Concurrency Control for Multilevel Secure Databases

Navdeep Kaur, Rajwinder Singh, Manoj Misra, and A. K. Sarje
(Corresponding author: Rajwinder Singh)

Department of Electronics and Computer Engineering
Indian Institute of Technology Roorkee, Roorkee-247667, Uttranchal, India
(Email: nrwsingh@yahoo.com)
(Received Feb. 2, 2006; revised Apr. 25. 2006; and accepted May 2, 2006)

Abstract

A multilevel secure (MLS) database is intended to protect classified information from unauthorized users based on the classification of the data and the clearances of the users. The concurrency control requirements for transaction processing in multilevel secure database management systems (MLS/DBMSs) are different from those in conventional transaction processing systems. In MLS/DBMSs, coordination of transactions at different security levels is needed to avoid both covert channels and the starvation of high security level transactions. In this paper we outline the transaction processing requirements in MLS/DBMSs, and survey the mechanisms proposed to address these requirements and propose a new secure multiversion concurrency control protocol. We also investigate the relative performance of existing secure concurrency control protocols while varying workloads.

Keywords: Multilevel security, database system, concurrency control, covert channel

1 Introduction

Database security is concerned with the capability of a database management system to enforce a security policy controlling the disclosure, modification or destruction of data. Security in database systems can be discretionary and mandatory (multilevel) [10]. Discretionary security restricts access of data at the discretion of the owner. Most commercial database management systems use some form of discretionary control, by controlling access privileges and modes of users to data [13]. Discretionary security is not appropriate for certain applications such as military, because it provides a low level of assurance and is subject to Trojan Horse attacks. These applications require mandatory security to restrict access to data items to cleared database users. It is widely used in military applications and provides a high level of assurance. In this paper, we focus on security aspects of database systems that enforce mandatory access control.

Multi-level secure database systems (MLS/DBSs) are shared by concurrent transactions with different clearance levels and manage data objects with different classification levels [35]. Many civilian, defense and commercial applications require MLS/DBSs that support data having different access classes and users with different authorization, or clearances. In such environments, multiple users share the same database, although some of the users can have restricted access to information from the database. Hence, it is necessary to provide database security for these databases.

Most of MLS/DBSs use an access control (mandatory) mechanism based on the Bell-LaPadula model [6]. This model is stated in terms of subjects and objects. An object is a data file, record or a field within a record. A subject is an active process that requests access to objects. Every object is assigned a classification level (i.e. unclassified, confidential, secret, and top secret are usual) based on the security requirement and every subject has a clearance level (i.e. unclassified, confidential, secret, and top secret) based on the degree to which it is trusted by the system. Classification levels and clearance levels are collectively referred to as security levels and are partially ordered.

In the context of MLS/DBs, each transaction as well as each data item is assigned a security level. Transactions are subject to the following restrictions [6]:

Simple Security Property: A subject (transaction) is allowed read access to an object (data item) only if the former's clearance is identical to or higher than the latter's classification.

The *-Property: A subject (transaction) is allowed write access to an object (data item) only if the former's clearance is identical to the latter's classification [21].

The above two restrictions are intended to ensure that there is no flow of information from higher security level objects to lower security level subjects. Database systems that support the Bell-LaPadula properties are called multilevel secure database systems (MLS/DBMSs).

The Bell-LaPadula model prevents direct flow of information from a higher security level to a lower security level, but the conditions stated above are not enough to prevent indirect transfer of information from a higher security level subject to a lower security level subject through covert channels [24]. There are two types of covert channels: storage covert channels and timing covert channels. Storage covert channels disclose information from high security level to low security level subjects by manipulating a physical object, which can or cannot be seen by the low security level subjects. For example, consider a secure operating system, which insists that a file name should be unique. A file name "SDB" exists and is classified "High". The mandatory access control does not allow a "Low" subject to see "SDB" in the directory catalog. If the subject attempts to create a new file named "SDB" the request is denied. Through this denial, the subject learns of the existence of the "High" file "SDB". A cooperating High subject can remove and create "SDB" to signal information to the Low subject. In contrast, timing covert channels signal information by modulating an observable delay. A timing covert channel depends on a resource shared between subjects with different security classification levels e.g. a processor or a disk drive. Let us consider the case of Low and High subjects sharing a single disk drive. By modulating the rate of disk accesses, the High subject can delay the Low subject's computation, and information is transferred to the Low subject. The High user may perform many disk accesses to transmit a 'one' and no disk access to transmit a 'zero'. In this way it can communicate a string of binary digits. The receiving process measures the delay experienced by its disk requests. Important classes of covert channels that are usually associated with concurrency control mechanisms are timing channels. In the context of concurrency control approaches, a covert channel arises when a resource or an object in the database is shared between subjects with different security levels. MLS/DBSs must be designed to avoid such covert channels and security protocols in the systems must guarantee that the low security level transactions are not delayed or aborted by high security level transactions [30].

Concurrency control is used in databases to manage the concurrent execution of operations by different subjects on the same data object such that consistency is maintained. Concurrency control in MLS/DBs, in addition to ensuring correct concurrent execution of transactions must also preserve security. The security community recognizes secrecy, integrity and availability as inherent components of security.

In this paper, we address the requirements of transaction processing in MLS/DBMSs and surveys the mechanisms proposed to address these requirements. This paper is organized as follows: Section 2 addresses the requirements of traditional transaction processing and requirements of secure concurrency protocols. Section 3 outlines the MLS/DBMSs architectures. Section 4 explains how conventional processing techniques can conflict with multilevel security constraints. Section 5 presents the efforts of a number of researchers in designing concurrency control protocols for MLS/DBSs. New secure multiversion concurrency control protocol is presented in Section 6. In Section 7, we evaluate the performance of secure concurrency control protocols through detailed simulation study. Section 8 concludes the paper.

2 Requirements of Transaction Processing

2.1 Traditional Transaction Processing

A database system consists of a collection of interrelated data and a set of programs that access [37] the data. The goal of the database system is to support efficient storage, retrieval and processing of large amount of data. Applications interact with the database system through transactions, which constitute the basic unit of work. Transactions provide the so-called ACID [7] semantics, (i.e. Atomicity, Consistency, Isolation, Durability).A proper concurrency control protocol ensures isolation property of transactions even in the presence of other concurrently running transactions. A proper recovery protocol ensures the atomicity and durability property.

