Secure Transaction Processing in Firm Real-Time Database Systems

Binto George Jayant Haritsa

Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore 560012, India
{binto, haritsa}@serc.iisc.ernet.in

Abstract

Many real-time database applications arise in safety-critical installations and military systems where enforcing **security** is crucial to the success of the enterprise. A secure real-time database system has to simultaneously satisfy two requirements – guarantee data security and minimize the number of missed transaction deadlines. We investigate here the performance implications, in terms of missed deadlines, of guaranteeing security in a real-time database system. In particular, we focus on the *concurrency control* aspects of this issue.

Our main contributions are the following: First, we identify which among the previously proposed real-time concurrency control protocols are capable of providing protection against both direct and indirect (covert channels) means of unauthorized access to data. Second, using a detailed simulation model of a firm-deadline real-time database system, we profile the real-time performance of a representative set of these secure concurrency control protocols. Our experiments show that a prioritized optimistic concurrency control protocol, OPT-WAIT, provides the best overall performance. Third, we propose and evaluate a novel *dual* approach to secure transaction concurrency control that allows the real-time database system to *simultaneously* use different concurrency control mechanisms for guaranteeing security and for improving real-time performance. By appropriately choosing these different mechanisms, we have been able to design *hybrid* concurrency control algorithms that provide even better performance than OPT-WAIT.

1 Introduction

Many real-time database applications arise in safety-critical installations and military systems where enforcing **security** is crucial to the success of the enterprise. Surprisingly, however, the issue of providing security in real-time database systems (RTDBS) has received comparatively little attention although real-time database research has been underway for close to a decade now. In this paper, we partially address this lacuna by making a detailed investigation of the performance implications of providing security in the context of real-time applications with "firm-deadlines" [9] – for such applications, completing a transaction after its deadline has expired is of no utility and may even be harmful.

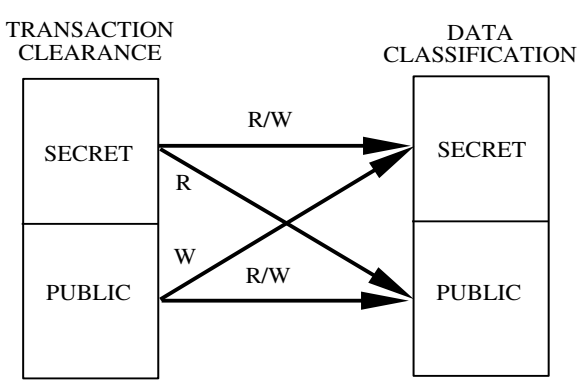
Database Security

Most secure database systems have access control mechanisms based on the Bell–LaPadula model [12]. This model is specified in terms of *subjects* and *objects*. An object is a data item, whereas a subject is a process that requests access to an object. For example, when a process accesses a data file for input/output operations, the process is the subject and the data file is the object. Each object in the system has a *classification* level (e.g., Secret, Classified, Public, etc.) based on the security requirement. Similarly, each subject has a corresponding *clearance* level based on the degree to which it is trusted by the system.

The Bell–LaPadula model imposes two restrictions on all data accesses:

- A subject is allowed *read* access to an object only if the former's clearance is higher than or identical to the latter's classification.
- A subject is allowed *write* access to an object only if the former's clearance is identical to or lower than the latter's classification.

Figure 1: Bell-LaPadula access restrictions



The Bell–LaPadula conditions, by enforcing a "read below, write above" constraint on transaction data accesses (an example is shown in Figure 1), prevent *direct* unauthorized

access to secure data. They are not sufficient, however, to protect from "covert channels". A covert channel is an *indirect* means by which a high security clearance process can transfer information to a low security clearance process [11]. For example, if a low security process requests access to an exclusive resource, it will be delayed if the resource is already held by a high security process, otherwise it will immediately be granted the resource. The presence or absence of the delay can be used to encode information by a high security process that is conspiring to pass on information to the low security process.

Covert channels that use the database system's *physical resources* as the medium for passing on information are relatively straightforward to tackle – for example, by introducing "noise" in the form of dummy transactions that make use of these resources. However, this approach is impractical for covert channels that use *data* as the medium (for example, presence or absence of a lock on a pre-determined data item). This is because, unlike physical resources which are typically few in number, the number of data items is usually enormous, especially in a database system. In fact, in heavily loaded systems, noise at the physical resources may be generated "for free", but this will probably never be the case for data since it is trivial to insert an additional data item that is of relevance only to the conspiring transactions. Therefore, *explicitly making data access covert-channel-free is more vital than doing the same for resource access.*

Covert channels based on data can be prevented by providing *higher* priority to the low security transaction whenever a data conflict occurs between a low security transaction and a high security transaction. Taking this approach ensures that low security transactions do not "see" high security transactions and are therefore unable to distinguish between their presence or absence. This notion is formalized in [6] as *non-interference*. From a database system perspective, it translates to implementing a *concurrency control* mechanism that supports the non-interference feature. In this paper, we quantitatively investigate the performance implications of secure concurrency control in the context of a firm-deadline real-time database system.

Real-Time Database Security

A secure real-time database system has to *simultaneously* satisfy two requirements, namely, provide security and minimize the number of missed transaction deadlines. Unfortunately, the mechanisms for achieving the individual goals often work at cross-purposes [8]. In a real-time database system, high priority is usually given to transactions with earlier deadlines in order to help their timely completion. On the other hand, in secure database systems, low security transactions are given high priority in order to avoid covert channels (as described above). Now consider the situation wherein a high security process submits a transaction with a tight deadline in a secure real-time database system. In this case, it becomes difficult to assign a priority since assigning a high priority may cause a security violation whereas assigning a low priority may result in a missed deadline.

One approach, used by Son et al in [4, 16, 17], to address the above problem is to adaptively *tradeoff* security for timeliness depending on the state of the system. Our view, however, is that for many applications security is an "all-or-nothing" issue, that is, it is a *correctness* criterion. In comparison, the number of missed deadlines is a *performance* issue. Therefore, in our research work, we are investigating the problem of how to minimize the number of missed transaction deadlines *without* compromising security. As a first step towards achieving this goal, we have conducted a detailed simulation study to evaluate what impact the choice of concurrency control protocol has on the real-time performance. Our simulation model captures a real-time database system with an open transaction arrival process. Transactions are assigned security levels and have corresponding restrictions on their data accesses. Each transaction also has a deadline and the deadline is "firm" – that is, transactions which miss their deadlines are considered to be worthless and are "killed" (immediately discarded from the system without being executed to completion).

Secure Real-Time Concurrency Control

In recent years, several concurrency control (CC) protocols that are specially tailored for real-time database systems have been developed. These include prioritized variants of two-phase locking (2PL) such as Wait Promote and High Priority [3], and prioritized variants of optimistic concurrency control (OPT) such as Sacrifice and Wait [9]. These algorithms were primarily designed to minimize the number of missed transaction deadlines and have been evaluated on this basis in [3, 9].

There are significant differences between the real-time environment in which the above concurrency control algorithms were compared and the *secure* real-time environment. In particular, the following issues need to be considered: First, not all real-time concurrency control algorithms may satisfy the non-interference property mentioned earlier. Second, there are multiple transaction classes corresponding to the various security clearance levels. Third, the data access patterns of transactions are constrained by the Bell–LaPadula model. Fourth, conflicts are resolved based on *both* security considerations and timeliness considerations. Finally, there is the question of *class fairness*, that is, how evenly are the missed deadlines spread across the transactions of the various clearance levels.

Due to the above differences, the performance profiles of real-time concurrency control algorithms need to be reevaluated in the secure domain – we address this issue here.

Dual Approach

A feature of the secure environment is that there are *two* categories of data conflicts: *inter-level* and *intra-level*. Inter-level conflicts are data conflicts between transactions belonging to different security clearance levels whereas intra-level conflicts are data conflicts between transactions of the same level. The important point to note here is that *only inter-level conflicts can result in security violations*, not intra-level conflicts. This opens up the possibility of using *different* concurrency control strategies to resolve the different types of conflicts. In particular, we can think of constructing mechanisms such that inter-level conflicts are resolved in a secure manner while intra-level conflicts are resolved in a timely manner. The advantage of this *dual* approach is that the real-time database system can maximize the real-time performance, by appropriate choice of intra-level CC protocol, without sacrificing security. In contrast, the tradeoff approach mentioned earlier requires the application to *compromise* on security in order to achieve enhanced real-time performance. We investigate here the performance of various combinations of concurrency control mechanisms for resolving inter-level and intra-level data conflicts.

