Maintaining Security in Firm Real-Time Database Systems

Quazi N. Ahmed and Susan V. Vrbsky
Department of Computer Science
The University of Alabama
Tuscaloosa, AL 35487-0290
U.S.A.

Abstract

Many real-time database systems, such as military institutions and government agencies, are contained in environments that exhibit restricted access of information, where mandatory access control for security is required. Hence, in addition to timing constraints, real-time database systems have security constraints. Conventional multi-level secure database models are inadequate for time-critical applications and conventional real-time database models do not support security constraints. The objective of this work is to incorporate security constraints in real-time database systems in such a way that not only is security achieved, but achieving security does not degrade real-time performance significantly in terms of deadlines missed. We propose a new optimistic concurrency control algorithm for secure firm real-time databases. Results show that the algorithm performs fairly well in terms of security and timeliness compared to a non-secure algorithm. We argue and show that achieving more security does not necessarily mean more sacrifice in real-time performance.

1. Introduction

Real-time database systems [3,4], such as military command control, communication, avionics, radar tracking, and managing automated factories, have timing constraints. These timing constraints are typically in the form of deadlines that require a transaction to be completed by a specified time. In such environments, multiple users share the same database, and some of the users can have restricted access to information from the database. Hence, it is necessary to provide database security for real-time databases. Conventional database security models are not adequate for time-critical applications [5]. A model for database security in real-time databases must address satisfying the timing constraints and security constraints. Major efforts to design secure databases have focused on databases without timing constraints. It is only recently that work [2,5,6] has been reported in the area of database security for real-time databases. The objective of this work is to study the factors involved in secure concurrency control, develop a suitable concurrency control algorithm, implement and study the performance of the algorithms with a real-time database system simulation.

Mandatory access control is implemented by multilevel secure (MLS) databases [7,8,9] and can be used in many environments that exhibit hierarchical propagation of information, such as military institutions, government agencies, hospitals, etc. Many real-time database systems are contained in such environments, and, hence, are considered for this study. Mandatory access control is applicable on large amounts of information that requires firm protection in an environment where the subjects are grouped into clearances and objects are grouped by their classifications. In a multilevel secure database model objects have security levels and subjects have clearance levels. According to the Bell-LaPadula simple property [7], a subject can read a certain data object if the subject's clearance level dominates the object's security level. According to the Bell-LaPadula star-property [7], a subject can write to an object if the subject's clearance level is dominated by the object's security level. The two Bell-LaPadula properties prevent the direct flow of information from objects and/or subjects at a higher security clearance level to subjects at a lower level, and are the basis for all MLS models. However, the concurrent execution of transactions results in contention for data objects, and as a result, it is possible to have an indirect flow of information from objects at higher levels to subjects at lower levels due to a covert channel [13].

A secure real-time database system has to simultaneously satisfy two goals, provide security and ensure that the deadline miss percentage is minimized. In many occasions, these two goals conflict and to achieve one goal is to sacrifice the other. For example, suppose we have a high priority transaction at a high security level and a low priority transaction at a low security level, and there is a data conflict between them. (Priority is based on deadline, where a higher priority means an earlier deadline.) If the high priority transaction gets the data and blocks the low security transaction, then although the priority is maintained, a covert channel is introduced. The lower security level can determine there are transactions at higher levels and may even be able to receive information from the length of the delay. If the reverse is allowed to happen, security will be maintained but the priority will be violated. Whether to maintain security or priority is dependent upon the system. If the system requires that security be maintained regardless of the deadline of the conflicting transactions, then conflict must be resolved in favor of security. If the system allows a compromise between security and priority, then the goal is to maintain as much security as possible without raising the deadline miss percentage significantly.

In firm real-time database systems transactions are removed as soon as they miss their deadlines. In this paper, we study security for firm real-time database systems. Our main investigation is to identify how to minimize the number of missed deadlines for real-time transactions while avoiding covert channels wherever possible. Our main contributions in this paper can be summarized as follows:

- We present a new concurrency control algorithm in this study. The algorithm recognizes the security as well as the priority of real-time transactions. The algorithm can be used for systems where security can be compromised for priority and vice versa.
- We introduce two metrics to measure the level of security maintained in secure real-time database systems.
- We show that by using our algorithm, it is possible to achieve an almost 100% covert channel free system without a great deal of sacrifice in real-time performance in terms of missed deadlines.