2.2 Requirement of Secure Concurrency Control Protocol

There are three security requirements [20] for multilevel secure concurrency control protocol.

- Integrity requirements: A secure concurrency control protocol must ensure correct execution of transactions (serializability).
- Secrecy requirements: A secure concurrency control protocol must be free of covert channels.
- Availability requirements: A secure concurrency control protocol should not cause starvations (indefinite delays).

3 Multilevel Secure DBMS Architectures

According to "The Woods Hole Report" [31], there are three architectures for MLS/DBSs, which protect classified information from unauthorized users based on the classification of the data and clearance level of users.

In the integrity lock architecture [11], as shown in Figure1, several classification levels of data are integrated in the same database and encryption is used to handle the secrecy of data items. It is basically a front-end and back-end architecture. The front-end is further divided into an untrusted and a trusted front-end. The un-trusted front-end works with query parsing, optimization, and computation. The trusted front-end (TFE) handle tasks such as

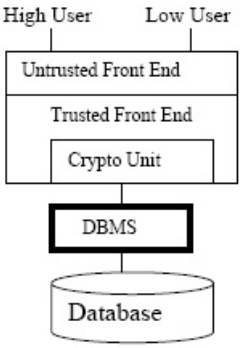


Figure 1: Integrity lock architecture

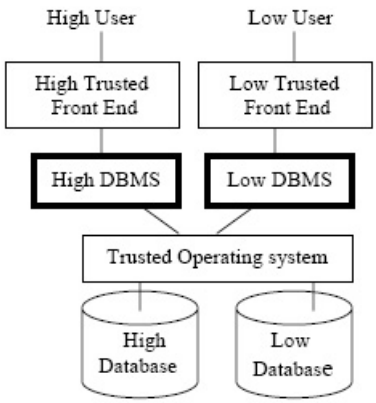


Figure 2: Kernelized architecture

authenticating users, installing integrity locks, and checking integrity locks. The trusted front-end basically encapsulates a filter that encrypts the sensitivity level of the data item to be stored in the database. It then stamps that data item with a checksum or integrity lock. The integrity lock is a function of the data item's sensitivity level, along with the data item itself. The integrity lock is stored, along with the corresponding data item, in the front-end or in the back-end database system. Upon retrieval of a data item, the TFE decrypts the integrity lock of the data item being queried and compares it with the one just computed. It also checks whether the sensitivity level of the data item is less than that of the user, then the data item retrieved is forwarded to the querying user.

The kernalized architecture [25], as shown in Figure 2, relies on decomposing the multilevel database into single level databases, which are stored separately, under the control of security kernel enforcing a mandatory access control policy. The trusted front end ensures that the user's queries are submitted to the DBMS with the same security level as that of the user, while the trusted backend makes sure that a DBMS at a specific level accesses data without violating the mandatory security policy (Bell-LaPadula restrictions). Processing of a user's query accessing data from multiple security levels involves expensive joins that may degrade the performance since different levels of data are stored separately. On the other hand, since this architecture has separate DBMSs for each security level, the scheduler that is responsible for the concurrency control also can be separated for each level.

The third architecture known as the replicated architecture [12], shown in Figure 3, uses a physical distinct backend database managements system for each security level. Each backend database contains information at a given security level and all data from lower security levels. The system security is assured by trusted front end, which permits a user to access only the backend database system, which matches his/her, security level.

If a high transaction wishes to read data from a low level, it will be given the replica of the low level data maintained in the high container. As a result, this architecture is impractical for large number of security levels.

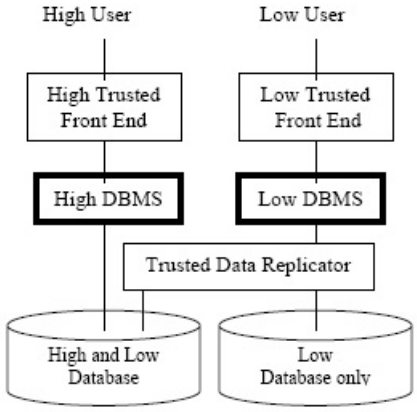


Figure 3: Replicated architecture

Though the query processing is not expensive as in the kernalizied architecture, the critical issue of transaction in this architecture is the propagation of the update of the lower level data to high level DBMSs in a secure and correct manner.

4 Security Threats to Traditional Concurrency Control Protocols

There are number of concurrency control protocols to produce serializable executions of transactions [7], the most commonly used are two phase locking, time stamp ordering, optimistic and multiversion concurrency control protocols.

4.1 Locking Protocol

Two-Phase Locking (2PL) is the most widely used concurrency control protocol in database systems. According to this protocol, a transaction should acquire a read (write) lock before it reads (writes) a data item, and every trans-

action must go through two-phases: a growing phase and a shrinking phase.

A transaction should acquire all the required locks during the growing phase and should release these locks during the shrinking phase. However, once a transaction releases a lock, it cannot acquire any more locks. Unfortunately, this protocol is not suitable for multilevel secure databases since it is prone to covert channels .The following example [5] illustrates how a covert channel can be created by malicious transactions under 2PL.

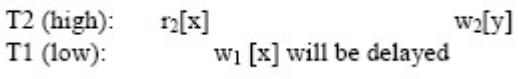


Figure 4: A covert channels with 2PL protocol

Consider a database that stores information of two security levels: high and low. Any low security level information is made accessible to all users of the database by the DBMS; on the other hand, high security level information is available only to a select group of users with special privileges. In accordance with the security policy, a transaction executing on behalf of a user with no special privileges would only be able to access (read and write) low security level data elements, while a high transaction (initiated by a high security level user) would be given full access to the high security level data elements and read-only access to the low security level elements.

Suppose that only two transactions $T1$ and $T2$ are currently active as shown in Figure 4. $T1$ is a transaction initiated by a low security level user and $T2$ is a transaction initiated by a high security level user and, therefore, able to read all data elements. If $T2$ requests to read a low security level data element x, a lock will be placed on x for that purpose. Suppose that next $T1$ wants to write x. Since x has been locked by another transaction, $T1$ will be forced by the scheduler to wait. $T1$ can measure such delays, for example, by going into a busy loop with a counter. Thus, by selectively issuing requests to read low security level data elements, transaction $T2$ could modulate delays experienced by transaction $T1$, effectively sending signals to $T1$. Since $T2$ has full access to high security level data, by transmitting such signals, it could pass on to $T1$ information that the latter is not authorized to see. The information channel thus created is a covert channel and is shown in Figure 4.