Contributions

In this paper, we quantitatively investigate the performance implications of guaranteeing security in a firm-deadline real-time database system. Our main contributions are the following:

1. We identify which among the previously proposed real-time concurrency control protocols are capable of providing protection against both direct and indirect (covert channels) means of unauthorized access to data. That is, which protocols support the concept of *non-interference*.

2. Using a detailed simulation model of a firm-deadline real-time database system, we profile the real-time performance of a representative set of secure concurrency control protocols. Our simulations consider a variety of security-classified transaction workloads and system configurations. To isolate and quantify the performance effects of supporting covert channel security, we also evaluate the performance of the CC protocols in the context of a baseline system that prevents *direct* unauthorized access, but not covert channels (that is, it only supports the Bell–LaPadula restrictions). Our experiments show that a prioritized optimistic concurrency control protocol, OPT-WAIT, provides the best overall performance.

3. We evaluate the effectiveness of a novel dual approach to secure transaction concurrency control wherein simultaneously different CC mechanisms are used for guaranteeing security and for improving real-time performance, respectively. In particular, we investigate the performance of various combinations of concurrency control mechanisms for resolving inter-level and intra-level data conflicts. Our results show that some of these *hybrid* concurrency control algorithms perform even better than OPT-WAIT.

2 Related Work

The design of secure CC protocols in the context of *conventional* database systems has been investigated by several research groups (see [20] for a survey). In comparison, little attention has been given to developing secure CC protocols for *real-time* database systems. The only work that we are aware of in this area is a series of papers by Son et al [4, 13, 16, 17, 19]. In particular, a concurrency control protocol that attempts to *balance* the dual requirements of security and timeliness is presented in [4, 16, 17]. In their scheme, transactions *dynamically* choose between 2PL-HP [3], an (unsecure) real-time version of 2PL, and S2PL [15], a secure (non-real-time) version of 2PL. The goal of the protocol is to tradeoff security for real-time performance with the tradeoff depending on the state of the system and the application's requirements.[1] In contrast, in our work, we have assumed that full security is a *fundamental* requirement and that it is not permissible to improve the real-time performance at the cost of security.

In [13], a concurrency control protocol that ensures *both* security and timeliness is proposed. For this scheme, however, the RTDBS is required to maintain *two* copies of each data item. Further, transactions are required to obtain *all* their data locks *before* starting execution (i. e., strict static locking). These requirements limit the applicability of the protocol. In our work, we consider more general database environments where all data items are single-copy and transactions acquire data locks dynamically.

Another feature of their work is that it is primarily addressed towards "soft-deadline" applications, that is, real-time applications in which there is value to completing tasks even *after* their deadlines have expired. In contrast, we have concentrated on firm-deadline applications. The type of deadline has a significant impact on both the performance evaluation model and on the interpretation of the results, as observed earlier for (unsecure) real-time transaction concurrency control [3, 9].

3 Secure Concurrency Control Protocols

As mentioned in the Introduction, assigning priorities in a secure real-time database system is rendered difficult due to having to satisfy multiple functionality requirements. In our study, since we assume that security is a correctness requirement, the database system is forced to assign transaction priorities based primarily on security clearance levels and only secondarily on deadlines. In particular, we assign priorities as a vector P = (LEVEL, INTRA), where LEVEL is the transaction security clearance level and INTRA is the value assigned by the priority mechanism used *within* the level. We assume that security levels are numbered from zero upwards, with zero corresponding to the lowest security level. Further, priority comparisons are made in lexicographic order with lower priority values implying higher priority.

With the above scheme, transactions at a lower security level have higher priority than all transactions at a higher security level, a necessary condition for non-interference. For the intra-level priority mechanism, any priority assignment that results in good real-time performance can be used. For example, the classical Earliest Deadline assignment where transactions with earlier deadlines have higher priority than transactions with later deadlines. In this case, the priority vector would be P = (LEVEL, DEADLINE).

In conjunction with the above priority assignment, it would seem at first glance that, in principle, any real-time concurrency control protocol could be used in a secure RTDBS and that the actual choice of protocol would be based only on the relative performance of these protocols. However, *not all the previously proposed real-time CC algorithms are amenable to supporting security requirements*. For example, consider the 2PL Wait Promote algorithm proposed in [3]: This protocol, which is based on 2PL, incorporates a *priority inheritance* mechanism [18] wherein, whenever a requester blocks behind a lower-priority lock holder, the lock holder's priority is promoted to that of the requester. In other words, the lock holder *inherits* the priority of the lock requester. The basic idea here is to reduce the blocking time of high priority transactions by increasing the priority of conflicting low priority lock holders (these low priority transactions now execute faster and therefore release their locks earlier).

The Wait Promote approach is *not* suitable for secure real-time database systems. This is because it permits the blocking of high priority transactions by low priority transactions which violates the requirement of *non-interference* between the transactions of different security levels (as mentioned in the Introduction, non-interference means that low

[1] The tradeoff approach, and alternative schemes to implement the tradeoff, have also been considered in the Secure Alpha project [8], which investigated the interactions between security and timeliness in the context of a distributed real-time operating system.

security transactions should not be able to distinguish between the presence or absence of high security transactions).

To generalize the above observation, a real-time CC protocol that permits, to even a limited extent, high priority transactions to be adversely affected by low priority transactions, a phenomenon known as *priority inversion* in the real-time literature [18], cannot be used in a secure RTDBS. Apart from Wait Promote, other examples of real-time CC algorithms that fall into this category include 2PL-CR [3], 2PL-OS/BI [2] and WAIT-50 [9].

In the remainder of this section, we briefly present a representative set of concurrency control protocols that, by virtue of being completely free from priority inversion, could be used to resolve conflicts in a secure real-time database system. These protocols use either locking or optimistic concurrency control as the basic regulatory mechanism.

3.1 2PL High Priority

The 2PL High Priority (2PL-HP) scheme [3] modifies the classical strict two-phase locking protocol (2PL) [5] by incorporating a priority conflict resolution scheme which ensures that high priority transactions are not delayed by low priority transactions. In 2PL-HP, when a transaction requests a lock on a data item that is held by one or more higher priority transactions in a conflicting lock mode, the requesting transaction waits for the item to be released (the wait queue for a data item is managed in priority order). On the other hand, if the data item is held by only lower priority transactions in a conflicting lock mode, the lower priority transactions are restarted and the requesting transaction is granted the desired lock.[2] Note that 2PL-HP is inherently deadlock-free if priorities are assigned uniquely (as is usually the case in real-time database systems).

3.2 OPT-SACRIFICE

The OPT-SACRIFICE algorithm [9] modifies the classical forward (or broadcast) optimistic concurrency control protocol (OPT) [14] by incorporating a *priority sacrifice* mechanism. In this algorithm, a transaction that reaches its validation stage checks for conflicts with currently executing transactions. If conflicts are detected and one or more transactions in the conflict set is a higher priority transaction, then the validating transaction is restarted – that is, it is sacrificed in an effort to help the higher priority transactions make their deadlines. Otherwise, the transaction is allowed to commit, restarting in the process the lower priority transactions (if any) in its conflict set.

3.3 OPT-WAIT

The OPT-WAIT algorithm [9] modifies the forward OPT protocol by incorporating a *priority wait* mechanism. Here, a transaction that reaches validation and finds higher priority transactions in its conflict set is "put on the shelf", that is, it is made to wait and not allowed to commit immediately. This gives the higher priority transactions a chance to make their deadlines first. While a transaction is waiting on the shelf, it is possible that it may be restarted due to the commit of one of the conflicting higher priority transactions. If at any time during its shelf period, the waiting transaction finds no higher priority transactions remaining in its conflict set, it is committed, restarting in the process the lower priority transactions (if any) in its conflict set.

[2] A new reader is allowed to join a group of lock-holding readers only if its priority is higher than that of *all* the waiting writers.