The rest of the paper is organized as follows. In section 2, we discuss related work in secure real-time concurrency control and the secure real-time factor. In section 3, we present the secure concurrency control algorithm and metrics for security maintenance, and in section 4 we describe the simulation model and the results. In section 5 we present our conclusions.

2. Secure real-time concurrency control

Real-time transactions can have soft, hard or firm deadlines [3,10]. A transaction with a soft deadline may miss its deadline, yet the result produced still has some value (although the value monotonically decreases with time after the deadline). Soft transactions are allowed to complete even if a deadline is missed. On the other hand, if a hard deadline of a transaction is missed, the consequence can be catastrophic. The value assigned to a result when a deadline is missed is negative. For a firm deadline, the consequences of missing a deadline are less severe than a hard deadline, but a result produced after a deadline is useless, and is assigned a value of 0. For both firm and hard deadlines, if a deadline is not met, the transaction is discarded as soon as the deadline is missed.

2.1 Related work

Few works that address the security of real-time databases are described in [2, 5, 6]. In [5] a concurrency control strategy is presented which trades off security for improved timeliness if the system does not provide the desired deadline miss percentage. They use a measure called “capacity” to adjust the covert channel to get better real-time performance. In [6] a secure two-phase locking algorithm is used to allow partial violations of security for improved timeliness. Decisions are made concerning the trade off between security and meeting deadlines by comparing two measures to resolve conflicts: the security factor and the deadline miss factor. This comparison is used to determine if a lower level transaction should be aborted or proceed when it conflicts with a higher level transaction. If the security properties are more important, then any conflict is resolved in favor of the lower level transaction, otherwise if meeting deadlines is more important, then the higher priority transaction is given precedence. Our work also considers the need to tradeoff between security and timeliness but shows that security can be achieved with negligible sacrifice in timeliness.

In [2], security of firm real-time databases is addressed. In this study, security is viewed as a correctness criterion, and the number of missed deadlines as a performance issue. They do not trade off between missed deadlines and security, but instead propose to minimize the number of missed deadlines without compromising security through the choice of concurrency control strategy. They examine the performance of such strategies as a two-phase locking priority scheme, a prioritized optimistic concurrency control algorithm and a new approach, called the dual approach. The dual approach utilizes different concurrency control strategies depending on whether transactions are at the same security level or at different security levels. In contrast, our algorithm is capable of handling transactions both at the same and different security levels.

2.2 Secure real-time factor

The concurrency control algorithm for a secure real-time database system must use security levels of transactions as well as their deadlines to resolve data conflicts. Not only that, the difference in security levels of transactions also needs to be considered when a security conflict is to be resolved. If one transaction is at the highest security level and the another one is at the lowest and a covert channel is introduced between them, then the severity of the covert channel will be higher than the one where transactions are at adjacent security levels. None of the previous works discussed in section 2.1 recognizes the difference in security levels as a measure to resolve a security conflict. In secure real-time database systems, where timeliness is one of the goals to be achieved, one cannot afford to sacrifice it for some covert channel not severe enough to be concerned about. Thus, we believe that difference in security level is an important issue for determining whether a covert channel has to be closed by possibly sacrificing timeliness. To be able to determine the severity or consequences of security violations, we introduce the following "covert channel property" for secure real time database systems.

Covert Channel Property: The greater the difference of security levels between two transactions at data conflict, the greater the severity or the consequence if a covert channel is introduced.