4.2 Timestamp Ordering Protocol

Timestamp ordering protocol assigns a unique time stamp to every transaction and maintains two values for each data item: a read timestamp and a write timestamp. When a transaction issues a read (write) operation on a data item, the scheduler allows this operation only if the write (read) timestamp of the data item is not larger than the timestamp of the transaction; otherwise the scheduler rejects the operation. When a transaction reads (write) a data item, the scheduler modifies the read (write) timestamp of the data item with the timestamp of the transaction. For example $T1$ and $T2$ are assigned unique timestamps $ts(T1)$ and $ts(T2)$ respectively. Let $ts(T1) < ts(T2)$. Here, $T2$ modifies the write stamp of x to $ts(T2)$. Since it is grater than $ts(T1)$, $T1$'s write is rejected, thus covert channel is established in time stamping protocol.

4.3 Optimistic Concurrency Control Protocol

In a traditional optimistic algorithm, transactions are allowed to read and update data items without any restriction. All data updates are made permanent during the commit time before which a transaction must pass a validation test where it tests that there is no currently executing conflicting transaction [23]. The validating transaction restarts if the test fails. Optimistic concurrency control (OCC) has the properties of non-blocking and deadlock freedom. These properties make the OCC scheme especially attractive to multilevel-secure transaction processing. Unlike all the pessimistic mechanisms such as two-phase locking (2PL) and timestamp ordering (TO), the MLS/OCC scheme never delays or rejects an operation submitted by a lower-level transaction which is passed by the mandatory access control. Optimistic concurrency control for a secure database can be made to work by ensuring that whenever a conflict is detected between a transaction at a higher security level in its validation phase and a transaction at a lower security level, the transaction at the higher security level is aborted, while the transaction at the lower security level is not affected. A major problem with using optimistic concurrency control is the possible starvation of higher security level transactions. For example, consider a long-running transaction T_h (higher security level) that must read several lower security level data items before the validation stage. In this case, there is a high probability of conflict, and as a result, T_h may have to be rolled back and restarted an indefinite number of times.

5 Concurrency Control Protocols for MLS/DBSs

Concurrency control is important for MLS/DBs because a covert channel can be easily created through collaboration of multilevel secure transactions in most traditional concurrency control protocols. In a MLS/DB, the concurrency control protocol must ensure that there are no covert channels between the transactions at different security levels. Traditional concurrency control protocols such as 2PL and Timestamp Ordering protocols are not

suitable for MLS/DBs, because when those concurrency control mechanisms are applied to multilevel secure transactions, problems such as covert channel, too much delay or repeated aborts of high security level transactions, and retrieval anomaly [14] can occur. Consequently, concurrency control algorithms for MLS/DB must address the problems originated by the security and availability issues of the MLS/DB. Several protocols have been proposed for concurrency control in MLS/DBMS. Due to the influx of these protocols, we have classified the protocols into following five categories:

5.1 Secure Locking Protocol

In the locking-based approaches, in order to prevent timing channels, the executions of transactions at lower security level are never delayed by the actions of a transaction at a higher security level. This can be accomplished by providing a high priority to a low transaction whenever a data conflict occurs between a high transaction and a low transaction.

In [26], Keefe, Tsai and Srivastava examined the security issues and present a formal framework for secure concurrency control in multilevel databases. In this, they have characterized several level of assurance in secure system and show how a scheduler can affect the security in this framework.

A secure locking-based protocol called S2PL was proposed by Jajodia and McCollum [16], which modified the strict two phases locking protocol to covert channel free protocol. In this protocol, a high security level transaction must release its lock on a data item when a low security level transaction requests a write lock on the same data item. When a read lock by a high security level transaction is broken, high security level transaction is to be aborted. Since a low security level transaction is never blocked or restarted by a high security level transaction, this protocol satisfies the secrecy (covert channel free) and integrity requirement, but a malicious low security level transaction may cause a high security level transaction to be aborted repeatedly, resulting in starvation.

McDermott and Jajodia [29] provide a way to reduce the amount of starvation. According to their approach, whenever a high security level transaction prematurely releases its read lock on a low security level data item due to security reasons, it does not abort and roll-back entirely, but holds its write locks on high security level data items, marks the low security level data item in its private workspace as unread and retries reading this data item by entering into a queue. This queue maintains the list of all high security level transactions waiting for retrial to read that particular data item and enables the first transaction in the queue to be serviced first. The modified approach, however, does not always produce serializable schedules [17].

Another secure two-phase locking-based protocol (S2PL), is based on a completely different approach was proposed by Son and David [38]. The basic principle behind this S2PL is to try to simulate the execution of conventional 2PL without blocking the actions of low security level transactions by high security level transactions. This is accomplished by providing a new lock type called virtual lock, which is used by low security level transactions that develop conflicts with high security level transactions. The actions corresponding to setting of virtual locks are implemented on private versions of the data item. When the conflicting high security level transaction commits and releases the data item, the virtual lock of the low security level transaction is upgraded to a real lock and the operation is performed on the original data item. To complete this scheme, an additional lock type called dependent virtual lock is required apart from maintaining, for each executing transaction Ti, lists of the active transactions that precede or follow Ti in the serialization order.

Another solution that has been proposed is to allow users to read and write information at multiple classification levels by decomposing the original transaction into multiple sub-transactions, each of which is assigned a single classification level, and all actions performed by subtransactions obey the Bell-LaPadula properties. However, even in such a scenario, it is impossible to simultaneously guarantee both transaction atomicity and absence of covert channels [9, 28, 39].

Jajodia et al. [17] proposed two secure locking protocol that attempts to detect all cycles in the serialization graph by painting certain transactions and data items accessed by the high security level transactions whose low security level locks are broken and by detecting a cycle at the moment. The first protocol produces pair-wise serializable histories while the second protocol produces serializable histories if the security levels form a total order.