3.4 S2PL

A secure locking-based protocol called Secure 2PL (S2PL) was recently proposed in [15]. The basic principle behind Secure 2PL is to try to simulate the execution of conventional 2PL *without* blocking the actions of low security transactions by high security clearance transactions. This is accomplished by providing a new lock type called *virtual lock*, which is used by low security transactions that develop conflicts with high security transactions. The actions corresponding to setting of virtual locks are implemented on *private versions* of the data item (similar to optimistic concurrency control). When the conflicting high security transaction commits and releases the data item, the virtual lock of the low security 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 T_i, lists of the active transactions that precede or follow T_i in the serialization order. The complete details are given in [15].[3]

Note that Secure 2PL may not perform well in the real-time domain since it does not include any real-time-specific features. We include it here for two reasons: First, it serves as a baseline against which to compare the real-time CC algorithms. Second, we use it in one of the "dual approach" protocols evaluated in this study.

4 Dual Approach

In this section, we move on to discussing our new dual approach to secure real-time concurrency control. As mentioned in the Introduction, a feature of the secure environment is that there are two categories of conflicts: inter-level and intra-level. This opens up the possibility of using *different* concurrency control strategies to resolve the different types of conflicts. In particular, we can think of constructing mechanisms such that inter-level conflicts are resolved in a secure manner while intra-level conflicts are resolved in a timely manner. For example, S2PL could be used for inter-level conflicts while OPT-WAIT could be used to resolve intra-level conflicts. The advantage of this *dual* approach is that the real-time database system can maximize the real-time performance without sacrificing security.

At first glance, it may appear that using multiple concurrency control mechanisms in parallel could result in violation of the transaction serializability requirement. This could happen, for example, if the serial orders enforced by the individual mechanisms were to be different. A detailed study of a generalized version of this problem is presented in [21], wherein the transaction workload consists of a mix of transaction classes and the objective is to allow each transaction class to utilize its preferred concurrency control mechanism. They propose a database system architecture wherein intra-class conflicts are handled by the class's preferred concurrency control manager while inter-class conflicts are handled by a new software module called the Master Concurrency Controller (MCC) that interfaces between the transaction manager and the multiple concurrency control managers. The MCC itself implements a complete concurrency control mechanism. A single global serialization order is ensured in the entire database system by using a *Global Ordering*

[3] In our implementation, we have had to partially modify Secure 2PL since the algorithm (as described in [15]) does not eliminate non-interference under all circumstances – the details of the modifications are available in [7].

Table 1: Simulation Model Parameters

$DBSize$	Number of pages in the database
$ClassLevels$	Number of Classification Levels
$ArrivalRate$	Transaction arrival rate
$ClearLevels$	Number of Clearance Levels
$SlackFactor$	Slack Factor in Deadline assignment
$TransSize$	Average transaction size (in pages)
$WriteProb$	Page write probability
$NumCPUs$	Number of processors
$NumDisks$	Number of disks
$PageCPU$	CPU time for processing a data page
$PageDisk$	Disk service time for a data page

Scheme. The details of this scheme and the proof of its correctness are given in [21].

For our study, we assume use of the above architecture. In this framework, a S2PL/OPT-WAIT combination, for example, would correspond to using S2PL at the Master Concurrency Controller for resolving inter-level conflicts, and using OPT-WAIT as the local concurrency controller within each security level for resolving intra-level conflicts.

5 Simulation Model

In the previous section, we discussed various secure concurrency control protocols. To evaluate the real-time performance of these algorithms, we developed a detailed simulation model of a firm-deadline real-time database system, similar to that described in [9]. A summary of the key model parameters is given in Table 1.

In our model, the system consists of a shared-memory multiprocessor DBMS operating on disk-resident data (for simplicity, we assume that all data is accessed from disk and buffer pool considerations are therefore ignored). The database is modeled as a collection of $DBSize$ pages that are uniformly randomly distributed across all of the disks. The database is equally partitioned into $ClassLevels$ security classification levels (for example, if the database has 1000 pages and the number of classifications is 5, pages 1 through 200 belong to level 1, pages 201 through 400 belong to level 2, and so on). Transactions are generated in a Poisson stream with rate $ArrivalRate$ and each transaction has an associated security clearance level and a firm completion deadline. A transaction is equally likely to belong to any of the $ClearLevels$ security clearance levels. (For simplicity, we assume in this study that the categories (e.g., Secret, Public) for data classification and transaction clearance are identical). Deadlines are assigned using the formula $D_T = A_T + SF * R_T$, where D_T, A_T and R_T are the deadline, arrival time and resource time, respectively, of transaction T, while SF is a slack factor. The resource time is the total service time at the resources that the transaction requires for its data processing. The $SlackFactor$ parameter is a constant that provides control over the tightness/slackness of transaction deadlines.

A transaction consists of a sequence of page read and page write accesses. The number of pages accessed by a transaction varies uniformly between half and one-and-a-half times the value of $TransSize$. The $WriteProb$ parameter determines the probability that a transaction operation is a write. Due to security reasons, each transaction can only access data from a specific segment of the database, and page requests are generated by uniformly randomly sampling (without replacement) from the database over this range. The permitted access range is determined by both the security clearance level of the transaction and the desired operation (read or write), and is according to the Bell–LaPadula specifications: a transaction cannot read (resp. write) pages that are classified higher (resp. lower) than its own clearance level. A transaction that is restarted due to a data conflict has the same clearance level, and makes the same data accesses, as its original incarnation. If a transaction has not completed by its deadline, it is immediately killed (aborted and discarded from the system).

A transaction 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. We assume that the RTDBS has sufficient buffer space to allow the retention of updates until commit time.

The physical resources of the database system consist of $NumCPUs$ processors and $NumDisks$ disks. There is a single common queue for the CPUs and the service discipline is Pre-emptive Resume, with preemptions being based on transaction priorities. Each of the disks has its own queue and is scheduled according to a Head-Of-Line (HOL) policy, with the request queue being ordered by transaction priority.[4] The $PageCPU$ and $PageDisk$ parameters capture the CPU and disk processing times per data page, respectively.

6 Experiments and Results

In this section, we present the performance results from our simulation experiments comparing the various secure CC protocols in a firm-deadline RTDBS environment. The primary performance metric of our experiments is $MissPercent$, which is the percentage of input transactions that the system is *unable* to complete before their deadlines. We compute this percentage on a *per-clearance-level* basis also. MissPercent values in the range of 0 to 20 percent are taken to represent system performance under "normal" loads, while MissPercent values in the range of 20 percent to 100 percent represent system performance under "heavy" loads [9]. Only statistically significant differences are discussed here [7].

An additional performance metric is *ClassFairness* which captures how evenly the missed deadlines are spread across the transactions of the various clearance levels. To compute this we use, for each class i, the formula $Fairness_i = \frac{CommitTrans_i/InputTrans_i}{CommitTrans/InputTrans}$. In this formula, $CommitTrans_i$ and $InputTrans_i$ are the number of committed transactions and the number of input transactions, respectively, of class i, while $CommitTrans$ and $InputTrans$ are the total number of committed transactions and the total number of input transactions, respectively, across all classes. With this formulation, a protocol is ideally fair, if the fairness value is 1.0 for all classes. Fairness values greater than one and lesser than one indicate positive bias and negative bias, respectively.

The transaction priority assignment used for the secure protocols in the experiments described here is P = (LEVEL, DEADLINE), thereby ensuring that there are no covert channels since a low security transaction is delayed only by transactions of its own level or those of lower security levels.

[4] For simplicity, our model uses the non-interference method for eliminating covert channels at the physical resources also – an alternative method is the "noise" technique mentioned in the Introduction.

Table 2: Baseline Parameter Settings

DBSize	1000 pages	*NumCpus*	10
ClassLevels	2 (Secret, Public)	*NumDisks*	20
ClearLevels	2 (Secret, Public)	*PageCPU*	10 ms
SlackFactor	4.0	*PageDisk*	20 ms
TransSize	16 pages	–	–
WriteProb	0.5	–	–

6.1 Comparative Protocol

To help isolate and understand the performance cost that occurs due to having to eliminate covert channels, we have also simulated the performance achievable in the *absence* of covert channel security requirements. That is, the performance achievable if only Bell–LaPadula conditions had to be satisfied. For this scenario, a priority assignment of P = DEADLINE is used. In the following experiments, we will refer to the performance achievable under this scenario as **DIRECT** since the Bell–LaPadula conditions prevent direct unauthorized access to data.

