 25, pp. 173-182, June 1996.

[19] P.K. Chrysanthis. "Transaction processing in a mobile computing environment," in *Proc. IEEE APADS*, pp. 77-82, 1998.

[20] S.K. Madria, and B. Bhargava. "A transaction model for mobile computing," in *Proc. IEEE IDEAS*, pp. 92-102, 1998.

[21] G.D. Walborn, and P.K. Chrysanthis. "Transaction processing in pro-motion," in *Proc. 14th ACM SAC*, pp. 389-398, 1999.

[22] R.A. Dirckze, and L. Gruenwald. "A pre-serialization transaction management technique for mobile multidatabases." *Mobile Netw. Appl.*, vol. 5, pp. 311-321, Dec. 2000.

[23] L. Gruenwald, and S.M. Banik. "A power-aware technique to manage real-time database transactions in mobile Ad-Hoc networks," in *Proc. DEXA*, pp. 570-574. *LNCS 2113*, 2001.

[24] B. Kogan, and S. Jajodia. "Concurrency control in multilevel secure databases based on a replicated architecture," in *Proc. ACM SIGMOD*, pp. 153-162, 1990.

[25] P. Ammann, and S. Jajodia. "An efficient multiversion algorithm for secure servicing of transaction reads," in *Proc. ACM CCS*, pp. 118-125, 1994.

[26] S.H. Son, and R. David. "Design and analysis of a secure two-phase locking protocol," in *Proc. ACM COMPSAC*, pp. 374-379, 1994.

[27] H.-W. Kim, H.-K. Rhee, T.M. Chung, Y.I. Eom, and U.-M. Kim. "A transaction length-sensitive protocol based on altruistic locking for multilevel secure database systems," in *Proc. ICICT*, pp. 107-118, 2005.

[28] H.-W. Kim, D.-S. Park, H.-K. Rhee, and U.-M. Kim. "Advanced transaction scheduling protocol for multilevel secure databases in wireless mobile network environments," in *Proc. IEEE ICATM*, pp. 240-244, 2001.

[29] R.A. Kemmerer. "Shared resource matrix methodology: an approach to identifying storage and timing channels." *ACM Trans. Comput. Syst.*, vol. 1, pp. 256-277, Aug. 1983.

[30] C. Pu, G.E. Kaiser, and N.C. Hutchinson. "Split-transactions for open-ended activities," in *Proc. 14th VLDB*, pp. 26-37, 1998.

Received: July 13, 2007 Revised: September 03, 2007 Accepted: September 03, 2007