The covert channel property indicates that consequence is proportional to the difference in security or access levels. In other words, the greater the difference in access levels, the more important it is to maintain security and close the covert channel. If two conflicting transactions are at two extreme security levels, the consequence of opening a covert channel is the maximum. If the two conflicting transactions are at two adjacent security levels the consequence is the minimum. Of course, if the two transactions are at the same security level, there is no covert channel, and hence no consequence. We introduce a metric to measure the consequence of introducing a covert channel, to be known as the *Covert Channel Factor* (CCF). The CCF is obtained by normalizing the difference of access levels between the two conflicting transactions:

$$\mathrm{CCF} = \frac{\text{Difference in access levels}}{\text{Maximum difference possible}}$$

$$= \frac{\text{Difference in access levels}}{\#\,\text{of access levels} - 1}$$

The maximum value of CCF is 1 when the two transactions are at two terminal access levels. The minimum value of CCF is $\frac{1}{\#\,\text{of access levels} - 1}$ when the two transactions are at two consecutive security levels. Of course, if two transactions are at the same security level, CCF is obviously zero meaning no covert channel.

Sometimes, security levels have a partial ordering instead of a complete ordering and, as a result, not all the security levels are comparable to each other. For example, assume we have 5 security levels namely, TOP SECRET, SECRET1, SECRET2, UNCLASSIFIED1 and UNCLASSIFIED2 and, the ordering is TOP SECRET > SECRET1, TOP SECRET > SECRET2, SECRET1 > UNCLASSIFIED1 and SECRET2 > UNCLASSIFIED2. In this case, certain levels are not directly comparable, e.g., SECRET1 and SECRET2, SECRET1 and UNCLASSIFIED2 etc. In order to be able to compute the CCF, the ordering between every possible pair of security levels needs to be known. This can be achieved by converting the partial order to a tree structure. For each ordering A > B, we create a parent node A and a child node B. If one of the nodes already exists, we only create the new one and make the appropriate hierarchy with the existing one. In this example, this can be done by having TOP SECRET as the root, SECRET1 and SECRET2 as the root's two children, UNCLASSIFED1 as the child of SECRET1 and UNCLASSIFIED2 as the child of SECRET2. Security levels at same depth of the tree can be considered as same levels and hence no covert channels exist between them. The difference between two security levels will be equal to the difference in their depths. Assuming the root has the minimum depth and the leaves have the maximum depth, one security level, A will be considered higher than another security level, B if depth(A) < depth(B). This work assumes that a partial ordering under MLS model is convertible to a single tree. Any ordering that leads to more than one tree is out of the scope of this paper.

3. Secure concurrency control algorithm

As mentioned earlier, secure real-time database systems have to satisfy the security constraints in addition to the timing constraints. Security can be considered as a correctness criteria where security must be enforced. It can also be thought of as a compromising criteria with timeliness where security can be sacrificed to achieve fewer deadline misses. The algorithm we present here supports the latter type. Although, the algorithm can achieve 100% covert channel free system, it can not guarantee that this is the case always. Later we will discuss why it is not possible to provide such a guarantee. Before describing our algorithm, we give a brief introduction to the existing non-secure algorithm upon which our algorithm is based. The existing algorithm is the Optimistic Concurrency Control (OPT) [11]. This algorithm and our algorithm, to be called the Secure OPT, are described in the following sub-section.

3.1 Secure OPT

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 [11]. The validating transaction restarts if the test fails. The main goal of real-time optimistic algorithms is to prevent low priority validating transactions from committing one-sidedly when they conflict with higher priority transactions. There are two different optimistic strategies for real-time transactions, priority sacrifice and priority wait. In priority sacrifice, known as the OPT-SACRFICE, the validating transaction restarts if any higher priority transaction exists in the conflicting set (the set of currently executing transactions that conflict with the validating transaction.) In the priority wait, known as the OPT-WAIT, the validating transaction waits if there is any higher priority transaction in the conflicting set. These algorithms do not check the security levels in the transactions and there can be covert channels when a validating transaction commits and aborts the transactions in the conflicting set. For example, in the priority-sacrifice algorithm, if the validating transaction's access level and priority are higher than those of any of the transactions' in the conflicting set, a covert channel is introduced when the validating transaction commits aborting all in the conflicting set. Similarly, in the priority wait algorithm, if the validating transaction is blocked and there exists a transaction of a higher access level in the conflicting set, a covert channel is also introduced. We examine the different conflicts that can occur between a validating transaction (VT) and those in the conflicting set (CS) and present their resolutions in the following.