6.2 Experiment 1: Resource and Data Contention

The settings of the workload parameters and system parameters for our first experiment are listed in Table 2. These settings were chosen with the objective of having significant data contention and resource contention in the system, thus helping to bring out the performance differences between the various concurrency control protocols.

In this experiment, there are two security levels: *Secret* and *Public*. For this system, Figures 2a and 2b show the MissPercent behavior as a function of the overall transaction arrival rate. In Figure 2a, the overall miss percentages of the fully secure algorithms and their DIRECT (Bell–LaPadula) counterparts is profiled. We see here that at normal loads the performance of the secure algorithms is worse than that of their DIRECT counterparts. In contrast, under heavy loads the performance of the secure algorithms is actually better than that of the DIRECT algorithms. The reason for this is that while the Earliest Deadline priority assignment is excellent for a set of tasks that *can* be completed before their deadlines, it becomes progressively worse as the task set overloads its capacity [10]. In this situation, the secure protocols feature of grouping the transactions into prioritized levels means that Earliest Deadline is operational within *smaller* sets of transactions, leading to improved performance at higher loads. In summary, although elimination of covert channels results in performance degradation at normal loads, it reduces the miss percentage under heavy loads.

Focusing on the secure real-time algorithms, we observe first in Figure 2a that the performance of 2PL-HP is significantly worse than that of the optimistic algorithms, OPT-WAIT and OPT-SACRIFICE (denoted by OPT-SCR in the legend). In fact, 2PL-HP's performance is no better than that of S2PL which, as mentioned in Section 3, is a non-real-time protocol! The poor performance of 2PL-HP is primarily because of its "wasted restarts" problem, which was its main drawback in unsecure real-time CC also [9]: A transaction may be restarted by a higher priority transaction that later misses its deadline. This means that the restart did not result in the higher priority transaction meeting its deadline. In addition, it may cause the lower priority transaction to miss its deadline as well, apart from wasting the resources invested in the transaction prior to its restart.

The effect of the wasted restarts problem is *magnified* in the secure domain for the following reason: In unsecure real-time CC, a transaction that is close to its deadline would usually not be restarted since it would have high priority. However, in the secure model, where the transaction *level* is also a factor in the priority assignment, Secret transactions that are close to their deadlines may still be restarted due to data conflict with a Public transaction.

Moving on to OPT-SACRIFICE, we find that there is a change of performance behavior in the secure environment in that the gap between its performance and that of 2PL-HP is *more* than that observed for unsecure real-time CC [9]. The main problem for OPT-SACRIFICE in the unsecure domain was that it suffered from "wasted sacrifices" (sacrifices for a transaction that is eventually killed). The effect of this problem is *diminished* in the secure domain due to the access pattern restrictions of the Bell-LaPadula model: The definition of conflict in forward optimistic concurrency control is that a conflict exists between the validating transaction V and an executing transaction E if and only if the intersection of the write set of V and the current read set of E is non-empty. For the LaPadula model, where *blind writes* are permitted due to the "read-below, write-above" paradigm, optimistic algorithms will correctly conclude that there is no conflict between items that are in the intersection of the write set of V and the write set of E but not in the read set of E. In fact, it is easy to see that *a validating Secret transaction will never have conflicts with executing Public transactions* in this model. Therefore, the possibility of wasted sacrifices decreases as compared to the unsecure domain. Note that for 2PL-HP, however, blind-writes can unnecessarily result in write-write conflicts and cause either blocking or restarts, thereby further deteriorating its performance.

Turning our attention to OPT-WAIT, we see that it provides the *best* performance across the entire loading range. This is because it derives, similar to OPT-SACRIFICE, the above-mentioned benefits arising out of the Bell-LaPadula access restrictions. In addition, it suffers neither from wasted restarts nor from wasted sacrifices. Instead, all restarts are useful in that they are made "on demand" and at the commit time of a higher priority transaction.

Finally, moving on to S2PL, we find that it manages to perform on par with 2PL-HP in spite of not being deadline cognizant. This is due to its "optimistic-like" feature of *virtual commit*, which considerably reduces the amount of blocking associated with 2PL. This phenomenon is similar to that seen in [9], wherein a conventional (non-real-time) optimistic protocol performed better than locking-based real-time protocols.

In Figure 2b, we present the miss percentages of the various concurrency control protocols on a *per-security-level* basis. This graph clearly shows how the high-security Secret transaction class (dashed lines) suffers much more than the Public transaction class (dotted lines) to satisfy the goal of avoiding covert channels. Figure 2c provides statistics about the corresponding breakup of the "restarts ratio" (the average number of restarts of a transaction) on a level basis. We see here that Secret transactions are restarted much more often than Public transactions under normal loads. Under heavy loads the number of restarts decrease for Secret transactions since resource contention, rather than data contention, becomes the more dominant reason for these transactions missing their deadlines.

In Figure 2d, we present a different view of the transaction restarts picture. Here, the restarts of Secret trans-

