

# SUPPORTING SECURITY REQUIREMENTS IN MULTILEVEL REAL-TIME DATABASES

Rasikan David and Sang H. Son

Department of Computer Science  
University of Virginia  
Charlottesville, Virginia 22903

Ravi Mukkamala

Department of Computer Science  
Old Dominion University  
Norfolk, Virginia 23529

## Abstract

*Database systems for real-time applications must satisfy timing constraints associated with transactions, in addition to maintaining data consistency. In addition to real-time requirements, security is usually required in many applications. Multilevel security requirements introduce a new dimension to transaction processing in real-time database systems. In this paper, we argue that due to the conflicting goals of each requirement, trade-offs need to be made between security and timeliness. We first define capacity, a measure of the degree to which security is being satisfied by a system. A secure two-phase locking protocol is then described and a scheme is proposed to allow partial violations of security for improved timeliness. The capacity of the resultant covert channel is derived and a feedback control scheme is proposed that does not allow the capacity to exceed a specified upper bound.*

## 1 Introduction

Database security is concerned with the ability of a database management system to enforce a security policy governing the disclosure, modification or destruction of information. Most secure database systems use an access control mechanism based on the Bell-LaPadula model [2]. This model is stated in terms of subjects and objects. An object is understood to be a data file, record or a field within a record. A subject is an active process that requests access to objects. Every object is assigned a classification and every subject a clearance. Classifications and clearances are collectively referred to as security classes (or levels) and they are partially ordered. The Bell-LaPadula model imposes the following restrictions on all data accesses:

- a) Simple Security Property: A subject is allowed read access to an object only if the former's clearance is identical to or higher (in the partial order) than the latter's classification.
- b) The \*-Property: 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.

The above two restrictions are intended to ensure that there is no flow of information from objects at

a higher access class to subjects at a lower access class. Since the above restrictions are mandatory and enforced automatically, the system checks security classes of all reads and writes. Database systems that support the Bell-LaPadula properties are called multilevel secure database systems (MLS/DBMS).

The Bell-LaPadula model prevents direct flow of information from a higher access class to a lower access class, but the conditions are not sufficient to ensure that security is not violated indirectly through what are known as covert channels [7]. A covert channel allows indirect transfer of information from a subject at a higher access class to a subject at a lower access class. In the context of concurrency control approaches, a covert channel arises when a resource or object in the database is shared between subjects with different access classes. The two subjects can cooperate with each other to transfer information. An important measure of the degree to which security is compromised by a covert channel is measured by its capacity. This will be explained in greater detail in Section 2.1.

A real-time database management system (RT-DBMS) is a transaction processing system where transactions have explicit timing constraints. Typically a timing constraint is expressed in the form of a deadline, a certain time in the future by which a transaction needs to be completed. In a real-time system, transactions must be scheduled and processed in such a way that they can be completed before their corresponding deadline expires. Conventional data models and databases are not adequate for time-critical applications. They are designed to provide good average performance, while possibly yielding unacceptable worst-case response times. As advances in multilevel security take place, MLS/DBMSs are also required to support real-time requirements. As more and more of such systems are in use, one cannot avoid the need for integrating real-time transaction processing techniques into MLS/DBMSs. In [15], the security impact on real-time database systems is studied, but to the best of our knowledge, no work has been reported on developing DBMSs that are multilevel secure and that support real-time requirements.

Concurrency control is used in databases to manage the concurrent execution of operations by different

subjects on the same data object such that consistency is maintained. In multilevel secure databases, there is the additional problem of maintaining consistency without introducing covert channels. For a more detailed description of and a possible solution to the problem of concurrency control in secure databases, the reader is referred to [3]. In this paper, we concern ourselves with concurrency control mechanisms that have to satisfy both security and real-time requirements. We advance our claim that conflicts between these two requirements are inherent and hence trade-offs between them are necessary. Some background information on covert channel analysis and correctness criteria for secure schedulers is covered in Section 2. In Section 3, the problems associated with time-constrained secure concurrency control are studied. In Section 4, the secure two phase locking protocol and 2PL-High Priority are discussed. A scheme that allows partial violations of security requirements is proposed in Section 5, and the capacity of the resultant covert channel is derived. A feedback control mechanism that maintains the capacity of the system at a specified upper bound is described in Section 6. In Section 7, it is shown that the analysis and control of the single covert channel considered in Section 5 is enough to bound the capacity of all covert channels that could be potentially exploited. An implementation and performance analysis of the feedback control mechanism is explained in Section 8. Section 9 concludes the paper.

## 2 Background

Since the basis of our approach is to trade-off security for real-time requirements, in this section, we shall discuss how the degree to which security is being satisfied can be measured in terms of the capacity of possible covert channels that can be exploited. We shall also look at the correctness criteria that must be satisfied by schedulers that are free from covert channels.

### 2.1 Covert Channels and Capacity

The systematic study of covert channels began with [7]. As an example of a simple covert channel consider two processes running on a system that schedules them alternately for exactly one or two time quanta each, the choice being up to the process [9]. One process (the sender) may send information covertly to the other (receiver) by encoding successive bits in the amount of time taken for its execution. If the receiver had to wait for one quantum before its execution, then it assumes a "0" was sent; if it waits for two quanta, it assumes a "1" was sent. In the absence of any other processes, the maximum rate at which information can be transmitted through this channel is one bit per quantum (assuming only 1's are transmitted). Assuming that 0's and 1's are transmitted with equal frequency the information rate is  $1/((0.5)(1) + (0.5)(2))$ , or  $2/3$  bits per quanta. The presence of other processes in the system interferes with the transmission and can be viewed as "noise". The presence of noise decreases the information rate.

Covert channel analysis is just a subset of information theory, which is concerned with sending signals

from a transmitter to a receiver, with the possibility of noise degrading the signal fidelity. Shannon's pioneering work [13] gives an upper limit on the rate at which messages can be passed through the communication channel based solely on how noise affects the transmission of signals. This upper limit is referred to as the capacity of the channel. Let us now derive a mathematical expression for capacity in terms of the noise in the channel:

In popular usage, the term "information" is elusive to define. However, information has a precise meaning to a communication theorist, expressed solely in terms of probabilities of source messages and actions of the channel. A precise measurement of information is based on various entropy (or uncertainty) measures associated with the communication process and information exchange is defined by reduction in entropy.

Consider a discrete scalar random variable  $X$ , which can be regarded as an output of a discrete message source. Suppose the variable  $X$  can assume one of  $K$  possible outcomes, labeled  $x_i, i = 0, 1, \dots, K-1$ , with probabilities specified by  $P_i$ . The entropy of the random variable  $X$  is

$$H(X) = \sum_{i=0}^{K-1} P_i \log \frac{1}{P_i}$$

The entropy measures the "information" or "surprise" of the different values of  $X$ . For a particular value  $x_i$  the surprise is  $\log(1/P_i)$ ; if  $x_i$  happens with certainty then its surprise is zero, and if  $x_i$  never occurs its surprise is maximal at infinity. Note that base two logarithm is used so that the units of information is in bits.