E. Bertino et al. [8] presented an approach to secure concurrency control for transactions in a multilevel secure environment. This approach, which uses single- version data items, is based on the use of nested transactions, application-level recovery, and notification-based locking protocols. The notification protocol is based on the use of signal locks. A signal lock is acquired by a transaction whenever it needs to read lower security level data; such a lock does not delay a write lock request by a low security level transaction on the same data item. Hence, timing covert channels arising from synchronization are eliminated. When a data item on which a write lock is acquired by a transaction is modified, all high security level transactions holding signal locks on that data are notified by the trusted lock manager, and thus may perform recovery actions. To better support recovery activity, transactions are organized according to the nested transaction model extended with specific primitives for supporting the notification protocol. The proposed approach satisfies most of the properties pointed out in Atluri et al. [5], as basic requirements for a secure concurrency control mechanism in a multilevel environment: it avoids starvation and timing channels, and guarantees serializability.

5.2 Secure Timestamp Ordering Protocol

Ammann and Jajodia [1] have proposed two single-version timestamp ordering (TO) protocols. In their first protocol, a high security level transaction trying to read a low security level object is delayed until all low security level transactions before it in the timestamp order have completed. A modification of this protocol allows the high security level transaction to read the low security level object when it needs to but delays its commit until all the low security level transactions with earlier timestamps have completed. In both algorithms, meet all secrecy and integrity requirements but they are prone to starvation.

5.3 Secure Optimistic Protocol

Kang and Moon [18] presented two different approaches in the area of optimistic concurrency control for MLS, single version database. Firstly, read-down conflict preserving serializability captured MLS database consistency requirements and secures transaction correctness properties via a single notion. Secondly, it presented a MLS optimistic concurrency control scheme that has two properties: If lower security level transactions were somehow allowed to continue with its execution in spite of the conflict of high security level transactions, timing covert channel freeness would be satisfied. This sort of optimistic approach for conflict insensitiveness and the properties of non-blocking and deadlock freedom make the optimistic concurrency control scheme especially attractive to MLS transaction processing.

5.4 Secure Multiversion Protocol

To eliminate starvation, instead of single version, multiple versions of data is to maintain. The contention to the same data items by high and low security level transactions can be resolved by maintaing multiple versions of data where high security level transactions are given older version of data. The significant efforts based on multiversion time stamp ordering techniques are made by Keefe and Tsai [19], Jajodia and Atluri [14] and Maimone and Greenberg [26].

Maimone and Greenberg [26] presented two algorithms that are based upon multiversion timestamp ordering technique implemented with single level subjects. These algorithms avoids covert channel but does not guarantee one-copy serializability.

It has been argued that the serializability requirement is overly restrictive for secure databases [26]. Therefore, alternative and weaker notions of database correctness for MLS/DBs, such as level-wise serializability, one-item read serializability, and pair-wise serializability; have been proposed in [14]. These weaker notions of correctness can be used as alternatives for one-copy serializability. They exploit the nature of integrity constraints in MLS/DBs to improve the amount of concurrency.

A multi-version timestamp ordering (MVTO) protocol for kernelized models based on multilevel secure scheduler was suggested by Keefe and Tsai [19]. The difference between basic MVTO and secure MVTO is that secure MVTO will sometimes assign a new transaction a timestamps that is earlier than the current timestamp. This effectively moves the transaction into the past respect to active transactions. To be more precise, when a transaction begins, it is assigned a timestamp that precedes the timestamps of all transactions active at strictly dominated security levels classes and follows the timestamps of all transactions at its own security level. This approach to timestamp assignment is what makes it impossible for a transaction to invalidate a read from a higher security level. This method has the drawback that transactions at a higher security level are forced to read arbitrarily old values from the database due to timestamp assignment. This problem can be especially serious if most of the lower security level transactions are long running transactions. Also, the number of versions required by the protocol does not have an upper bound, and the version pool being physically separate from the primary version pool can reduce the overall physical clustering and therefore affect the system performance.

Kogan and Jajodia [15] discussed a concurrency control mechanism in secure databases using a replicated architecture, which guarantees serializable execution of concurrent transactions. This protocol, however, has the same problem as that associated with the Secure MVTO algorithm, where transactions at a higher security level might be forced to read arbitrarily old values.

Amman et al.[3] have proposed a timestamp-oriented concurrency control protocol that ensures one-copy serializability [7]. This protocol represents a significant improvement over [19], as it requires a fixed number of versions. Their protocol uses two snapshots of the database, in addition to the most recently committed version on which same security level reads (i.e., read operations in which the issuing transactions are at the same security level as the accessed objects) and committed updates execute. Amman and Jajodia [2] have proposed an extension of their two-snapshot algorithm to support the execution of long read-only transactions. A long transaction T reads old snapshot even at its own security level; this sets a deadline by which T must complete, failing which T aborted. In both these papers, the authors assume a kernelized architecture for data access.

Jajodia and Atluri's single level scheduler [14] is based on multiversion timestamp ordering protocol, but differs from Keefe Tsai [19] scheduler in many respects. Here, when low security level transaction tries to write a data item x while high security level transaction already acquired a read lock on x, low security level transaction creates a new version of x. Consequently, there will be no chance to open a covert channel, since conflicts are resolved by giving different versions to each multilevel transaction. However, there will be an additional problem of inconsistent versions given to the read operations of a high transaction, termed a retrieval anomaly [14]. For example, when high security level transaction that has been

blocked waiting for low security level transaction to be finished resumes, it may see new versions created by low security level transaction while high security level transaction is blocked. In order to solve the retrieval anomaly for the multilevel transactions, the one-copy serializability algorithm is proposed [14]. In the proposed algorithm, if a data item read by a high security level transaction is updated and invalidated by a low level transaction, then after low security level transaction commits, high security level transaction is re-executed starting from reading that invalidated data item in order to produce one-copy serializable schedule.

The existing multilevel secure concurrency control algorithms [14, 19, 26] use Timestamp ordering protocol [7] to establish serialization of transactions among various security levels. However, as the inherent problem of the timestamp ordering protocol, the scheduler rejects operations that arrive too late and then aborts the corresponding transaction. Due to security reasons, in a MLS/DBS, all high security level transactions trying to read low security level data are bound to wait and thus are aborted later. Considering the fact that the end result of high security level transactions leads to real-time decision-making processes, abortion of high security level transactions is never an acceptable solution. Moreover, the system must maintain a clock in these protocols, which should be accessed by all transactions at all security levels in order to obtain the timestamp. This method not only creates a bottleneck in the system, but also possesses the threat of a covert channel.