Figure 2a: Covert security cost

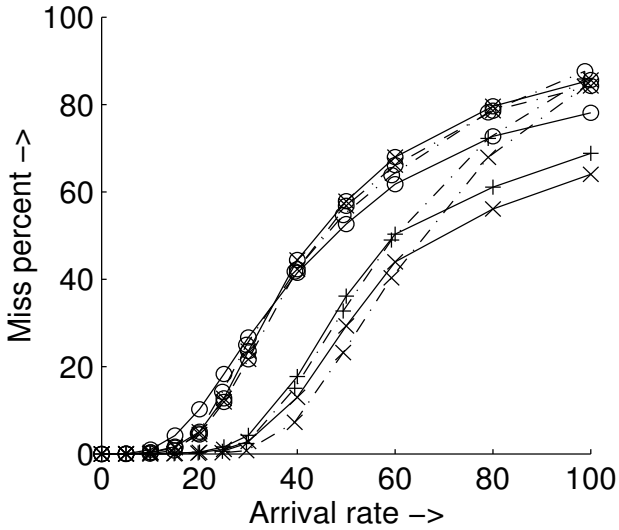


Figure 2b: Level miss percent

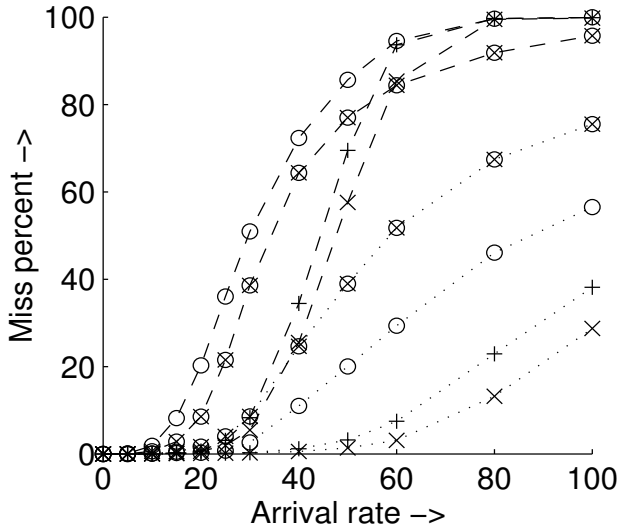


Figure 2c: Level restart ratio

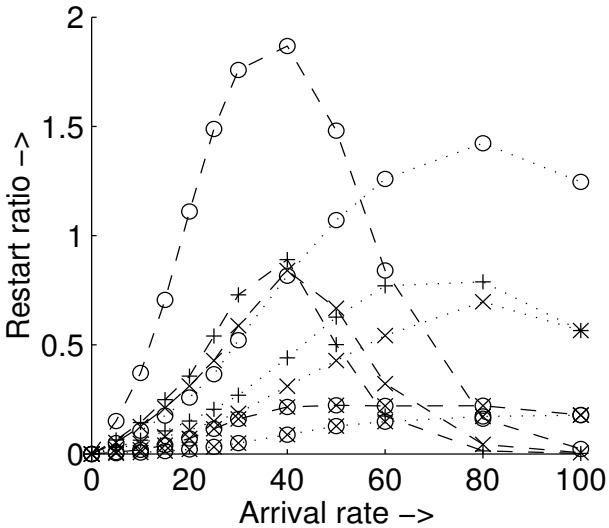


Figure 2d: Secret restart ratio

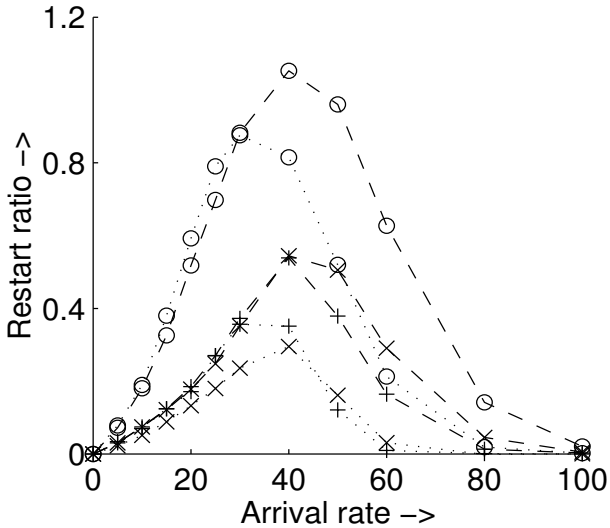
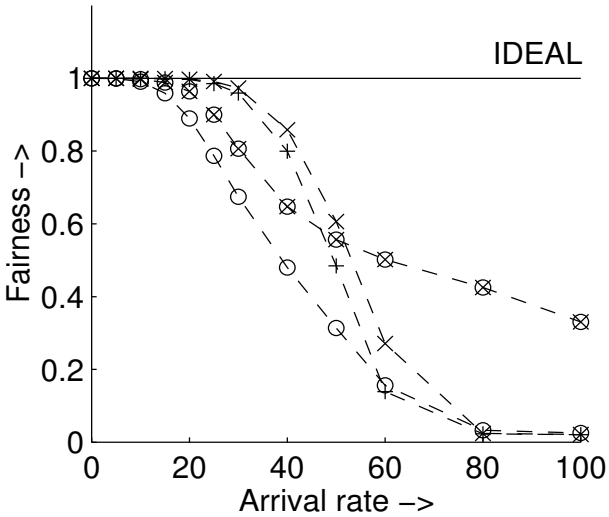
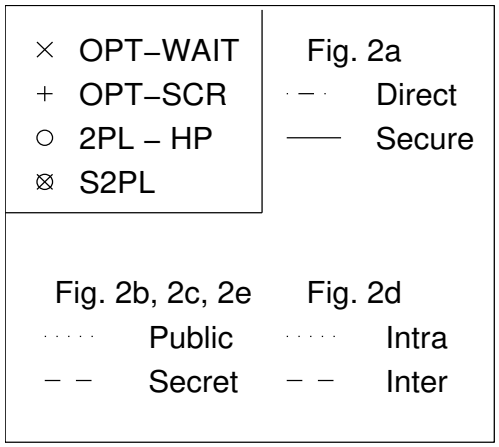
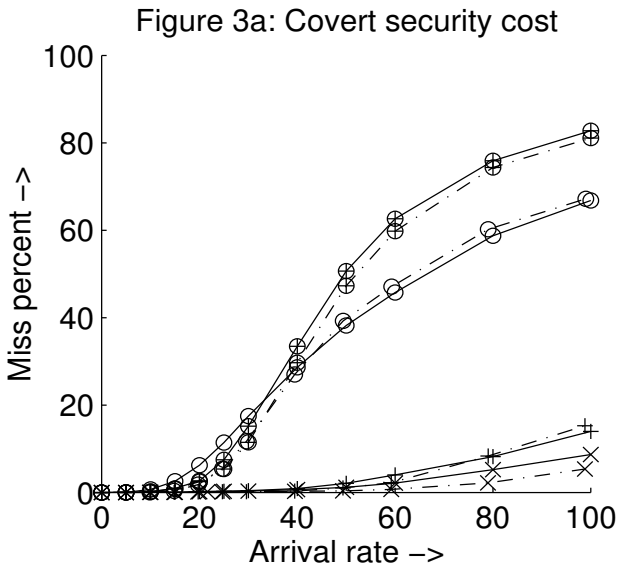
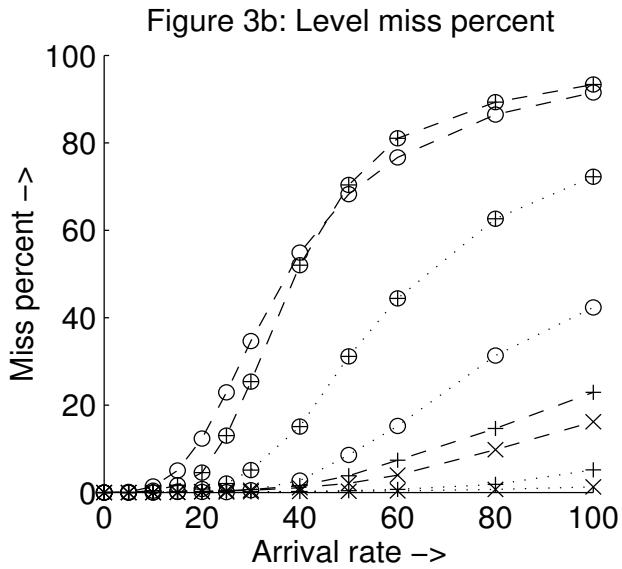


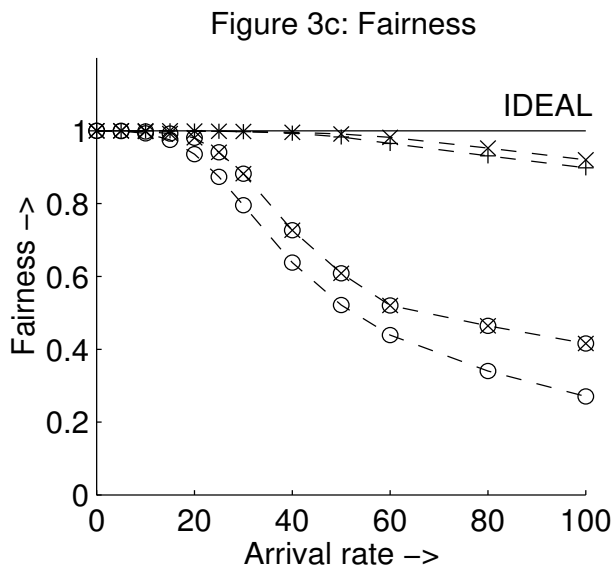
Figure 2e: Fairness

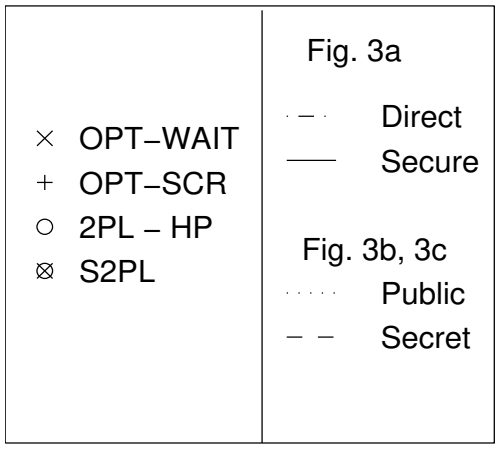












actions are categorized into those caused by Public transactions (i. e., *inter-level* restarts) and those caused by Secret transactions (i. e., *intra-level* restarts). Note that this breakup is meaningful only for Secret transactions since all restarts are intra-level for Public transactions. The graph clearly shows that Secret transactions suffer more from inter-level conflicts (dashed lines) than from intra-level conflicts (dotted lines) over most of the loading range.

Finally, in Figure 2e, we plot the *fairness factor* of each CC protocol for the Secret transaction class. We observe that at light loads when virtually *all* transactions make their deadlines, all the concurrency control protocols are (trivially) fair. As the loading increases, however, the protocols become increasingly unfair since they selectively miss the deadlines of Secret transactions to accommodate the Public transactions. With regard to the relative fairness of the secure real-time algorithms, the graph clearly shows that OPT-WAIT and OPT-SACRIFICE provide much better fairness than 2PL-HP, and that OPT-WAIT is the best overall. It may seem in Figure 2e that, at high loads, S2PL is more fair than OPT-WAIT. Note, however, that this is really a "virtual fairness" since it arises out of S2PL, due to its non-real-time nature, *missing* a large fraction of the Public transactions, rather than out of *meeting* the deadlines of more Secret transactions.