Information theory is concerned with how the input or transmission entropy changes while it travels through the channel. If the channel is noiseless then the amount of information in a transmission should be unchanged. If there is noise in the channel, then the fidelity of the signal is degraded and the information sent is diminished. If the channel noise is so great and all encompassing, then there is no more surprise in seeing any symbol over another. This is mathematically modeled by the *equivocation* or *conditional entropy*  $H(X|Y)$ , where  $X$  is the random variable representing the channel input and  $Y$  is the random variable representing the channel output. The uncertainty associated with  $X$ , given that  $Y = y_j$  is given by

$$H(X|Y = y_j) = - \sum_{i=0}^{K-1} P(x_i|y_j) \log P(x_i|y_j)$$

Conditional entropy can therefore be defined as:

$$\begin{aligned} H(X|Y) &= \sum_j P(y_j) H(X|Y = y_j) \\ &= - \sum_i \sum_j P(x_i, y_j) \log P(x_i|y_j) \end{aligned}$$

Shannon defined information as follows: the (average) *mutual information* shared between random variables  $X$  and  $Y$  is

$$I(X;Y) = H(X) - H(X|Y)$$

i.e., the information  $Y$  reveals about  $X$  is the prior uncertainty in  $X$ , less the posterior uncertainty about  $X$  after  $Y$  is specified. From this definition, we have

$$\begin{aligned} I(X;Y) &= -\sum_i P(x_i) \log P(x_i) + \\ &\quad \sum_i \sum_j P(x_i, y_j) \log P(x_i|y_j) \\ &= -\sum_i \sum_j P(x_i, y_j) \log P(x_i) + \\ &\quad \sum_i \sum_j P(x_i, y_j) \log P(x_i|y_j) \\ &= \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i|y_j)}{P(x_i)} \end{aligned}$$

Using the definition of conditional probability,

$$I(X;Y) = \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)}$$

When transmitting, the transmitter can do nothing about the noise, and the receiver is passive and waits for symbols to be passed over the channel. However, the transmitter can send different symbols with different frequencies; thus there are different distributions for  $X$ . By changing the frequency of the symbols sent, the transmitter can affect the amount of information sent to the receiver.  $C_{\max}$  is the maximum amount of information, in units of bits per channel usage, that can be sent:

$$C_{\max} = \max(I(X;Y))$$

where the maximum is taken over the different distributions on  $X$ .

There is a critical difference between covert channels and communication channels, though. The goal of a communication channel designer is to maximize capacity and minimize the influence of noise. When covert channels exist, the goal of the system designer is exactly the opposite — to try to minimize the capacity, usually by increasing noise. Although the terminology is not precise, in this paper — as in most of the literature on covert channels — the terms *mutual information* and *capacity* will be used interchangeably.

## 2.2 Correctness Criteria for Secure Schedulers

Covert channel analysis and removal is the single most important issue in multilevel secure concurrency control. The notion of *non-interference* has been proposed [4] as a simple and intuitively satisfying definition of what it means for a system to be secure. The property of *non-interference* states that the output as seen by a subject must be unaffected by the inputs of another subject at a higher access class. This means that a subject at a lower access class should not be able to distinguish between the outputs from the system in response to an input sequence including actions from a higher level subject and an input sequence in which all inputs at a higher access class have been removed [6].

An extensive analysis of the possible covert channels in a secure concurrency control mechanism and the necessary and sufficient conditions for a secure, interference-free scheduler are given in [6]. Three of these properties are of relevance to the secure two phase locking protocol discussed in this paper. For the following definitions, given a schedule  $s$  and an access level  $l$ ,  $purge(s, l)$  is the schedule with all actions at a level  $> l$  removed from  $s$ .

1) *Value Security*: A scheduler satisfies this property if values read by a subject are not affected by actions with higher subject classification levels. Stated formally, for an input schedule  $p$ , the output schedule  $s$  is said to be *value secure* if  $purge(s, l)$  is view equivalent to the output schedule produced for  $purge(p, l)$ .

2) *Delay Security*: This property ensures that the delay experienced by an action is not affected by the actions of a subject at a higher classification level. For an input schedule  $p$  and an output schedule  $s$ , a scheduler is *delay secure* if for all levels  $l$  in  $p$ , each of the actions  $a_1$  in  $purge(p, l)$  is delayed in the output schedule produced for  $purge(p, l)$  if and only if it is delayed in  $purge(s, l)$ .

3) *Recovery Security*: A set of transactions is in a deadlock state when every transaction in the set is waiting for an event that can only be caused by another transaction in the set (such as release of a lock). Deadlock is a problem unique to locking protocols and is not an issue in timestamp schedulers and optimistic concurrency control. Even these schedulers, however, can reach a state from which they cannot continue without aborting one or more transactions. For simplicity, these two conditions are lumped together and called as deadlock [6].

When a deadlock is detected, some of the actions in the schedule must be aborted, allowing the others to proceed. If resolving the deadlock can allow a high-level transaction to modify the values read by a low level transaction or to affect the delay a low level transaction experiences, a covert channel can arise. When

a deadlock occurs, other channels are available for signaling in addition to those protected by *value security* and *delay security*. The following condition takes care of these channels:

A scheduler is *recovery secure* for all schedules  $p$  if, on the arrival of an action  $A_X$  for scheduling:

- 1) If a deadlock occurs, resulting in a set of actions  $D$  being rolled back, then for all subject classification levels  $l$  in  $p$ , which dominate one of those in  $D$ , a deadlock also occurs in response to the schedule  $purge(p, l)$  on the arrival of the action  $A_X$ , with the actions  $purge(D, l)$  being rolled back.
- 2) If no deadlock occurs on the arrival of  $A_X$ , then for all subject classification levels  $l$  in  $p$ , it does not occur on the arrival of  $A_X$  in the input schedule  $purge(p, l)$ .

*Recovery security* ensures that the occurrence of a deadlock appears the same to a low-level subject, independent of whether higher level actions are in the schedule or not. The actions taken to recover from deadlock are also not affected by the presence of higher level transactions.

### 3 Security and Real Time Requirements

The property of non-interference has the unfortunate effect of degrading performance for transactions at a higher access class. In a secure environment, a transaction at a higher access class:

- cannot cause a transaction at a lower access class to abort. If it is allowed to do so, it is possible that it can control the number of times a lower level transaction is aborted, thereby opening a covert channel.
- cannot conflict with a transaction at a lower access class. If such a conflict does occur, the higher level transaction has to be blocked or aborted, not the low level transaction.
- cannot be granted greater priority of execution over a transaction at a lower access class.