1. *$\forall T \in CS$ if deadline(T)>deadline(VT) and $\forall T \in CS$ if access level(T)>access level(VT) then commit VT and abort all T in CS.* Since VT is at the highest priority and lowest security level, both the priority and the security are maintained.
2. *$\forall T \in CS$ if deadline(T)<deadline(VT) and $\forall T \in CS$ if access level(T)<access level(VT) then block VT.* VT is at the lowest priority and highest security level. Again priority is maintained and no covert channel is introduced.
3. When none of the above conditions are met, a covert channel will be introduced no matter which side is aborted or blocked unless the size of the conflicting set is 1. Hence, using the optimistic strategy the system cannot be made 100% covert channel free if some validation phase falls in this case (case 3) and there are more than one transactions in the conflicting set. We want to resolve this case such that CCF is minimized and priority is also recognized whenever possible.
    a) *Minimize the CCF.* For this, we calculate the CCF when the validating transaction is aborted and also calculate the CCF when the conflicting set is aborted. The two CCFs are compared to see which option introduces more of a covert channel.
    b) *Consider the priority.* We need to consider the priority of the transactions in the conflicting set in order not to lose real-time performance. To do this we simply check if there is any transaction in the conflicting set with a higher priority than the VT.

Considering the above two cases, a conflict can be resolved in several ways. However, we want to minimize the CCF and also to recognize the priority. Thus, if one of the above conditions (a or b) are in favor of the VT, VT is allowed to commit.

```
    If (CCF for aborting CS <CCF for aborting VT)
                OR
      (no high priority transaction exists in CS)
    Then
          Commit the VT and abort the CS.
    Else
      Abort the VT.
```

For example, assume that {T1 = (4, 10.40), T2 = (5, 11.57), T3 = (2, 12.67)} is the conflicting set where each member is a transaction presented as an ordered pair (access level, deadline). Assume also that the validating transaction is VT=(3,10.54). If the total number of access levels is 6 then the CCF for aborting the conflicting set is =(3-2)/5=1/5 due to the only security conflict with T3. Aborting T1 and T2 will not cause any covert channel as they are at higher security levels. The CCF for aborting the VT is =(4-3)/5 +(5-3)/5=1/5+2/5 =3/5. The first *if* condition is satisfied and VT commits. This option recognizes both security and priority, and if either the CCF is minimized (case a) or the priority is considered (case b), a VT is allowed to commit. The objective of this option is to have as many commits as possible and increase real-time performance. Maintaining security is not of prime importance here. By considering both the CCF and the priority, this option maximizes the number of commits and therefore, it is an *optimal* option. However, if only security is of concern, we use only the first *if* condition (case a) above) to see which side to abort. We call this the *secure option.*

```
If (CCF for aborting CS<CCF for aborting VT) Then
        Commit the VT and abort the CS
Else
        Abort the VT
```

3.2 Metrics of security maintenance

We now introduce two metrics or security factors to measure the security maintenance of a system. One metric keeps track of the number of times security has been maintained. The other one recognizes the differences between the access levels.

$$\text{Security factor } 1 = \frac{\#\text{ of times security is maintained}}{\text{Total number of security conflicts}}$$

Security factor 2

$$= \frac{\text{Sum of the difference in access levels for conflicts having security maintained}}{\text{Sum of the difference in access levels in all security conflicts}}$$

Both the metrics are suitable for measuring the performance of the system. Depending upon the system, one metric might be more appropriate than the other. If only the number of conflicts maintained is of concern the first factor is appropriate. The second factor is appropriate in systems where difference in access-levels is crucial. In this study, we choose the security factor 2.

3.3 Metric of priority maintenance

In order to express the level of priority maintenance in a system, we use the following priority maintenance factor.

Priority maintenance factor =

$$= \frac{\text{\# of times priority is maintained}}{\text{Total number of data conflicts}}$$

This metric is used to determine how priority maintenance affects the real-time performance.

4. Simulation model

This section outlines the structure and details of our simulation model used to evaluate the performance of our concurrency control algorithms for real-time database systems. Central to our simulation model is a single-site main memory database system operating on a single processor. The database is modeled as a collection of data pages in memory. The simulation consists of three main components: a Transaction Manager (TM), a CPU Manager (CM), and a Log Manager (LM). The TM is responsible for issuing lock requests, CM for granting CPU access, and LM for log disk access. The service discipline used for the queues is Earliest Deadline First (EDF) [12] without preemption. Each transaction consists of multiple operations each of which can be either read or write. If the operation is read, then the accessed page is not updated. The write operation updates the accessed page and an entry is written into the log buffer.