In summary, for the workload and system configuration considered in this experiment, OPT-WAIT provides the lowest miss percentage, both on an overall basis and on a per-level basis, *and* the best overall fairness.

6.3 Experiment 2: Pure Data Contention

The goal of our next experiment was to isolate the influence of data contention on the performance of the concurrency control protocols. For this experiment, therefore, the resources were made "infinite", that is, there is no queueing for these resources [1]. The remaining parameter values are the same as those used in the baseline experiment. The performance results for this system configuration are presented in Figures 3a through 3c. We observe in these figures that the differences in the relative performance of the various protocols increases as compared to those seen in the previous

experiment. The overall (Figure 3a) as well as the per-level (Figure 3b) miss percentages of OPT-WAIT are considerably better than those of 2PL-HP. Here, OPT-WAIT does better than 2PL-HP for two reasons: First, the basic wasted restarts problem of 2PL-HP occurs here too and is magnified due to the higher level of data contention. Second, the blocking component of 2PL-HP reduces the number of transactions that are making progress in their execution. This blocking causes an increase in the average number of transactions in the system, thus generating more conflicts and a greater number of restarts. With OPT-WAIT, however, transactions are never blocked for data access.

Moving on to the performance of OPT-SACRIFICE, we observe that its performance becomes even closer to that of OPT-WAIT as compared to the previous experiment. The reason for this is that the wasted utilization arising out of its wasted sacrifices problem has no impact here since resource contention is not an issue.

Finally, in Figure 3c, which profiles the fairness factors of the protocols, OPT-WAIT and OPT-SACRIFICE are almost ideally fair over most of the loading range since they miss very few deadlines overall. In contrast, 2PL-HP and S2PL are noticeably unfair with increasing loading levels.

6.4 Experiment 3: Increased Security Levels

A two-security-level system was modeled in the previous experiments. In our next experiment, we investigated the performance behavior for a *five-security-level* system, where the levels are *TopSecret, Secret, Confidential, Classified and Public*. The remaining parameter values are the same as those used in Experiment 1. The results for this experiment are shown in Figures 4a through 4e (for graph clarity, the results for only 2PL-HP and OPT-WAIT are presented).

In Figure 4a, we see that there is a greater difference between the performance of the secure algorithms and their DIRECT counterparts at normal loads, as compared to the equivalent two-level experiment (Experiment 1). This is because in a five-level system, priority is much more level-based than deadline-based, thereby denying Earliest Deadline its ability to complete most transactions in a feasible set under normal loads. Under heavy loads, however, the smaller sizes of the transaction sets in each level results in Earliest Deadline performing well for the low security transactions. (This feature of Earliest Deadline was used in the Adaptive Earliest Deadline scheduling algorithm described in [10] where transactions are split up into prioritized groups with the size of the highest priority group set equal to an estimate of the maximum number of transactions that could be successfully completed by Earliest Deadline.)

In Figures 4b and 4c, we plot the miss percentage on a per-security-level basis for OPT-WAIT and 2PL-HP, respectively. These graphs clearly show the extent to which the miss percentages are skewed among the various transaction security levels, with Top Secret transactions having the most number of missed deadlines and Public transactions having the least. The graphs also show that OPT-WAIT's performance is better than that of 2PL-HP for *every* transaction security level.

In Figures 4d and 4e, the fairness factors for the top four security levels (Top Secret, Secret, Confidential and Classified) are plotted on a per-security-level basis for OPT-WAIT and 2PL-HP, respectively. These figures clearly show that as the loading factor increases, progressively more and more security classes become discriminated against by the lowest security class (Public). We also find that OPT-WAIT's performance is more fair than that of 2PL-HP for *every* transaction security level.

In summary, just as in the two-security-level experiment, we find that OPT-WAIT provides the lowest miss percentage, both on an overall basis and on a per-level basis, and the maximum fairness (this observation regarding OPT-WAIT is true also with regard to the OPT-SACRIFICE and S2PL protocols whose results were not presented here).

6.5 Experiment 4: Dual Approach

As mentioned earlier, we have experimented with a *dual* approach to secure real-time concurrency control where inter-level conflicts are handled by one protocol while intra-level conflicts are handled by a different protocol. In Figures 5a through 5c we show the performance of three dual systems for the same environment as that of Experiment 1. The combinations (in inter/intra order) are 2PL-HP/OPT-WAIT, OPT-WAIT/2PL-HP, and S2PL/OPT-WAIT, which we will refer to as HP-WAIT, WAIT-HP, and S2PL-WAIT, respectively. For the sake of comparison, the performance of a pure OPT-WAIT protocol is also shown in these graphs.

Focusing our attention on the WAIT-HP and HP-WAIT dual protocols, we first observe in Figure 5a, which compares the overall miss percentages of the protocols, that the performance of both these approaches is considerably worse than that of the pure OPT-WAIT protocol. The reason that OPT-WAIT remains the best among them is that 2PL-HP is a wasteful algorithm, as seen in the previous experiments, and therefore "dilutes" the effect of OPT-WAIT in both the dual protocols.

We also observe in Figure 5a that that the performance of WAIT-HP is worse than that of HP-WAIT throughout the loading range. The reason for this is that, in the secure system, the number of intra-level conflicts are significantly more than the number of inter-level conflicts. Therefore, the algorithm which is used to handle intra-class conflicts has more effect on the overall miss percentage than the protocol used to handle inter-class conflicts. In WAIT-HP, it is 2PL-HP which handles intra-class conflicts and this results in worse performance than that of HP-WAIT, which uses OPT-WAIT to handle this category of conflicts.

The miss percentages of the protocols on a per-security-level basis is provided in Figure 5b and the fairness factors are shown in Figure 5c. An interesting feature in these graphs is that at high loads, the fairness of WAIT-HP is *greater* than that of OPT-WAIT – this is the first time in all the experiments discussed so far that a real-time protocol has improved on OPT-WAIT's fairness performance. The reason that this happens is the following: In WAIT-HP, due to 2PL-HP being used for intra-class conflicts, many of the Public transactions are so busy "fighting" each other that they don't ever reach the end of their execution, which is when the OPT-WAIT policy of checking for inter-class conflicts comes into play. Therefore, Secret transactions suffer much less restarts from the Public transactions. This is clearly seen in Figure 5b where the miss percentage of the Public transactions for WAIT-HP is quite high as compared to the corresponding numbers for the other protocols.

Moving on to the S2PL-WAIT dual protocol, we find that, unlike the other two dual protocols, it performs better than OPT-WAIT, especially at lower loading levels. For example, at an arrival rate of 40 transactions per second, S2PL-WAIT more than halves the miss percentage suffered by OPT-WAIT (Figure 5a). The reason that this combination works well is that Secure 2PL handles inter-class con-

Figure 4a: Covert Security Cost

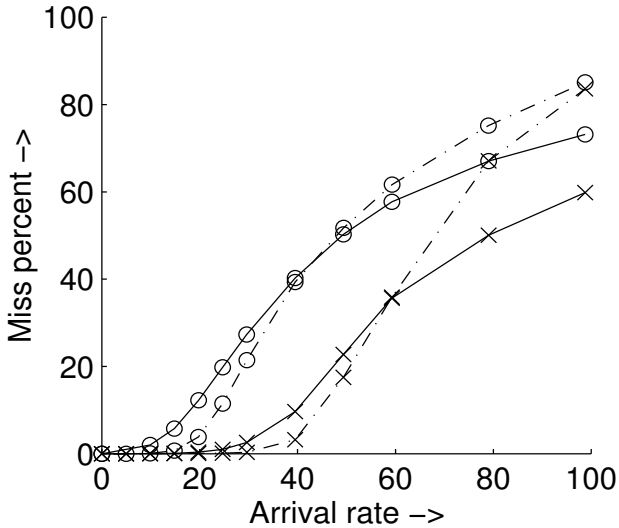


Figure 4b: Level wise (OPT–WAIT)