There have been a number of papers in the real-time databases literature that have explored priority based scheduling approaches with respect to conventional databases [1],[12], [14]. The problem arises when these approaches are applied to secure databases, because covert channels can be introduced by priority based scheduling. All existing real-time systems schedule transactions based on some priority scheme. The priority usually reflects how close the transaction is to missing its deadline. Priority-based scheduling of real-time transactions, however, interacts with the property of non-interference which has to be satisfied by secure schedulers [6]. Take the sequence of transactions input to a scheduler as shown (the transactions arrived in the  $T_1, T_2, T_3, T_4$  order):

$T_1$ (SECRET)	:	$r(x)$	
$T_2$ (UNCLASSIFIED)	:	$w(x)$	
$T_3$ (UNCLASSIFIED)	:		$w(x)$
$T_4$ (UNCLASSIFIED)	:		$r(x)$

Assume that  $T_1, T_2, T_3$  and  $T_4$  have priorities 5, 7, 10 and 12 respectively and the priority assignment scheme is such that if  $priority(T_2) > priority(T_1)$ , then  $T_2$  is more critical and has to be scheduled ahead of  $T_1$ . In the above example,  $T_2$  and  $T_3$  are initially blocked by  $T_1$  when they arrive. When  $T_1$  completes execution,  $T_3$  is scheduled ahead of  $T_2$ , since it has a greater priority than  $T_2$  and the transaction execution order would be  $T_1 T_3 T_2 T_4$ . However, if the transaction  $T_1$  is removed, the execution order would be  $T_2 T_3 T_4$  because  $T_2$  would have been scheduled as soon as it had arrived. The presence of the SECRET transaction  $T_1$  thus changes the value read by the UNCLASSIFIED transaction  $T_4$ , which is a violation of *value security*. Delay security is also violated, since the presence of  $T_1$  delays both  $T_2$  and  $T_3$ .

Therefore, to satisfy the correctness properties discussed in Section 2.2 (to close all covert channels), we see that a very high performance penalty would be paid. In our approach to improving performance, we shall discuss a method to trade-off capacity of a covert channel with performance (measured in terms of deadline miss percentage).

### 4 Secure Two Phase Locking

Before a discussion and analysis of covert channels, let us study two concurrency control approaches at different ends of the spectrum - Secure 2PL, a fully secure protocol which does not consider transaction priorities while scheduling and 2PL-HP, which has some deadline cognizance built into it, but is not free from covert channels.

#### 4.1 Secure 2PL

Basic two-phase locking does not work for secure databases because a transaction at a lower access class (say  $T_l$ ) cannot be blocked due to a conflicting lock held by a transaction at a higher access class ( $T_h$ ). If  $T_l$  were somehow allowed to continue with its execution in spite of the conflict, then non-interference would be satisfied. The basic principle behind the secure two-phase locking protocol is to try to simulate execution of Basic 2PL without blocking the lower access class transactions by higher access class transactions.

Consider the two transactions in example 1:

$T_1$ (SECRET)	:	$r[x]$	$\dots$	$c_1$
$T_2$ (UNCLASSIFIED)	:	$w[x]$		$c_2$

#### EXAMPLE 1

Basic two phase locking would fail because  $w_2[x]$  would be blocked waiting for  $T_1$  to commit and release read-lock on  $x$  ( $ru_1[x]$ ). In our modification to the two phase locking protocol,  $T_2$  is allowed to set a virtual lock  $wvl_2[x]$ , write onto a version of  $x$  local to  $T_2$  and continue with the execution of its next operation, i.e.  $c_2$ . When  $T_1$  commits and releases the lock on  $x$ ,  $T_2$ 's virtual write lock is upgraded to a real lock and  $w_2[x]$  is performed. Until  $w_2[x]$  is performed, no conflicting

action is allowed to set a lock on  $x$ . The sequence of operations performed are therefore,  $rl_1[x] r_1[x] vwl_2[x] vw_2[x] c_2 \dots c_1 ru_1[x] wl_2[x] w_2[x] wu_2[x]$ .

This modification alone is not enough, as illustrated in Example 2:

$T_1$ (SECRET)	:	$r[x]$	$r[y]$	$c_1$
$T_2$ (UNCLASSIFIED)	:	$w[x]$	$w[y]$	$c_2$

#### EXAMPLE 2

The sequence of operations that would be performed are  $rl_1[x] r_1[x] vwl_2[x] vw_2[x] wl_2[y] w_2[y] c_2$ . After these operations, deadlock would occur because  $r_1[y]$  waits for  $w_2[y]$  to release its virtual lock and  $vw_2[x]$  waits for  $r_1[x]$  to release its lock. This deadlock would not have occurred in basic two phase locking. Note that our aim of trying to simulate execution of basic two phase locking is not being achieved. On closer inspection, it is obvious that this problem arises because  $w_2[y]$  is allowed to proceed with its execution even though  $w_2[x]$  could only write onto a local version of  $x$  due to the read lock  $rl_1[x]$  set by  $T_1$ . To avoid this problem, for each transaction  $T_i$ , two lists are maintained — *before*( $T_i$ ) which is the list of active transactions that precede  $T_i$  in the serialization order and *after*( $T_i$ ) which is the list of active transactions that follow  $T_i$  in the serialization order. This idea is adapted from [14], where *before\_cnt* and *after\_cnt* are used to dynamically adjust the serialization order of transactions. The following additions are made to the basic two phase locking protocol:

- 1) When an action  $p_i[x]$  sets a virtual lock on  $x$  because of a real lock  $ql_j[x]$  held by  $T_j$ , then  $T_i$  and all transactions in *after*( $T_i$ ) are added to *after*( $T_j$ ), and  $T_j$  and all transactions in *before*( $T_j$ ) are added to *before*( $T_i$ ).
- 2) When an action  $w_i[x]$  arrives and finds that a previous action  $w_i[y]$  (for some data item  $y$ ) has already set a virtual write lock  $vwl_i[y]$ , then a dependent lock  $dvwl_i[x]$  is set with respect to  $vwl_i[y]$ .
- 3) When an action  $p_i[x]$  arrives and finds that a conflicting virtual or dependent lock  $vql_j[x]$  or  $dvql_j[x]$  has been set by a transaction  $T_j$  which is in *after*( $T_i$ ), then  $p_i[x]$  is allowed to set a lock on  $x$  and perform  $p_i[x]$  in spite of the conflicting lock.
- 4) A dependent virtual lock  $dvpi[x]$ , dependent on some action  $qi[y]$  is upgraded to a virtual lock when  $vql_i[x]$  is upgraded to a real lock.

The maintenance of a serialization order and the presence of dependent locks are necessary to prevent uncontrolled acquisition of virtual locks by transactions at lower access classes.