When an operation of a transaction makes a data access request, i.e., lock request on a data object to the TM, the request goes through concurrency control to obtain a lock on the data object. If the request is granted, the transaction requests CPU access to the CM. If the CPU is free, the request is granted and the transaction does the CPU computation. After the CPU computation, if there are any more operations left, the transaction proceeds with the next operation and makes a lock request to the TM. If all operations are done, the transaction requests log disk access to the LM and if access is granted, it writes the log buffer to the log disk and commits.

If the request for the lock is denied, the transaction will be placed into a block queue. The blocked transaction will be awakened when the requested lock is released.

Table 1: System Resource Parameters

Parameter	Explanation	Value
DBSIZE	Number of data pages in the database	400
CPU_TIME	CPU time for processing a data page	5 msec
WRITE_PROB	Probability that an operation is write	0.5
MAXACCESS	# access levels	6

4.1 Parameters of simulation model

Table 1 gives the system resource parameters. The parameter CPU_TIME is the time to process a page by a CPU. The simulation does not explicitly take into account the time required for accessing the transaction manager, the CPU manager, and the log manager. It is assumed that those times are included in the time required to access the resources, i.e., the CPU, and the log disk.

Table 2 summarizes the workload parameters that characterize the transactions and the system workload. Transactions' inter-arrival rates are exponentially distributed. The *Rate* parameter specifies the mean rate of transaction arrivals. The *TransSize* determines the mean number of operations in the transactions determined from a normal distribution with mean of *TransSiz*e. The actual data objects or pages accessed by each operation are uniformly distributed across the whole database. The *LogDelay* is the time required for writing a log buffer to the log disk. The *RestartDela*y is the overhead for roll back when a transaction is aborted. The parameters *MinSlac*k and *MaxSlack* are used to set the lower and upper bound of transactions' deadlines. The following deadline assignment formula is used to assign the deadline to the arrived transaction.

Deadline = Arrival time + Uniform(MinSlack,MaxSlack)
** Execution time*

The *Arrival tim*e is the time of arrival of each transaction. The execution time of a transaction is calculated from the data requirements in all the operations using *TranSize*, *CPU_TIME*, and *LogDelay*.

Table 2: Workload Parameters

Parameters	Meaning	Value
Rate	Arrival rate of the transactions	[5,50]
TransSize	Average transaction size	6
LogDelay	Overhead for writing log buffer to the log disk	5 msec
RestartDelay	Overhead for restarting	5 msec
MinSlack	Minimum slack factor	2
MaxSlack	Maximum slack factor	8

4.2 Experiments and results

The simulation program is written in C++ using the next event simulation strategy and is run for 5000 transactions. Different random seeds are used for different calls to the random number generator to make sure that the arrived transactions are exactly the same for different algorithms. In order to simulate the firm real-time system, at the beginning of each event, the system is checked to see if there is any transaction that has missed its deadline and if so, it is removed from the system. We do a detailed simulation study on firm database systems using our proposed algorithm and compare them with the existing non-secure algorithm. We discuss the effectiveness of our algorithms in terms of maintaining security and minimizing deadline miss. We also study restart ratios for both the algorithms.

Figure 1 illustrates the deadline miss percentage for the non-secure OPT and the Secure OPT algorithm with *secure* option. We use the OPT-SACRIFICE as the non-secure OPT algorithm in this study. OPT-SACRIFICE is priority cognizant and hence has a better performance over the secure OPT algorithm. The non-secure algorithm does not have any deadlines missed with arrival rates below 20. On the other hand, in the secure algorithm transactions start to miss deadlines around an arrival rate of 15 but has a low deadline miss percentage if the arrival rate is below 20. The main difference between the performance of the two algorithms is prominent between arrival rates 15 and 25, after which a majority of the transactions start to miss their deadlines in both the algorithms.