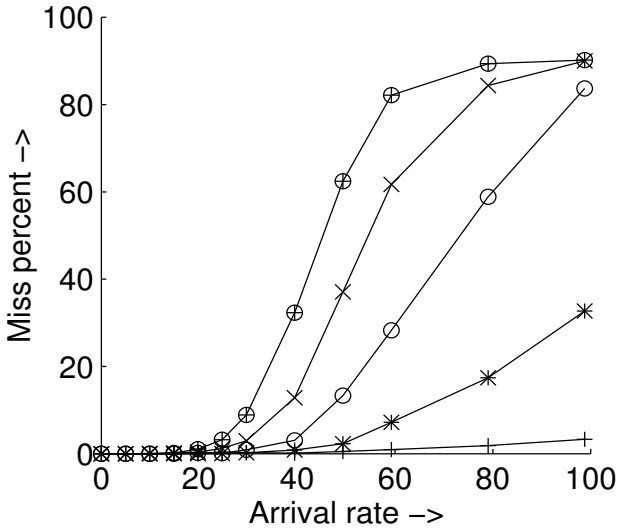


Figure 4c: Level wise (2PL–HP)

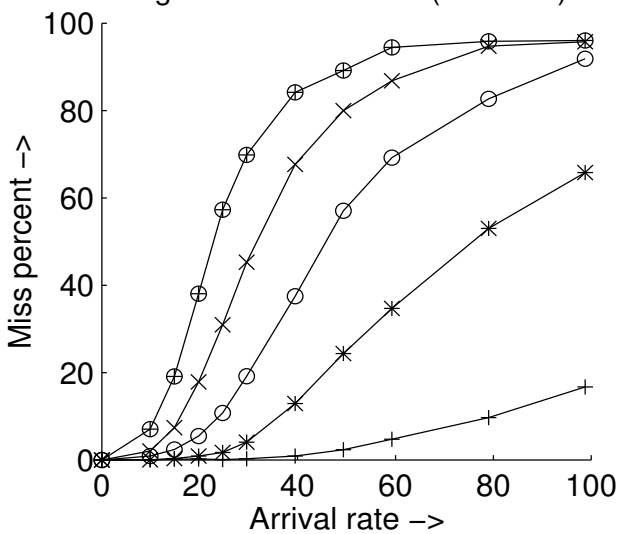


Figure 4d: Fairness (OPT–WAIT)

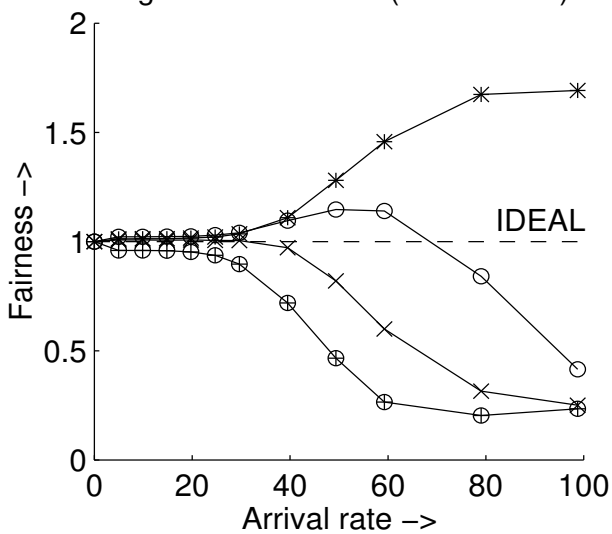


Figure 4e: Fairness (2PL–HP)

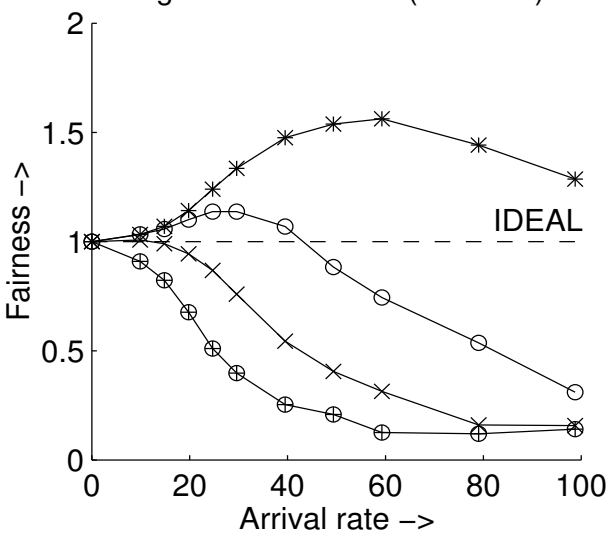
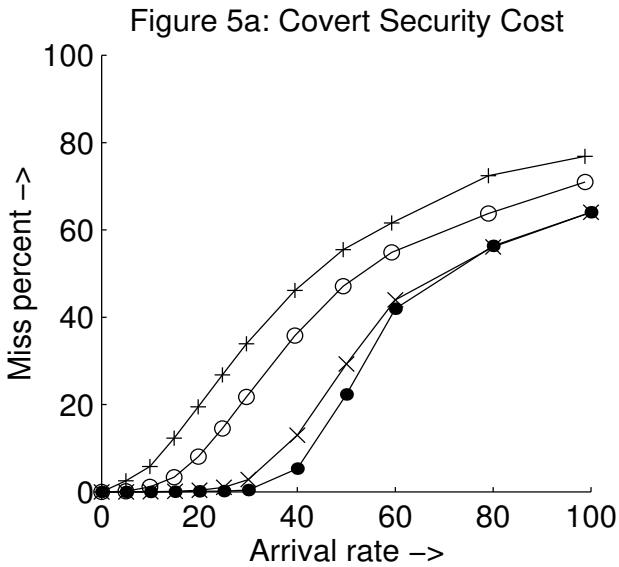
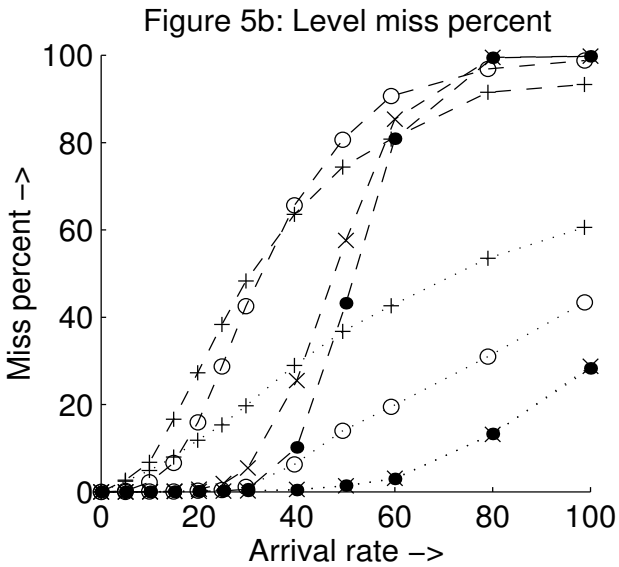
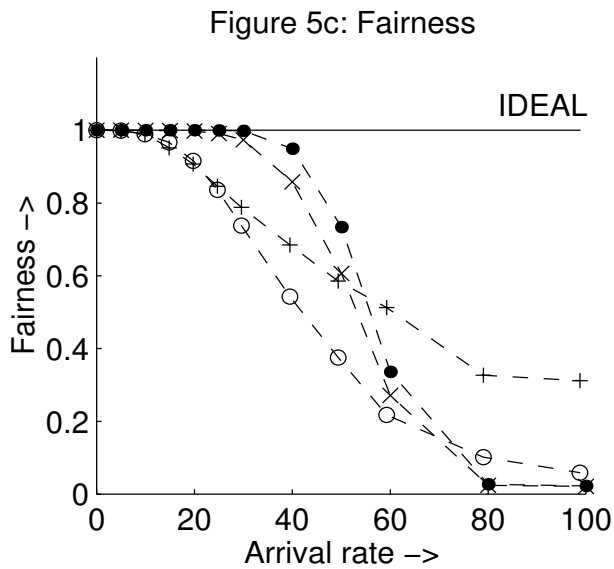
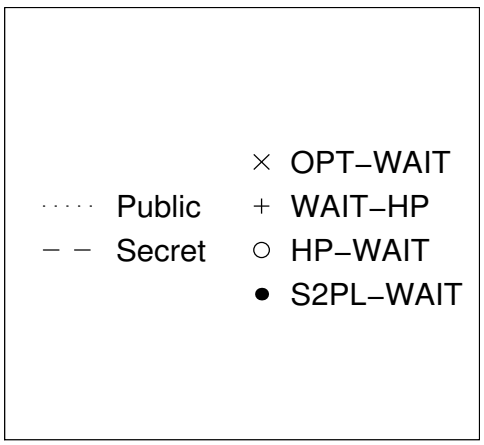


Fig. 4a	Fig. 4b – 4e
× OPT–WAIT	+ Public
○ 2PL–HP	✳ Classified
	○ Confidential
Direct	× Secret
Secure	⊕ Top Secret









flicts *without* resorting to restarts unlike in WAIT-HP or HP-WAIT. This means that Secret transactions suffer much less in this environment (as confirmed in Figures 5b and 5c). At the same time, using OPT-WAIT for handling intra-class conflicts helps to derive the inherent good real-time performance associated with this protocol. At high loads, S2PL-WAIT performs almost identically to OPT-WAIT because, in this region, the primary reason for transactions missing their deadlines is resource contention, rather than data contention – therefore, the virtual commit feature of S2PL rarely provides the intended benefits.