For example 2, the sequence of operations that would now be performed are  $rl_1[x] r_1[x] vwl_2[x] vw_2[x] dvwl_2[y] vw_2[y] c_2 rl_1[y] r_1[y] c_1 ru_1[x] ru_1[y] wl_2[x] w_2[x] wu_2[x] wl_2[y] w_2[y] wu_2[y]$ .

## 4.2 2PL - High Priority

In 2PL-HP [1], all data conflicts are resolved in favor of the transaction with higher priority. When a transaction requests a lock on an object held by other transactions in a conflicting mode, if the requester's priority is higher than that of all lock holders, the holders are restarted and the requester is granted the lock; if the requester's priority is lower, it waits for the lock holders to release the lock. In addition, a new read lock requester can join a group of read lock holders only if its priority is higher than that of all waiting write lock operations.

A real-time secure concurrency control must possess two characteristics - speed and minimal deadline miss percentage. The secure two phase locking protocol [3] was shown to yield best average case performance among all the secure concurrency control approaches whose performance was evaluated in [16]. We therefore use it as a basis for our solution to the problem of real-time secure concurrency control. From our discussion earlier in this paper, it is clear that priority based transaction scheduling is not feasible for a fully secure database system. Therefore, for minimizing deadline miss percentage, we take the approach that partial security violations under certain conditions are permissible, if it results in substantial gain in time cognizance.

## 5 Covert Channel Analysis

### 5.1 A Noisy Covert Channel

In any system where a locking mechanism is used for synchronization of concurrently executing transactions, whenever a transaction  $T_1$  requests a lock on a data item  $x$  on which another transaction  $T_2$  holds a conflicting lock, there are two possible options:

- $T_1$  could be blocked until  $T_2$  releases the lock.
- $T_2$  could be aborted and the lock granted to  $T_1$ .

The latter option is a "non-secure" option that is taken by 2PL-HP when  $T_1$  has a higher priority than  $T_2$ . If  $T_1$  were at a higher security level than  $T_2$ , this option would be a violation of security. The former option, along with the additional conditions and actions described in Section 4.1, would be the "secure" option if  $T_1$  were at a lower security level than  $T_2$ . However, this option does not take into account the priorities of  $T_1$  and  $T_2$ . In our approach, we try to strike a balance between these two options. Consider a Bernoulli random variable  $X$  with parameter  $q_2$ . Now, whenever a conflict arises between a lock holding transaction ( $T_2$ ) and a lock requesting transaction ( $T_1$ ), such that  $priority(T_1) > priority(T_2)$ ,  $T_2$  is aborted if  $X = 1$ . Since  $P(X = 1) = q_2$ ,  $T_2$  is aborted with a probability  $q_2$  (the "non-secure" option is taken). If  $X = 0$ , then the "secure" option is taken. Note that  $q_2$  can be used to control the extent to which security is satisfied. Lesser the value of  $q_2$ , greater the extent to which security is satisfied and therefore greater the miss percentage.

Unfortunately, this approach is not free from covert channels. Consider two collaborating transactions,

one at security level LOW and the other at security level HIGH, each consisting of just one operation. Assume that at the start of a time interval of duration  $t$  (henceforth referred to as a tick), the LOW transaction submits a write on a data item  $x$  and shortly thereafter (within the tick), the HIGH transaction submits a read on  $x$ . Also assume that the transactions collaborate to ensure that the HIGH transaction has an earlier deadline than the LOW transaction. Now, in the absence of other transactions and if  $q_2$  were 1, then the LOW transaction would certainly be aborted due to the HIGH transaction. If the HIGH transaction were not submitted, then the LOW transaction would commit. Therefore it takes just one tick for the HIGH transaction to transmit either a “1” (by submitting its operation) or a “0” (by not submitting its operation). In this case, the capacity of the channel is 1 bit/tick. There are, however, two factors which introduce noise into this channel — firstly, the presence of other transactions and secondly, the probability  $q_2$  of the lock holding transaction being aborted. The first factor is modeled by a probability  $q_1$ , where  $q_1$  is the probability that a transaction  $T_3$  with an earlier deadline than the LOW transaction submits a read or a write on  $x$  before the HIGH transaction, i.e., the aborting of the LOW transaction is caused by  $T_3$ . In the next sub-section, we shall derive an equation for capacity in terms of these two factors.

An important assumption has to be stated at this point regarding the extent of knowledge that a HIGH user has. We assume that a HIGH user has information only about the transactions that it and its collaborators submit, i.e., all system-maintained information such as current arrival rate of transactions, the deadlines of other transactions in the system, locks held by other transactions, etc., is at a SUPER-HIGH level and inaccessible to HIGH users. This assumption is not unfair because the concurrency control manager is trusted and therefore, should not leak out information that could be used by a malicious user. This assumption is important because if a malicious HIGH user has access to system information, it has control over  $q_1$ . If it knows which transactions could possibly interfere with its transmission of a “1” to the LOW user, it can then get rid of those transactions as follows: At the start of a tick, the HIGH user first finds a set of active transactions that have an earlier deadline than the collaborating LOW level transaction and a data item on which each transaction holds a lock. This can be represented as a set of tuples  $\{[T_1, x_1], [T_2, x_2], \dots, [T_n, x_n]\}$ . It then submits transactions with a lesser deadline that access each of these data items, thereby causing the abortion of all the transactions in the set. This does not eliminate the effect of  $q_1$  on the channel, but reduces its value.

## 5.2 Capacity Analysis

Let  $X$  be the random variable representing the input to the covert channel from the HIGH transaction (in this paper, we shall refer to it as the transmitter) and  $Y$  the output random variable corresponding to the LOW transaction (referred to as the receiver). Let

$0_x$  - event that the transmitter sends a 0 (HIGH transaction not submitted).

$1_x$  - event that the transmitter sends a 1 (HIGH transaction submitted).

$0_y$  - receiver receives a 0 (LOW transaction committed).

$1_y$  - receiver receives a 1 (LOW transaction aborted).

Let  $P(i_y|j_x)$  be the probability of an  $i$  being received given that a  $j$  was sent and  $P(i_x, j_y)$  the joint probability that an  $i$  is sent and a  $j$  was received.

Assume  $P(0_x) = p$  and  $P(1_x) = 1 - p$ .