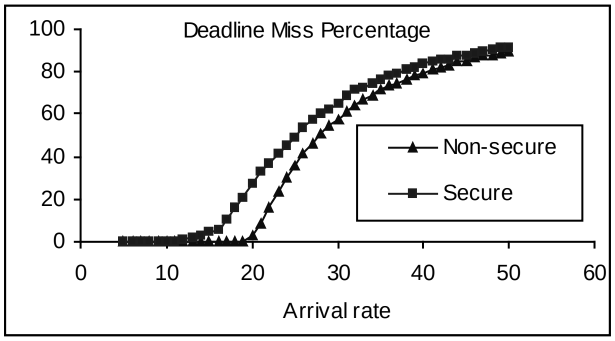

Figure 1 Comparison of Deadline Miss Percentages

The Secure OPT algorithm recognizes covert channels and, therefore, the security factor of a system is improved when the algorithm is used. This result is illustrated in the figure 2, which compares the security factors for the secure and non-secure algorithm. In the secure algorithm, the security factor is very close to 1; in contrast, in the non-secure algorithm, it is inconsistent and remains around 0.5. It is interesting to notice in Figure 1 that although we enforce security with the Secure OPT, we do not necessarily have to sacrifice the real-time performance a great deal. This is because there are many data conflicts for which security enforcement does not violate priority based on deadline. Increasing security means more data conflicts resolved in favor of lower security transactions, irrespective of their deadlines. If a lower security transaction has a later deadline, enforcing security means loss of priority. However if it happens to have an earlier deadline, then priority is maintained as well. Therefore, achieving high security (e.g. security factor close to 1) does not mean that we lose all the priority.

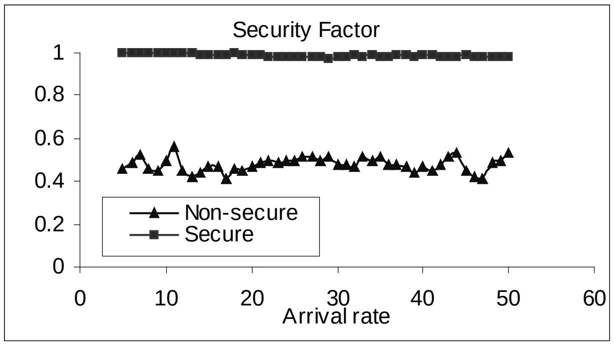

Figure 2 Comparison of Security Factors

Figure 3 shows the priority maintenance factor when the security factor is very close to 1.0. In this case priority maintenance factor is not zero but instead varies between 0.15 to 0.35. That is the reason why the real-time performance with the Secure OPT in Figure 1 is similar to the non-secure OPT. This is a very important feature of our algorithm.

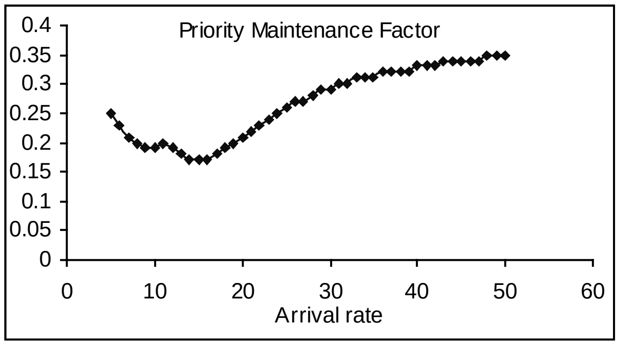

Figure 3 Priority Maintenance Factor for Secure OPT

Figure 4 shows a comparison of the restart ratios between the non-secure and the Secure OPT algorithms. For each algorithm, the restart ratio rises with the increase of arrival rate and starts to decrease after a certain arrival rate, e.g., 15-16 for secure and 21-22 for non-secure case. This can be explained as follows. Most of the transactions meet their deadlines until the arrival rate exceeds 15-16 for the secure case or 21-22 for the non-secure case. If the arrival rate is low, there are fewer conflicts and consequently fewer restarts. However as the transactions start to miss the deadlines, some of the aborted transactions may be already late and hence may not even restart because they are removed from the system as soon as they miss their deadlines. With an increased arrival rate, the number of late aborted transactions increases, which in turn decreases the number of restarts. Thus, the restart ratio increases with the increase in arrival rate until the system starts missing deadlines, after which the restart ratio starts decreasing and it continues to decrease with the subsequent increase in arrival rate.