In summary, this experiment shows that by carefully choosing the right combination of protocols in the dual approach, we can design hybrid concurrency control algorithms that provide even better miss percent and fairness performance than OPT-WAIT. This highlights the power and flexibility that is provided by the dual approach. In fact, it may be possible to develop hybrid algorithms that perform even better than S2PL-WAIT by appropriately choosing the constituent protocols.

7 Other Experiments

We conducted several other experiments to explore various regions of the workload space. In particular, we evaluated the sensitivity of the results to the database size, number of security levels, deadline slack factor, etc. In many secure systems, the Bell–LaPadula model of "read below, write above" is further restricted to allow only "read below", that is, blind writes by low security transactions to high security data are disallowed. We conducted experiments to evaluate the performance behavior of the concurrency control protocols under this model also.

The complete details and results of the above experiments are available in [7]. Our general observation was that the relative performance behaviors of the protocols in these other experiments remained qualitatively similar to those seen in the experiments described here. That is, OPT-WAIT performed the best among the individual protocols, while S2PL-WAIT provided the best overall performance with respect to both the individual protocols and the dual combination protocols.

8 Conclusions

In this paper, we have quantitatively investigated the performance implications of maintaining covert-channel-free security in a firm-deadline real-time database system. Unlike previous studies, which used a *tradeoff* approach between security and timeliness, we have considered security as an "all-or-nothing" issue, that is, as a *correctness* criterion. In comparison, the number of missed deadlines is a *performance* issue. Therefore, our study investigates the problem of how to minimize the number of missed transaction deadlines *without* compromising security. To the best of our knowledge, this is the first detailed study of real-time database security in the firm-deadline context.

We first identified that, in order to satisfy the requirement of non-interference, only those real-time concurrency control protocols that are free from priority inversion can be used in a secure RTDBS. This requirement ruled out several previously proposed real-time CC protocols, including algorithms such as 2PL Wait Promote [3] and WAIT-50 [9].

Then, using a detailed simulation model of a firm-deadline RTDBS, we studied the relative performance of the secure versions of the 2PL-HP, OPT-SACRIFICE and OPT-WAIT real-time concurrency control algorithms; a non-real-time secure algorithm, S2PL, was also included in the evaluation suite. The performance of these algorithms was also evaluated for a baseline system where only direct unauthorized access, but not covert channels, is prevented.

Our experiments showed that, under normal loads, the overall miss percent of the secure system is worse than that of the direct system, whereas under heavy loads, it is the other way around. Within the secure system, the performance of high-security transactions was significantly worse than that of the low-security transactions. Among the secure concurrency control protocols, OPT-WAIT performed best in minimizing the miss percentages on both an overall basis and on a per-level basis. Moreover, it exhibited the maximum degree of fairness. These results show that OPT-WAIT, which provided excellent performance in traditional (unsecure) real-time concurrency control, continues to perform well even in the secure real-time domain.

Finally, we proposed a novel dual approach to secure concurrency control wherein different concurrency control algorithms are used to resolve inter-level conflicts and intra-level conflicts. A global serialization order was ensured, in spite of having multiple CC algorithms operating simultaneously, by using the system architecture proposed in [21]. The dual combinations of HP-WAIT and WAIT-HP were implemented and evaluated – they both generally performed worse than pure OPT-WAIT. However, the dual combination of S2PL-WAIT performed better than OPT-WAIT, especially at lower miss percent levels. This is because S2PL is a non-restart-oriented algorithm unlike both OPT-WAIT and 2PL-HP, and therefore ensures reduction of the harm done to high-security transactions.

Another advantage of the dual approach, not exploited here, is that the separation of security and timeliness concerns makes it possible to use even *non-secure* real-time CC algorithms (e.g., Wait-Promote, WAIT-50) for resolving intra-level conflicts! The dual approach therefore empowers the use, even in the secure RTDBS domain, of the rich set of real-time CC algorithms developed during the last decade.

Acknowledgements

This work was supported in part by a research grant from the Dept. of Science and Technology, Govt. of India.

References

[1] R. Agrawal, M. Carey and M. Livny, "Concurrency control performance modeling: Alternatives and implications", *ACM Trans. on Database Systems*, 12(4), December 1987.

[2] D. Agrawal, A. El Abbadi and R. Jeffers, "Using Delayed Commitment in Locking Protocols for Real-Time Databases", *Proc. of ACM SIGMOD Conf.*, June 1992.

[3] R. Abbott and H. Garcia-Molina, "Scheduling Real-time Transactions: A Performance Evaluation", *ACM Trans. on Database Systems*, September 1992.

[4] R. David, S. Son and R. Mukkamala, "Supporting Security Requirements in Multilevel Real-Time Databases", *Proc. of IEEE Symp. on Security and Privacy*, May 1995.

[5] K. Eswaran et al, "The Notions of Consistency and Predicate Locks in a Database Systems", *Comm. of ACM*, November 1976.

[6] J. Goguen and J. Meseguer, "Security Policy and Security Models", *Proc. of IEEE Symp. on Security and Privacy*, 1982.

[7] B. George and J. Haritsa, "Secure Processing in Real-Time Database Systems", *TR-97-02, DSL/SERC, Indian Institute of Science*, 1997.

[8] I. Greenberg et al, "The Secure Alpha Study (Final Summary Report)", *Tech. Report ELIN A012*, SRI International, June 1993.

[9] J. Haritsa, M. Carey and M. Livny. "Data Access Scheduling in Firm Real-Time Database Systems", *Real-Time Systems Journal*, 4(3), 1992.

[10] J. Haritsa, M. Livny and M. Carey, "Earliest Deadline Scheduling for Real-Time Database Systems", *Proc. of 12th IEEE Real-Time Systems Symp.*, December 1991.

[11] W. Lampson, "A Note on the Confinement Problem", *Comm. of ACM*, October 1973.

[12] L. LaPadula and D. Bell, "Secure computer systems: Unified Exposition and Multics Interpretation", *The Mitre Corp.*, March 1976.

[13] R. Mukkamala and S. Son, "A Secure Concurrency Control Protocol for Real-Time Databases", *Proc. of Annual IFIP WG 11.3 Conference of Database Security*, August 1995.

[14] J. Robinson, "Design of concurrency control protocols for transaction processing systems", *Ph.D. Thesis*, Computer Sciences Dept., Carnegie Mellon University, 1982.

[15] S. Son and R. David, "Design and Analysis of a Secure Two-Phase Locking Protocol", *Proc. of Intl. Computer Software and Applications Conf.*, November 1994.

[16] S. Son, R. David and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems", *Proc. of Annual IFIP WG 11.3 Conference of Database Security*, August 1995.

[17] S. Son, R. David, B. Thuraisingham, "Improving Timeliness in Real-Time Secure Database Systems", *SIGMOD Record*, Special Issue on Real-Time Database Systems, March 1996.

[18] L. Sha, R. Rajkumar and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization", *Tech. Rep. CMU-CS-87-181*, Depts. of CS, ECE and Statistics, Carnegie Mellon University, 1987.

[19] S. Son and B. Thuraisingham, "Towards a Multilevel Secure Database Management System for Real-Time Applications", *Proc. of IEEE Workshop on Real-Time Applications*, May 1993.

[20] B. Thuraisingham and H. Ko, "Concurrency Control in Trusted Database Management Systems: A Survey", *SIGMOD Record*, 22(4), December 1993.

[21] S. Thomas, S. Seshadri and J. Haritsa, "Integrating Standard Transactions in Real-Time Database Systems", *Information Systems*, 21(1), March 1996.