We have:

$$\begin{aligned} P(0_y|0_x) &= 1 - q_1q_2 \\ P(1_y|0_x) &= q_1q_2 \\ P(0_y|1_x) &= 1 - q_2 \\ P(1_y|1_x) &= q_2 \\ P(0_x, 0_y) &= p(1 - q_1q_2) \\ P(0_x, 1_y) &= pq_1q_2 \\ P(1_x, 0_y) &= (1 - p)(1 - q_2) \\ P(1_x, 1_y) &= (1 - p)q_2 \\ P(0_y) &= p(1 - q_1q_2) + (1 - p)(1 - q_2) \\ P(1_y) &= pq_1q_2 + (1 - p)q_2 \end{aligned}$$

The mutual information of a channel (from Section 2.1) is given by:

$$I(X; Y) = \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)}$$

Substituting the derived probabilities and simplifying, the capacity can be derived as:

$$\begin{aligned} I(X; Y) &= p(1 - q_1q_2) \log \frac{1 - q_1q_2}{\alpha} + \\ & pq_1q_2 \log \frac{q_1q_2}{\beta} + \\ & (1 - p)(1 - q_2) \log \frac{1 - q_2}{\alpha} + \\ & (1 - p)q_2 \log \frac{q_2}{\beta} \end{aligned}$$

where

$$\begin{aligned} \alpha &= p(1 - q_1q_2) + (1 - p)(1 - q_2) \\ \beta &= pq_1q_2 + (1 - p)q_2 \end{aligned}$$

When  $q_2 \rightarrow 0$  (i.e., the current holder is never aborted), each of the four terms in  $I(X; Y)$  tend to

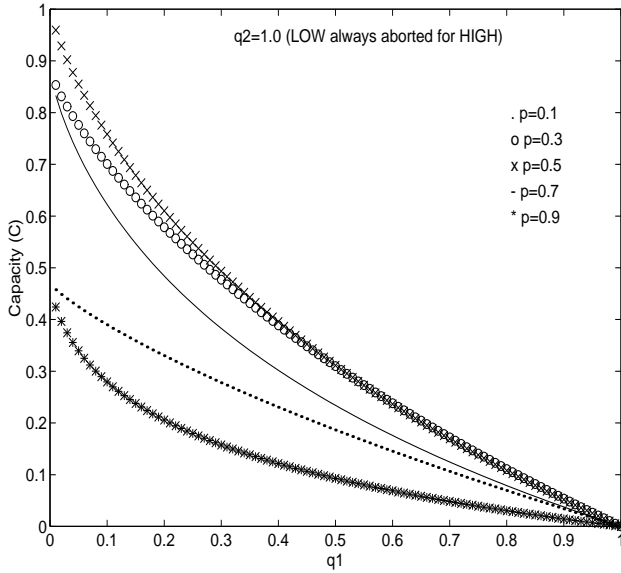


Figure 1: Capacity vs.  $q_1$

zero, and hence the capacity is zero. This result bears out the fact that Secure 2PL is free from covert channels, because lower access class transactions are never aborted by higher access class transactions.

When  $q_2 = 1$ , the capacity of the covert channel is dependent on  $p$  and  $q_1$  (the noise in the system). The capacity is given by:

$$C = pq_1 \log q_1 - p(1 - q_1) \log p - (1 - p + pq_1) \log(1 - p + pq_1)$$

A plot of  $C$  vs.  $q_1$  for different values of  $p$  with  $q_2 = 1$  (i.e., LOW is always aborted when HIGH arrives) is shown in Figure 1. For low values of  $q_1$ , the capacity of the channel is close to 1, since a low value of  $q_1$  signifies low interference (or “noise”) in the transmission of a “1” from the transmitter to the receiver. However, as  $q_1$  increases, “noise” increases and therefore capacity decreases, approaching zero as  $q_1$  approaches 1. It is also interesting to note that the capacity is highly sensitive to the value of  $p$ , with high reaching when  $p = 0.5$ .

When  $q_1 = 0$  (i.e., no interference from other transactions), the capacity of the covert channel is given by:

$$C = (1 - p)(1 - q_2) \log(1 - q_2) - (1 - p)q_2 \log(1 - p) - (1 - q_2 + pq_2) \log(1 - q_2 + pq_2)$$

A plot of  $C$  vs.  $q_2$  for different values of  $p$  with  $q_1 = 0$  (i.e., no interference from other transactions) is shown in Figure 2. For high values of  $q_2$ , the behavior of the scheme approaches that of 2PL-HP, resulting in a maximum capacity of 1. For low values of  $q_2$ , however, the behavior approaches that of Secure 2PL and the capacity is low. For  $q_2 = 0$ , the scheme is

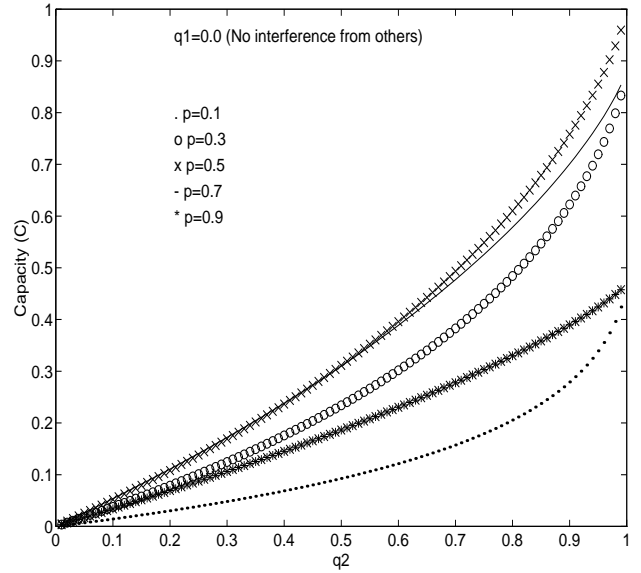


Figure 2: Capacity vs.  $q_2$

identical to Secure 2PL and the capacity is 0. As before, the capacity is sensitive to the value of  $p$ , with high values at  $p = 0.5$ .

A plot of  $C$  vs. both  $q_1$  and  $q_2$  when  $p = 0.5$  is shown in Figure 3. It shows that for any given value of  $q_1$ , capacity increases with increasing  $q_2$ . For any given  $q_2$ , however, capacity decreases with increasing  $q_1$ . In addition, capacity seems to be highest when either  $q_1 = 0$  or  $q_2 = 1$ . The results support our intuitive understanding of the capacity.

## 6 A Secure Real-Time Concurrency Control Mechanism

To bound the capacity of a covert channel, there are two parameters that can be tuned:  $q_1$  and  $q_2$ .  $q_1$  can be increased by introducing “fake” transactions which do not change the state of the database, but which access data items randomly. This is not a desirable option, since these transactions compete for resources and data items that would otherwise be allocated to normal transactions, thereby degrading the performance. Therefore, we shall assume that  $q_1$  is a parameter that cannot be controlled and it is  $q_2$  that has to be tuned. The value of  $q_1$  is available to the system at any given time and given the value of  $q_1$  and the desired capacity, the system can adjust the value of  $q_2$ .  $q_1$  depends on two factors — the arrival rate of transactions and the duration of a tick (the period of time to transmit a bit covertly to the low-level user). The two transactions involved in the covert channel can collaborate to reduce the duration of a tick, thereby reducing  $q_1$ . However, there is a certain lower bound, below which the duration cannot be reduced. This is because there are three steps involved in the transmission of a bit:

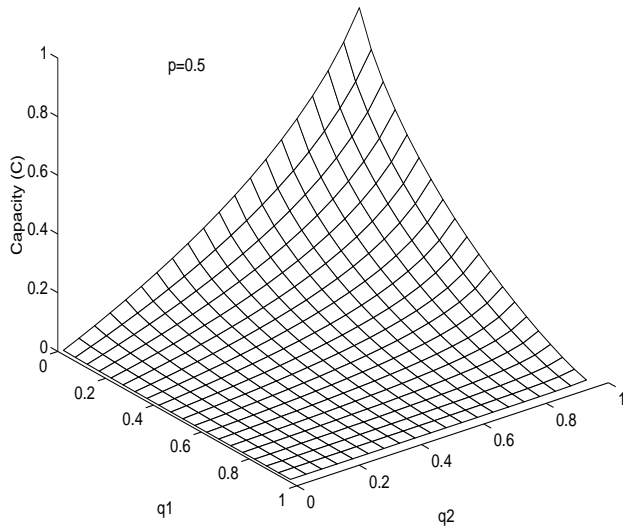


Figure 3: Capacity vs.  $q_1$  and  $q_2$

- at the start of a tick, the LOW transaction submits its write operation.
- if the HIGH user wishes to transmit a “1”, it submits its read operation.
- the system has to send a “TRANSACTION ABORTED” message to the LOW user.
- at the start of a tick, the LOW transaction submits its write operation.
- if the HIGH user wishes to transmit a “1”, it submits its read operation.
- the system has to send a “TRANSACTION ABORTED” message to the LOW user.

For the covert channel to be effective, the duration of a tick cannot be lower than the overhead involved in performing these three operations in the worst case. The probability  $q_1$  depends on both the arrival rate of transactions as well as the duration of a tick. For a fairly accurate calculation of  $q_1$ , a queueing model of the system is required and is beyond the scope of this paper.

There are two requirements on a secure real-time concurrency control mechanism — a security requirement, expressed as an upper bound on capacity and a real-time requirement, expressed as an upper bound on miss percentage. Given a capacity and a value for  $q_1$ ,  $q_2$  can be calculated from the equation derived for capacity in the previous section. It is very difficult to derive a closed form solution for  $q_2$  in terms of  $q_1$  and  $C$ , but a simple iterative solution for  $q_2$  can be obtained easily using the Newton-Raphson method. There is no direct mathematical relationship between

the deadline miss percentage and  $q_1$  and  $q_2$ . However, the performance results discussed in [16] indicate that with increasing arrival rate (and therefore  $q_1$ ), the deadline miss percentage increases slowly but steadily up to a certain point, after which the system becomes unstable. Similarly, with increasing  $q_2$  (from 0 to 1), the deadline miss percentage first increases (up to a value of  $q_2 = 0.3 - 0.4$ ) and then decreases continuously until  $q_2 = 1$ .

Our approach to a real-time secure concurrency control mechanism uses a feedback control mechanism to ensure that the capacity at any given time does not exceed the upper bound specified. The approach is described by the following pseudo-code:

- 1) *Input desired deadline miss percentage (DDMP) and capacity (C)*
- 2) *Calculate  $q_2$ , given C and current  $q_1$*
- 3) *Observe resulting deadline miss percentage (DMP)*
- 4) *If (DMP  $\geq$  DDMP)  
report back to database administrator (DBA);  
DBA readjusts DDMP and(or) C;  
go to step 2;*
- 5) *Else If ((DDMP - DMP) > THRESHOLD)  
decrease  $q_2$  to 0; /\* reduce capacity to 0 \*/  
go to step 3;*
- 6) *Else  
/\*(DDMP - DMP)  $\leq$  THRESHOLD \*/  
go to step 2;*

This approach provides guarantees only on the capacity of the resulting channel, not on the deadline miss percentage. If the miss percentage increases above the desired miss percentage specified, there is nothing that the system can do. The only thing that can be done is to raise an ERROR condition as in step 4. The DBA can then either increase the upper bound on capacity, thereby increasing  $q_2$  and in turn decreasing the miss percentage, or can relax the miss percentage requirement and increase the value of the desired miss percentage.

If the deadline miss percentage requirement is being comfortably met by the system, then a drop in miss percentage can be afforded. This is what is done in step 5, where the covert channel is effectively closed by setting  $q_2$  to 0. When the miss percentage again increases and approaches the desired miss percentage value, normal operation is resumed and the value of  $q_2$  is calculated from  $C$  and the current value of  $q_1$ .

## 7 Discussion

The capacity of a covert channel varies inversely with the degree of randomness in the system. In the scheme that we have discussed, there is not much randomness, since we strive to maintain the capacity at a specified value. One can therefore argue that since the capacity is maintained more or less constant, a malicious subject can utilize this channel — albeit at

a much lower fidelity — to still transmit information. A certain degree of randomness can be introduced by the following procedure: the value of  $q_2$  is calculated from the desired value of  $C$  and the current value of  $q_1$ . Instead of using the value of  $q_2$  thus calculated, the value of  $q_2$  is sampled from a uniform distribution between  $[q_2 - \delta, q_2 + \delta]$ . The greater the value of  $\delta$ , the greater the uncertainty in the resulting value of  $C$ . This might mean that sometimes the capacity might increase beyond the upper bound specified, but due to the uncertainty it is very difficult for a user to exploit this channel.

All the capacity derivations/methods to control capacity explained in this paper have been for the type of the covert channel discussed in Section 5.1. Are there other covert channels that malicious users can exploit and whose capacity would not be controlled by the feedback monitoring method explained earlier in the previous section? Let us investigate this issue further. From the correctness criteria for secure schedulers, covert channels can be broadly classified into three categories — those that communicate information through a violation of *delay security*, those that violate *recovery security* and those that violate *value security*. In [3], it is proved that Secure 2PL satisfies *delay security*. Our real-time secure concurrency control mechanism explained in Section 6 is based on the Secure 2PL protocol. The approach differs from Secure 2PL only when there is a conflict between a lock holding transaction  $T_1$  and a lock requesting transaction  $T_2$  and  $priority(T_2) > priority(T_1)$ . In this case,  $T_1$  is aborted and  $T_2$  granted the lock, i.e., no transaction is being blocked. Therefore, *delay security* is not violated at any point. The covert channel studied in Section 5.1 is a canonical example of a channel that exploits a violation of *recovery security*. There might be other, more complicated channels that could involve more than two transactions, but the parameters on which their capacity would be dependent on would be a superset of  $q_1$  and  $q_2$ . A covert channel involving four collaborating transactions — one at HIGH and the rest at LOW — that exploits a violation in value security can work as follows:

- At the start of a tick, a LOW transaction  $T_1$  submits a write on a data item  $x$  ( $w_1[x]$ ).
- A second LOW transaction  $T_2$  then submits a write on  $x$  ( $w_2[x]$ ).
- If the HIGH transaction  $T_3$  wants to transmit a “1,” it submits a read on  $x$ , such that  $deadline(T_1) < deadline(T_3) < deadline(T_2)$ . As a result,  $T_2$  is aborted.
- The “receiving” LOW transaction  $T_4$  then submits a read on  $x$ . If it reads the value written by  $T_1$ , then a “1” is received and if it reads the value written by  $T_2$ , a “0” is received.