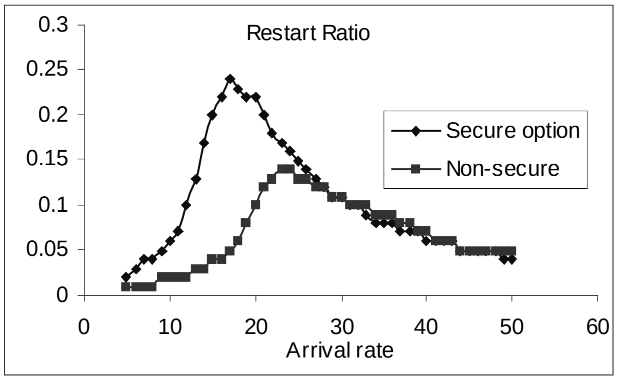

Figure 4 Comparison of Restart Ratios

5. Conclusion

In this paper we have identified the covert channel property of secure real-time databases and, we have proposed a new secure optimistic concurrency algorithm based on this property. We have implemented the secure algorithm and a non-secure algorithm and studied their performance in detail using a real-time database system simulation model. Our study covers firm real-time database systems. We have also introduced metrics to measure security in real-time database systems. Results clearly show that our algorithm performs fairly well in terms of security and timeliness compared to the non-secure algorithm. We also have shown that achieving security does not necessarily mean a great deal of sacrifice in real-time performance. A system can be made almost 100% covert channel free, but can still have a low deadline miss percentage for an arrival rate as high as 20. In the future we will examine temporal consistency, design suitable concurrency control algorithms and study their performance.

References

[1] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation" *ACM Transactions on Database Systems* 17, 1992, pp. 513-560.

[2] George, B. and J. Haritsa, "Secure Transaction Processing in Firm Real-Time Database Systems," *Proceedings SIGMOD*, Phoenix, AZ, 1997, pp. 462-473.

[3] Ozsoyoglu, Gultekin, and Richard T. Snodgrass, "Temporal and Real-Time Databases: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7 no. 4, Aug. 1995, pp. 513-532.

[4] Ramamritham, Krithi, "Real-Time Databases," *International Journal of Distributed and Parallel Databases*, 1993, pp. 199-226.

[5] Rasikan, David, Sang H. Son and Ravi Mukkamala, "Supporting Security requirements in Multilevel Real-Time Databases," *Proceedings IEEE Symposium on Security and Privacy*, Oakland, CA, May 1995, pp. 199-210.

[6] Son, Sang H., Rasikan David and Bhavani Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems," *Proceedings of Annual IFIP WG 11.3 Conference of Database Security*, Aug. 1995.

[7] Bell D. E., and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," *Technical Report*, MITRE Corporation, 1974

[8] Denning, Dorothy E., "The Sea View Security Model," *Proceedings IEEE Symposium on Security and Privacy*, Oakland, Ca, April, 1988.

[9] Jajodia, Sushil and R. Sandhu, "Toward a Multilevel Secure Relational Data Model," *Proceedings ACM SIGMOD*, Denver, Colorado, May, ACM, New York, 1991, pp. 50-59.

[10] Wolfe, V. F. and L. C. DePippo. "Real-Time Database Systems" in *Database Systems Handbook*, (Paul J. Fortier, ed.), McGraw Hill Publishers, 1997.

[11] Haritsa, J. R., M. J. Carey and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", *The Journal of Real-Time Systems*, 4, 203-241 (1992).

[12] Liu, C. L. and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, Jan 1979, pp. 46-61.

[13] Moskowitz, I. S. and Allen R. Miller, "Simple Timing Channels", *Proceedings of the IEEE Symposium on Security*, 1994, pp.56-64.

[14] Qian, Xialoei, "Inference Channel-Free Integrity Constraints in Multilevel Relational Databases", *Proceedings of the IEEE Symposium on Security*, 1994, pp. 158-167.