S. Pal. [32] has proposed a locking protocol using two committed versions of data, which produced one-copy serializable and strict schedules. In this protocol, higher security level transactions read down on an earlier committed version of the data while transactions accessing data at their own security level execute on the later committed version. The protocol is free of starvation of higher security level transactions because a higher security level transaction is never aborted due to a lower security level transaction. However, the protocol is too conservative. It imposes a deadline within which transaction must complete if they read data at dominated levels. If they cannot, then such transactions have to be aborted, possible resulting in indefinite delay of such transactions. Moreover, higher security level transactions are always given older versions of the data to read and this may not be acceptable always.

Mancini and Ray [27] proposed a secure concurrency control algorithms that is based on locking strategy and that requires only two versions - one committed and one non-committed version of data. All read operations proceed on the committed version while the write operations proceed on the uncommitted version. All transaction processing mechanism maintain before image information (i.e. last committed version) of at least those data items which have been updates by active transactions (i.e. the current non-committed version) for recovery purposes in case these updating transactions abort. Thus using these two versions of data for concurrency control does not entail any additional cost to data management and hence seems to be useful. Version management is thus cheaper than that in [19]. Moreover unlike [32], this algorithm does not impose any deadlines within which transactions must complete if they read data at dominated levels.

Kim and Kim [22] proposed a multiversion secure concurrency control algorithm that satisfies both requirements of security and integrity and security is viewed as correctness criteria stated in [41]. This algorithm is free of starvation of high transactions by introducing the concepts of conflict transaction set, invisible area and t-lock while it generates no covert channel and produces serializable schedule using multiversion data regardless of the number of security levels considered in the concurrency control. Additionally, it can enhance the availability of the MLS/DB by the simple strategy of version management that is based on the t-locks and only write timestamps.

Y. Sohn and S. Moon [40] proposed multiverison database based secure concurrency controller, named verified ordering-based secure concurrency controller (VO), which maintains the information about ordering relationships among transactions. Whenever VO receives an operation from a transaction, it verifies an ordering relationship between the transactions. By referencing the information, VO is capable for guarding unnecessarily aborted transactions and reading excessively outdated data versions. They also compare the performance of VO with orange-locking secure concurrency control [29] and orderstamp secure concurrency control [19] through simulation.

5.5 Other Secure Protocol

Another MLS concurrency control algorithm was published in [34] that is based on the ROLL [36]. Although this method neither aborts high security level transactions nor has the potential of any covert channels, it does require that the transactions must predeclare their read and write sets. This is not possible in many applications, in particular, where the data to be accessed by an operation is determined by the result of another operation of the same transaction.

B. Panda [33] presented a concurrency control algorithms based on the Serialization Graph Testing (SGT) technique to eliminate the problem with all secure concurrency control algorithms [14, 19, 26]. This approach uses stored serialization graphs (SSGs) to determine the appropriate execution order of various MLS transactions to ensure correctness of transaction execution. The algorithm provides global serializability, guarantees multilevel security, substantially reduces abortion of high security level transactions, allows multiple concurrent transactions at any security level, eliminates system-wide timestamp assignment bottlenecks, and allows single-level concurrency control activities to execute in parallel.

6 Proposed Secure Multiversion Concurrency Control Protocol

Recently Kim et al. [22] and Sohn et al. [40] proposed multiverison database based secure concurrency protocols. [40] has to maintain ordering relationships among transactions whenever it receives operations, resulting need more storage space and more access overheads over [22]. With such advantages in mind, we propose a new secure multiversion concurrency control protocol based on [22]. Kim et al proposed protocol avoids starvation of high level transactions and schedules transactions based on their priorities.

Let $L(T)$ be the security level of transaction T. Transaction T_i has higher priority over transaction T_j if $L(T_i) < L(T_j)$. Kim et. al. defined invisible area to a high transaction T_j as an interval from the time when T_j is blocked by the execution of another transaction T_i to the time when T_j resumes its execution. A ***conflict transaction set*** of a transaction T_j denoted as **C-set**T_j, consists of the transactions that enter an invisible area of T_j due to conflicts with T_i. Suppose T_i is submitted to the scheduler with its read and write sets. If R-set$T_j \bigcap$ W-setT_i is not empty, then $T_i \in$ C-setT_j. To avoid retrieval anomaly the value updated by transactions in C-setT_j are visible to T_j.

We show the improvement of [22] by modifying the condition that is used to add a transaction into **C-set**T_j. Our proposed protocol provides higher degree of concurrency, reads recent version by high transaction and fair execution of all transactions, regardless of their security levels than existing protocols.

We show that the size of **C-set**T_j can be considerably reduced by relaxing the condition used in [22] without compromising on consistency or security of the database. The examples given below explain this.

Let T_j and T_i be high and low transactions respectively. x and y are low data objects and access by both T_j and T_i whereas z is high data object and only access by T_j.r, w and c stands for read, write and commit operation respectively.

Example 1. *Suppose that T_j is currently in execution when T_i is submitted and $L(T_i) < L(T_j)$. T_i blocks the execution of T_j and writes y_1. T_j may be allowed to read y_1 without creating retrieval anomaly.*

$$\begin{aligned} T_j: &\quad r_j[x0] \qquad\qquad r_j[y_1]w_j[z_0]c_j \\ T_i: &\qquad\quad r_i[x0]r_i[y_0]w_i[y_1]c_i. \end{aligned}$$

Example 2. *Suppose that T_j is currently in execution when T_i and T_k are submitted and $L(T_i) < L(T_j)$.T_j may be allowed to read z_k but not y_1 as T_i writes x.*

$$\begin{aligned} T_j: &\; r_j[x0] \qquad\qquad r_j[y_0]w_j[z_k]c_j \\ T_i: &\qquad\quad r_i[x_0]w_i[x_1]r_i[y_0]w_i[y_1]c_i \\ T_k: &\qquad\quad\; r_k[z_0]w_k[z_k]c_k. \end{aligned}$$

Based on these observations we modify the membership of conflict transaction set *by modifying the condition that is used to add a transaction into* ***C-set***T_j.

Modified Condition:
We divide R-setT_j into two parts R-set$_{done}T_j$ and R-set$_{remaining}T_j$. Both R-set$_{done}T_j$ and R-set$_{remaining}T_j$ are dynamically updated when T_j read a data object and R-set$_{done}T_j \cup$ R-set$_{remaining}T_j$ = R-setT_j.