This covert channel too is dependent on two factors — the probability  $q_1$  that a transaction  $T_4$  would cause the aborting of  $T_2$  before  $T_3$  arrives and a probability  $q_2$  that  $T_2$  would actually be aborted when  $T_3$  submits

its operation. In addition, there is also the possibility that  $T_1$  could be aborted before  $T_4$  submits its read, introducing an additional “noise” factor. As a result, the capacity of this channel would actually be less than that of the simple channel studied in Section 5.1.

Summarizing, we find that simpler the covert channel, the lesser the number of factors that the capacity of the channel is dependent on and therefore greater its capacity. The covert channel studied in Section 5.1 is the simplest possible channel that can be exploited, given the correctness properties that are violated and therefore bounding its capacity is enough to bound the capacity of more complicated covert channels that could be exploited.

## 8 Performance Evaluation

In this section, we present the results of our performance study of the feedback control mechanism for a range of transaction arrival rates. The goal of the analysis is to show the variation in miss percentage for varying capacity of the covert channel.

### 8.1 Simulation Model

Central to the simulation model is a single-site disk resident database system operating on shared-memory multiprocessors [8]. The system consists of a disk-based database and a main memory cache. The unit of database granularity is the page. When a transaction needs to perform an operation on a data item it accesses a page. If the page is not found in the cache, it is read from disk. CPU or disk access is through an M/M/k queueing system, consisting of a single queue with  $k$  servers (where  $k$  is the number of disks or CPUs). The amounts of CPU and disk I/O times are specified as model parameters in Table 1. Since we are concerned only with providing security at the concurrency control level, the issue of providing security at the operating system or resource scheduling layer is not considered in this paper. That is the reason why we do not consider a secure CPU/disk scheduling approach. Our assumption is that the lower layers provide the higher concurrency control layer with a fair resource scheduling policy.

The feedback approach is implemented as a layer over Secure 2PL. In the model, the execution of a transaction consists of multiple instances of alternating data access requests and data operation steps, until all the data operations in it complete or it is aborted. When a transaction makes a data request, i.e., lock request on a data object, the request must go through concurrency control to obtain a lock on the data object. If the transaction’s priority is greater than all of the lock holders, and its lock request conflicts with that of the holders, then the holders are aborted and the transaction is granted a lock with a probability  $q_2$  else the steps taken by the Secure 2PL protocol are followed; if the transaction’s priority is lower, it waits for the lock holders to release the lock [1]. The probability  $q_2$  depends on the factors capacity ( $C$ ) and  $q_1$ .  $C$  is available directly, but  $q_1$  is calculated based on the arrival rate of transactions, the probability of contention, and their deadlines. The analysis is based on preemptive priority queueing policy with restart. (The details are omitted for brevity.)

Parameter	Base Value
<i>DBSize</i>	350
<i>NumCPUs</i>	2
<i>NumDisks</i>	4
<i>CPUTime</i>	15 msec
<i>DiskTime</i>	25 msec
<i>BufProb</i>	0.5
<i>NumSecLevels</i>	6

Table 1: System Resource Parameters

If the request for a lock is granted, the transaction proceeds to perform the data operation, which consists of a possible disk access (if the data item is not present in the cache) followed by CPU computation. However, if only a virtual or dependent lock is granted, the transaction only does CPU computation, since the operation should only be performed on a local version. If the request for the lock is denied (the transaction is blocked), the transaction is placed into the data queue. When the waiting transaction is granted a lock, only then can it perform its data operation. Also, when a virtual lock for an operation is upgraded to a real lock, the data operation requires disk access and CPU computation. At any stage, if a deadlock is detected, the transaction to be aborted to break the deadlock is determined, aborted and restarted. When all the operations in a transaction are completed, the transaction commits. Even if a transaction misses its deadline, it is allowed to execute until all its actions are completed.

## 8.2 Parameters and Performance Metrics

Table 1 gives the names and meanings of the parameters that control system resources. The parameters, *CPUTime* and *DiskTime* capture the CPU and disk processing times per data page. Our simulation system does not explicitly account for the time needed for data operation scheduling. We assume that these costs are included in *CPUTime* on a per data object basis. The use of a database cache is simulated using probability. When a transaction attempts to read a data page, the system determines whether the page is in cache or disk using the probability *BufProb*. If the page is determined to be in cache, the transaction can continue processing without disk access. Otherwise disk access is needed.

Table 2 summarizes the key parameters that characterize system workload and transactions. Transactions arrive in a Poisson stream, i.e., their inter-arrival rates are exponentially distributed. The *ArriRate* parameter specifies the mean rate of transaction arrivals. The number of data objects accessed by a transaction is determined by a normal distribution with mean *TranSize*, and the actual data objects to be accessed are determined uniformly from the database.

The assignment of deadlines to transactions is controlled by the parameters *MinSlack* and *MaxSlack*, which set a lower and upper bound, respectively, on a transaction’s slack time. We use the formula for

Parameter	Base Value
<i>ArriRate</i>	-
<i>TranSize</i>	6
<i>RestartDelay</i>	1 msec
<i>MinSlack</i>	2
<i>MaxSlack</i>	8

Table 2: Workload Parameters

deadline-assignment to a transaction.

$$Deadline = AT + \text{Uniform}(MinSlack, MaxSlack) * ET$$

*AT* and *ET* denote the arrival time and execution time, respectively. The execution time of a transaction used in this formula is not an actual execution time, but a time estimated using the values of parameters *TranSize*, *CPUTime* and *DiskTime*. The priorities of transactions are decided by the *Earliest Deadline First policy*.

The performance metric used is miss percentage, which is the ratio of the number of transactions that do not meet their deadline to the total number of transactions committed.

## 8.3 Experimental Results

An event-based simulation framework was written in ‘C’. For each experiment, we ran the simulation with the same parameters for 6 different random number seeds. Each simulation run was continued until 200 transactions at each access class were committed. For each run, the statistics gathered during the first few seconds were discarded in order to let the system stabilize after an initial transient condition. For each experiment the required performance measure was measured over a wide range of workload. All the data reported in this paper have 90% confidence intervals, whose endpoints are within 10% of the point estimate.

In the experiment, the miss percentages for the feedback approach are measured for two different arrival rates. The resulting graph is shown in Figure 4. Since we are considering a real-time database system, we restrict attention to the portion of the graph where miss percentages are less than 10%. The performance after the saturation point is not an issue. At low arrival rates, the dependence of miss percentage on the capacity is minimal. This is because very few transactions miss their deadline even at low capacities and increasing the capacity does not appreciably decrease the number of transactions that miss their deadline. Note that one does not see a progressive decrease in miss percentage from  $C = 0$  to  $C = 1$ . The miss percentage increases from  $C = 0$  to  $C = 0.16$ , but subsequently decreases up to  $C = 1$ . This is because at lower values of  $C$ , the number of actions scheduled according to priority is very low (since  $q_2$  is low). However, since the system is not fully secure, some transactions are being aborted, when security needs

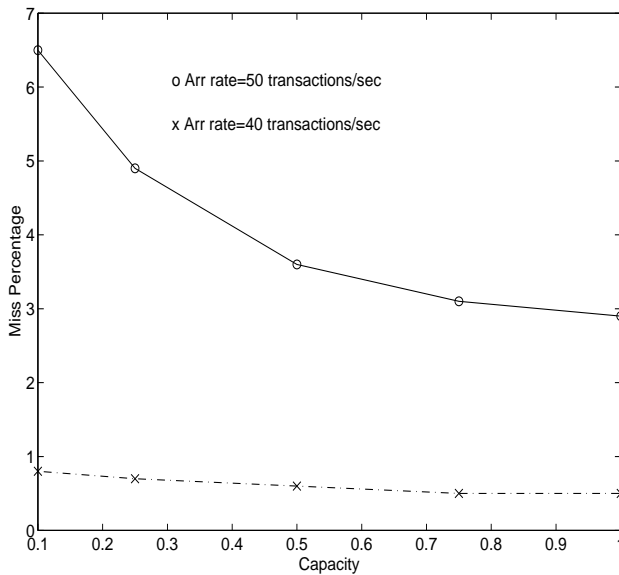


Figure 4: Miss percentage vs. Capacity (C)

to be violated. The problem is that the former factor does not offset the latter factor, resulting in increased miss percentage. As  $C$  increases, the former factor increases, resulting in more transactions meeting their deadline.

## 9 Conclusion

In this paper, we have explored a possible direction for research in scheduling transactions to meet their timing constraints in a secure database. A possible way in which security could be partially compromised for improved miss percentage was explained and an expression for the capacity of the resultant covert channel derived. A feedback control system was then developed, which ensured that the capacity of the covert channel did not exceed a desired upper bound. Although no guarantees can be provided by the system on the deadline miss percentage, a facility is provided for renegotiating on the desired deadline miss percentage and the desired capacity when the desired miss percentage is exceeded.

There are a number of issues for future work. In the derivation of the capacity of the covert channel, we have concentrated mainly on the dependence of capacity on parameter  $q_2$ . The dependence of capacity on the presence of other transactions in the system was conveniently abstracted away into a single parameter  $q_1$ . Although an approximate method for the estimation of  $q_1$  was used in the performance analysis, a precise calculation of  $q_1$  has not been considered. A formal queueing model of the system, based on the arrival rate of transactions, a calculation of lock conflict probabilities, blocking time, etc., is important not only for determining  $q_1$ , but could also help in establishing a probabilistic relationship between miss percentage and  $q_1$  and  $q_2$ . This could eliminate the need for raising an ERROR condition when the desired

miss percentage is exceeded, since the correct setting of  $q_2$  can be obtained mathematically from capacity and desired miss percentage. Secondly, in [11] the use of capacity as a measure of security is questioned. Examples of zero capacity channels are provided, where short messages can be sent through without any errors (or loss in fidelity). A small message criterion (SMC) is introduced, which is an indication of what will be tolerated by the system in terms of covertly leaking a short covert message of length  $n$  bits in time  $t$  and with fidelity of transmission  $r\%$ . Further work is needed to design a formal criterion that captures all these factors and has the same mathematical elegance as capacity.

## Acknowledgements

This work was supported in part by NASA LaRC, ONR, VCIT, and Loral Federal Systems.

## References

- [1] Robert K. Abbott and Hector Garcia-Molina. "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 17, No. 3, pp 513-560, September '92.
- [2] D. E. Bell and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretation," The Mitre Corp., March 1976.
- [3] Rasikan David and Sang H. Son. "A Secure Two Phase Locking Protocol," *Proceedings of the 12th Symposium on Reliable Distributed Systems*, Princeton, NJ, Oct '93.
- [4] J. A. Goguen and J. Meseguer. "Security Policy and Security Models," *Proceedings of the IEEE Symposium on Security and Privacy*, pp 11-20, 1982.
- [5] Sushil Jajodia and Vijayalaksmi Atluri. "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Databases," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1992.
- [6] T. F. Keefe, W. T. Tsai and J. Srivastava. "Multilevel Secure Database Concurrency Control," *In Proceedings of the Sixth International Conference on Data Engineering*, pp 337-344, Los Angeles, CA, February 1990.
- [7] Butler W. Lampson. "A Note on the Confinement Problem," *Communications of the ACM*, Vol. 16, No. 10, pp 613-615, October 1973.
- [8] Juhnyoung Lee and Sang H. Son. "Concurrency Control Algorithms for Real-Time Database Systems" in "Performance of Concurrency Control Mechanisms in Centralized Database Systems," Vijay Kumar (editor), Prentice Hall 1995 (to appear).
- [9] Jonathan K. Millen. "Finite-State Noiseless Covert Channels," *The Second Computer Security Foundations Workshop*, pp 81-86, June 1989.

- [10] Ira S. Moskowitz and Allen R. Miller. "The Channel Capacity of a Certain Noisy Timing Channel," *IEEE Transactions on Information Theory*, pp 1339-1344, July 1992.
- [11] Ira S. Moskowitz and Myong H. Kang. "Covert Channels - Here to Stay?," *COMPASS '94*, 1994.
- [12] Sha L., R. Rajkumar and J. P. Lehoczky. "Priority Inheritance Protocol: An Approach to Real-time Synchronization," Technical Report, Computer Science Department, Carnegie-Mellon University, 1987.
- [13] C. E. Shannon and W. Weaver. "The Mathematical Theory of Communication," The University of Illinois Press, Urbana, IL, 1949.
- [14] Sang H. Son, Juhnyoung Lee and Yi Lin. "Hybrid Protocols Using Dynamic Adjustment of Serialization Order for Real-Time Concurrency Control," *Real-Time Systems Journal*, Vol. 4, No. 3, pp 269-276, September 1992.
- [15] Sang H. Son and Bhavani Thuraisingham. "Towards a Multilevel Secure Database Management System for Real-Time Applications," *IEEE Workshop on Real-Time Applications*, New York, New York, May 1993.
- [16] Sang H. Son and R. David, "Design and Analysis of a Secure Two-Phase Locking Protocol," 18th International Computer Software and Applications Conference (COMPSAC'94), Taipei, Taiwan, November 1994, pp 374-379.