R-set$_{done}T_j$ *contains data objects read by T_j.*

R-set$_{remaining}T_j$ *contains data objects to be read by T_j.*

Let T_i be the transaction submitted to the scheduler with its read and write sets and $L(T_i) < L(T_j)$. It is added to C-setT_j iff

(R-set$_{done}T_j \cap$ W-set$T_i \neq 0$) or ((R-set$_{remaining}T_j \cap$ W-set$T_i \neq 0$) & (R-set$T_i \cap$ W-set$T_k \neq 0$))—(i) where $T_k \in$ C-setT_j.

This modification increases concurrency and reduces the blocking time of high transactions resulting into improvement of their response time. The example explains this.

Example 3. *Suppose that T_j is executing when T_i is submitted and $L(T_i) < L(T_j)$. As at the time of arrival of T_i condition (i) is not satisfied, T_i is added into C-setT_j. Both T_j and T_i execute concurrently and T_j reads y_1 written by T_i.*

$$\begin{aligned} Time: &\; 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \\ T_j: &\; r_j[x_0] \qquad\qquad\qquad r_j[y_1]w_j[z_1]\; c_j \\ T_i: &\qquad\; r_i[x_0]r_i[y_0]w_i[y_1]c_i. \end{aligned}$$

This can also be verified that modified condition produces one-copy serial schedule $T_i\, T_j$. The modified version of the algorithm given in [22] is as follows:

Algorithm 1.

Step 1. *The scheduler receives R-setT_i and W-setT_i when T_i is submitted.*

Step 2. *When there is no transaction in execution, the scheduler executes T_i. When T_i commits, the scheduler performs Step 6.*

Step 3. *When there is a transaction T_j that is currently in execution, the scheduler has three cases according to the security levels of engaging transactions:*

- *Case 1: $L(T_i) > L(T_j)$. Step 4 is performed.*
- *Case 2: $L(T_i) = L(T_j)$. Step 5 is performed.*
- *Case 3: $L(T_i) < L(T_j)$. Scheduler blocks T_j and let T_i enter an invisible area of T_j for the whole duration of T_i's execution. The scheduler executes the operations of T_i immediately such that a covert channel should not be created. T_i is added into C-setT_j if R-set$_{done}T_j \cap$ W-set$T_i \neq 0$. When T_i commits, Step 6 is performed.*

Step 4. *Since a low transaction T_j has been running, the scheduler makes the high transaction T_i wait until T_j terminates. When T_j commits, the scheduler performs Step 6.*

Step 5. *Because both transactions are in the same security level, the scheduler executes them concurrently if ($(W\text{-}setT_j \cap R\text{-}setT_i = 0)$ & $(W\text{-}setT_j \cap W\text{-}setT_i = 0)$) However If (i) is satisfied the new version created by T_i are t-locked and not visible to T_j. When each of T_i and T_j commits, Step 6 is performed.*

Step 6. *When T was the only transaction executed at Step 2, the scheduler waits for other transactions to be submitted. Otherwise, the scheduler selects and executes those transactions, say T_k, whose security levels are the lowest among the blocked transactions, along with new transactions in the same security level as T_k submitted while T_k has been blocked. Those t-locks set on the new versions created in the invisible areas of T_k must not be released until T_k commits. When T_k terminates, the scheduler releases all the t-locks by obtaining information from $C\text{-}setT_k$. And the old versions are discarded at this point, if they are not accessed by any other transaction.*

7 Performance Evaluation

We evaluate the performance of proposed secure concurrency control protocol and SMVCC [22] via simulation at two security levels (high and low). In this section, we first describe the workload and system model used in our simulations and metrics used to access the performance of secure concurrency control protocols. We then describe our results.

In our model, the system consists of a shared-memory multiprocessor DBMS operating on disk-resident data. The database itself is modeled as a collection of pages that are uniformly distributed across all of the disks. Transactions are generated in a Poisson stream and each transaction has an associated security clearance level. A transaction consists of a sequence of page read and page writes accesses. A read access involves a concurrency control request to get access permission, followed by a disk I/O to read the page, followed by a period of CPU usage for processing the page. Write requests are handled similarly except for their disk I/O, their disk activity is deferred until the transaction has committed. Here we assume that the DBS has sufficient buffer space to allow the retention of updates until commit time. A transaction that is restarted due to a data conflict follows the same data access pattern as its original incarnation. The following two subsections describe the workload generation process and the hardware resource configuration.

7.1 Workload Model

The workload model characterizes transactions in terms of their security clearance levels and their data accesses. Table 1 summarizes the key parameters of the workload model. Transactions arrive in Poisson stream, i.e., their inter arrival rates are exponentially distributed. The *ArriRate* parameter specifies the mean rate of transaction arrivals. A transaction is equally likely to belong to any of the *ClearLevels* security clearance levels.

The number of pages accessed by a transaction is determined by normal distribution with mean *TransSize*, and the actual pages to be accessed are determined uniformly from the database. Due to security reasons, each transaction can only access data from a specific part of the database. The database is equally partitioned into *ClassLevels* security classification levels. Transaction access to data pages is given strictly based on Bell-LaPadula specifications. That is, a transaction cannot read pages classified above its clearance level. Also, a transaction is not allowed to write to pages that have a security classification below and above its own clearance level. The *WriteProb* parameter determines the probability that a transaction operation is a write.

7.2 System Model

The physical resources of the database system consist of multiple CPUs and multiple disks. There is a single common queue for the CPUs whereas each of the disks has its own queue.

Table 2 shows the resource parameters of the system. The *DatabaseSize* parameter gives the number of pages in the database. The *NumCPUs* and *NumDisks* parameters specify the hardware resource composition, while the *PageCPU* and *PageDisk* parameters capture CPU and disk processing times per data page. The *ClassLevels* parameter specifies the number of classification levels in the system.

7.3 Performance Metrics

In our simulation experiments, the primary performance metric used is average response times of transactions at each security level for varying arrival rates. The response time of a transaction is the time from the start of the transaction to its final commitment, regardless of the number of times it has been aborted and restarted. This is not only a measure of comparison between the different secure concurrency control protocols, but is also one estimate of fairness since a fair concurrency control protocol must ensure near identical response times across different security levels. Throughput can be inferred from response time. We therefore have omitted the presentation of results of throughput.

Table 1: Workload parameters

Parameter	Meaning	Base Value
ArriRate	Mean transaction arrival rate	Varies (0- 100)
ClearLevels	Number of clearance Levels	2
TransSize	Average transaction size	10
WriteProb	Write probability	30

Table 2: System parameters

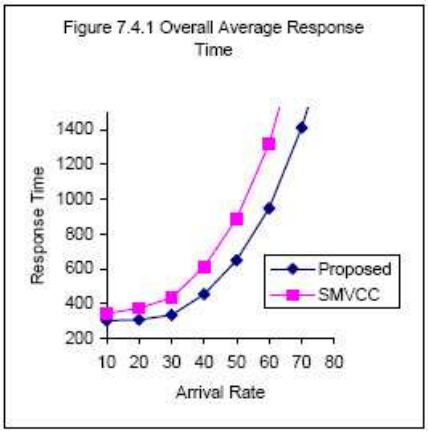
Parameter	Meaning	Base Value
DatabaseSize	Number of pages in the database	500
NumCPUs	Number of processors	2
NumDisks	Number of disks	4
PageCPU	CPU time for processing a data page	12ms
PageDisk	Disk service time for a page	35ms
ClassLevels	Number of classification levels in the system	2

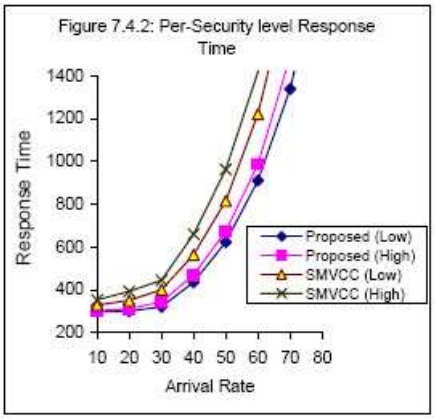
7.4 Experiments and Results

The simulation program is written in C language. For each experiment, we ran the simulation with same parameters for six different random number seeds. Each simulation ran terminated where at least 2,000 transactions of each security level had committed. The results depicted are the average over four runs. All results reported in this paper have 95% confidence intervals. In our simulation experiments, the workload and system parameter settings are set such that there is significant data and resource contention in the system, thus helping to bring out the differences. The parameter settings used for experiments are shown in Tables 1 and 2.

In Figure 7.4.1, the response times of proposed and SMVCC [22] are measured for varying arrival rates. In this figure, we first see that the response time of both concurrency control protocols is same at low arrival rates. This is because the contention levels are low, and the majority of time is spent in disk access and CPU access rather than in resource queues, lock queues, or transaction aborts. As the arrival rate increases, the impact of these factors increases, and depending on how much they increase in each concurrency control approach, the performance varies. As the arrival rate increases, the contention level for data items increases. As a result, in SMVCC more high security level transactions are blocked to give the higher priority to low security level transactions than that of proposed protocol.

The fairness of both protocols can be measured by measuring the response times at each security level for varying arrival rates. The resulting graph is shown in Figure 7.4.2. The performance margin of proposed protocol at both security levels (high or low) is small, indicating a better measure of fairness than in SMVCC.





8 Conclusions

This paper is an attempt to summarize available literature pertaining to work in the direction of developing concurrency control protocols for multilevel secure database systems, since the traditional concurrency control protocols

such as two phase locking, timestamp ordering and optimistic cannot satisfy the multilevel secure requirements. Although researchers have made efforts to modify the traditional concurrency control protocol to make them secure but these protocols suffer from starvation of high security level transactions. In this paper, we proposed and evaluated a new secure multiversion concurrency control protocol. It was observed that our protocol has better response than SMVCC. In addition to this, results show that our protocol achieve fair performance than SMVCC across different security levels. In this paper, we have restricted ourselves to only two security levels. As a part of future work, we would like to relax this assumption and consider multiple security levels.

References

[1] P. Ammann and S. Jajodia "A timestamp ordering algorithm for secure, single version, multilevel database," *Database Security, V: Status and Prospectus*, pp. 23-25, North Holland, 1992.

[2] P. Ammann, and S. Jajodia, "An efficient multiversion algorithm for secure servicing of transaction reads," *Proceedings of Second ACM Conference on Computer and Communications Security*, pp. 118-125, Nov. 1994.

[3] P. Amman, F. Jaeckle, and S. Jajodia, "A two snapshot algorithm for concurrency control in secure multilevel databases," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 1992.

[4] V. Atluri, S. Jajodia, and E. Bertino, "Transaction processing in multilevel secure databases using kernelized architecture: challenges and solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 5, pp. 697-708, 1997.

[5] V. Atluri, S. Jajodia, T. F. keefe, C. McCollum, and R. Mukkamala, "Multilevel secure transactions processing: Status and prospectus," *Database Security X: Status and Prospects*, pp. 79-98, Chapman & Hall, London, 1997.

[6] D. E. Bell and L. J. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, The MITRE Corporation, 1976.

[7] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Company, 1987.

[8] E. Bertino, B. Catania, And E. Ferrari,"A nested transaction model for multilevel secure database management systems," *ACM Transactions on Information and System Security*, vol. 4, no. 4, pp. 321-370, Nov. 2001.

[9] B. T. Blaustein, S. Jajodia, C. D. McCollum, and L. Notargiacomo, "A model of atomicity for multilevel transactions," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 120-134, Oakland, California, 1993.

[10] S. Castano, M. G. Fugini, G. Martella, and P. Samarati, *Database Security: Addison-Wesley*, Reading, MA, 1994.

[11] D. Denning, "Commutative filters for reducing inference threats in multilevel database systems," *Proceedings Of the IEEE Symposium on Security and Privacy*, pp. 134-146, 1985.

[12] J. N. Froscher and C. Meadows, "Achieving a trusted database management system using parallelism," *Database Security II: Status and Prospectus*, pp. 151-160, 1989.

[13] P. P. Griffiths and B. W. Wade, "An authorization mechanism for a relational database system," *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 242-255, 1976.

[14] S. Jajodia and V. Atluri, "Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 839-854, Oakland, California, 1992.

[15] S. Jajodia and B. Kogan, "Concurrency control in multilevel secure databases based on a replicated architecture," *Proceedings of the llth IEEE Symposium on Security and Privacy*, pp. 360-368, Oakland, CA, Apr. 1990.

[16] S. Jajodia and C. McCollum, "Using two-phase commit for crash recovery for federated multilevel secure database management systems," *Dependable Computing and Fault Tolerant Systems*, vol. 8, pp. 365-381, New York, Springer-Verlag, 1993.

[17] S. Jajodia, L. V. Mancini, and I. Ray, "Secure locking protocol for multilevel database management systems," *Proceedings of the Annual IFIP WG 11.3 Conference of Database Security*, pp. 177-194, 1995.

[18] S. Kang and S. Moon, *Read-down Conflict-preserving erializability as a correctness criterion for multilevel-secure optimistic concurrency control: CSR/RD*, *Journal of Systems Architecture*, pp. 889-902, 2000.

[19] T. F. Keefe and W. T. Tsai, "Multiversion concurrency control for multilevel secure database systems," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 369-383, Oakland, California, 1990.

[20] T. Keefe, W. Tsai, and J. Srivastava, "Multilevel secure database concurrency control," *Proceedings of IEEE International Conference on Data Engineering*, pp. 337-344, Feb. 1990.

[21] T. F. Keefe, W. T. Tsai, and J. Srivastava, "Database concurrency control in multilevel secure database management systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 1039-1055, 1993.

[22] H. T. Kim and M. H. Kim, "Starvation-free secure multiverison concurrency control," *Information Processing Letters*, vol. 65, pp. 247-253 pp. 247-253, 998.

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

[24] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613-615, 1973.

[25] T. Lunt, et el., "The sea view security model," *IEEE Transactions on Software Engineering*, vol. 16, pp. 247-253, 1990.

[26] W. T. Maimone and I. B. Greeberg, "Single level multiversion schedulers for multilevel secure database systems," *Proceedings of the Sixth Annual Computer security Application Conference*, Tucson, Arizona, pp. 137-174, Dec. 1990.

[27] L. Mancini and I. Ray, "Secure concurrency control in MLS databases with two versions of data," *Proceedings of the Esorics'96*, LNCS 1146, pp. 304-323, Springer Verlag, 1996.

[28] A. G. Mathur and T. F. Keefe, "The concurrency control and recovery problem for multilevel update transactions in MLS systems," *Proceedings of the IEEE Computer Security Foundations Workshop*, pp. 10-23, Franconia, NH, 1993.

[29] J. McDermott and S. Jajodia, "Orange locking: Channel free database concurrency control via locking," *Database Security, VI: Status and Prospects, Database Security*, pp. 267-284, 1995.

[30] I. S. Moskowitz and M. H. Kang, "Covert channels - Here to stay," *Proceedings of the COMPASS'94*, pp. 235-243, Gaithersburg, MD, IEEE press, 1994.

[31] *Multilevel Data Management Security: Committee on Multilevel Data Management Security, Air Force Studies Board*, National Research Council, Washington, 1983.

[32] S. Pal, "A locking protocol for multilevel secure databases using two committed versions," *Proceedings of the 10th Annual Conference on Computer Assurance, COMPASS 95*, pp. 197-210, June 1995.

[33] B. Panda, "An alternative approach to serialization of multilevel secure transactions," *Proceedings of the 1997 ACM Symposium on Applied Computing*, pp. 134-135, 1997.

[34] B. Panda, W. Perrizo, and R. Haraty, "Secure transaction management and query processing in multilevel secure database systems," *Proceedings of the 1994 ACM Symposium on Applied Computing*, pp. 363-368, Phoenix, AZ, 1994.

[35] C. Park and S. Park, *Multiversion concurrency control protocol with freezing method in multilevel secure database systems*, *Journal of KISS (B): Software and Applications*, vol. 25, no. 8, pp. 1159-1169, 1998.

[36] W. Perrizo, "Request order linked list (ROLL): A concurrency control object for centralized and distributed database systems," *Proceedings of the 7th International Conference on Data Engineering*, pp. 278-285, Kobe, Japan, 1991.

[37] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts: 3rd ed. McGraw-Hill*, 1997.

[38] S. H. Son and R. David, "Design and analysis of a secure two-phase locking protocol," *18th International Computer Software and Applications Conference (COMPSAC'94)*, pp. 374-379, IEEE Computer Society Press, 1994.

[39] K. P. Smith, B. T. Blaustein, S. Jajodia, and L. Notargiacomo, "Correctness criteria for multilevel transactions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 32-35, 1996.

[40] Y. Sohn and S. Moon, "Verified order-based secure concurrency controller in multilevel secure database management system," *IEICE Transactions of Information & Systems*, vol. E83-D, no. 5, pp. 1128-1141, May 2000.

[41] R. Thomas and S. Sandhu, "Towards a unified framework and theory for reasoning about security and correctness of multilevel transactions," *Proceedings of the IFIP WGl 1.3 Workshop on Database Security*, Lake Guntersville, AL, 1993, in *Database Security VII: Status and Prospects*, pp. 309-328, North-Holland, Amsterdam, 1994.

Navdeep Kaur received her M.Tech degree in Computer Science and Engineering from Kurukshetra University, Kurukshetra, India . She is currently doing research on Distributed Database Security for her Ph.D degree from I.I.T Roorkee, Roorkee, India. Besides this, her research interests are in computer and network security.

Rajwinder Singh received the M.Tech (Computer Science and Engg) degree from Kurukshetra University, India in 1998. He is currently pursuing the Ph. D. degree in the Dept. of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India. His current research interests include Security, Mobile Computing Distributed Database.

Manoj Misra did Phd from University of Newcastle Upon Tyne, UK in 1997 and currently working as an Associate Professor in Electronics & Computer Engineering Department, IIT Roorkee. He is also Co-coordinator of Information Super Highway Center, IIT Roorkee that provides state of the art infrastructure for voice, video, and data communication within and outside the IIT for education, research and training. Before joining IIT Roorkee, he worked at Hindustan Aeronatics Ltd, Bharat Heavy Electricals Limited and Computer Maintenance Co. Ltd. He has published more than 50 papers in International Journals and Conferences and visited countries like UK, USA, France and China.

A. K. Sarje received the B.E., M.E. and PhD degrees from Indian Institute of Science, Bangalore , India in 1970, 1972 and 1976 respectively. He is currently Professor of Computer Science and Engineering at Indian Institute of Technology Roorkee, Roorkee , India. Dr Sarje's research interests include Distributed Systems, Computer Networks, Real Time Systems, Network security. He has published more than 100 technical papers in referred journals and conference proceeding.