Application of Unique View Consistency for Elimination of Covert Channels in Real-Time Secure Transaction Processing Systems

Namgyu Kim and Songchun Moon
Database Laboratory, Department of Management Engineering,
Korea Advanced Institute of Science and Technology, Graduate School of Management,
207-43 Cheongnyangri-dong, Dongdaemun-gu, Seoul 130-012 Korea
Email: ngk@kaist.ac.kr
Email: scmoon@kgsm.kaist.ac.kr

Yonglak Sohn
Department of Computer Engineering, Seokyeong University
Email: syl@skuniv.ac.kr

To prevent data from being accessed by unauthorized users, it is necessary for Credit Card Transaction Processing (CCTP) systems to use multilevel secure database management systems to control concurrent execution among multiple transactions. In CCTP systems, analytical transactions as well as mission critical transactions are executed concurrently, which causes difficulties in using traditional secure real-time transaction management schemes in the systems. In this paper, we propose a read-down secure single snapshot scheme (RS4) that is devised for secure real-time transaction management. By maintaining a snapshot as well as the working database, RS4 blocks covert-channels without causing a so-called priority inversion phenomenon. We introduce the process of RS4 protocol with some examples, present proofs of the devised protocol, and then evaluate performance gains by means of a simulation method.

ACM Classification: H.2 (Database Management)

1. INTRODUCTION

In most secure database management systems, data with different security classification levels can be accessed by users with different security clearance levels. This type of multiplicity is based on the access rule mechanism of so-called mandatory access control like restricted Bell-LaPadula (BL) model (Bell and La Padula, 1974), in which data can be accessed only by its authorized users. According to the restricted BL model, write operations should be allowed only when the security level of a transaction and that of target data are the same, while read operations can be allowed when the security level of a transaction is higher than or equal to that of target data. Although the restricted BL model controls information flows explicitly by read and write operations, illegal and indirect leakage of information may occur via covert channels which is caused by an interaction...
between transactions with different security levels. In the literature, there has been considerable research into methods to block covert channels. The non-interference approach, the most prominent of this research, tries to guarantee that any high level transaction cannot interfere with any low level transaction. This kind of research could be divided into three schemes, single version-based scheme (Amman and Jajodia, 1992; McDermott and Jajodia, 1992; George and Haritsa, 2000), restricted multiple version-based scheme (Mukkamala and Son, 1995), and multiversion-based scheme (Atluri et al, 1996; Keefe et al, 1993; Kim and Kim, 1998; Maimone and Greenberg, 1990), based on the number of data versions maintained. According to the number of versions, each scheme reveals trade-offs between the data availability and the cost of version management.

Previous research on database security has focused on some specific properties of transactions such as real-time transactions (Ahmed and Vrbsky, 2002; George and Haritsa, 2000; Lee et al, 1999; Mukkamala and Son, 1995; Park and Park, 1998; Son et al, 1996), and on-line analytical processing (OLAP) transactions (Priebe and Pernul, 2000). However, because they have only been devised to meet the requirements of individual application areas, they are not suitable for the complex environment in which various transactions with different properties are executed concurrently. Credit Card Transaction Processing (CCTP) systems are typical examples in the complex environment. CCTP systems should not only handle operational transactions such as usual credit transactions but also managerial ones such as transactions for fraud detection. The existence of a covert channel may have a critical impact on the secure execution of CCTP transactions. In CCTP systems, managerial transactions should be allowed to execute analytical operations on overall transactional histories while transactions of users must be protected from unauthorized access by other users. In this situation, sensitive information may be disclosed to end users by an interaction between managerial transaction issued by administrative users and operational transaction issued by end users. Example 1 shows this scenario.

**Example 1 (Illegal leakage of information via covert channel):** Let us suppose that there are 100 credit cards \((C_1, C_2, \ldots, C_{100})\) each of which is owned by a customer. The customers are carrying out usual credit transactions like buying on credit. Let us suppose that a \(C_1\) is stolen. This credit card may be abused because no one can be charged for the purchases that are made. To detect such illegal use of lost credit cards, many credit information companies use various data mining techniques like fraud detection. The fraud detection technique regards a credit transaction as suspicious if the transaction is abnormal when compared with previous usage patterns of the credit card. To minimize the damage caused by illegal usage of lost credit cards, it is necessary for the fraud detection transaction to be processed as promptly as possible. Let us assume that an analytical transaction \(F_a\) is running for the purpose of fraud detection together with other operational credit transactions. Via an interaction with another malicious user (i.e. the person in possession of \(C_{100}\)), \(F_a\) is able to send him some sensitive information (Figure 1).

The person in possession of \(C_{100}\) may issue an illegal credit transaction because he has acquired some sensitive information from \(F_a\) via a covert channel.

To prevent the scenario in Example 1 from occurring, it is necessary for the CCTP system to block every type of covert channel during scheduling. In Example 1, operational transactions could be regarded as on-line transactional processing (OLTP) transactions while managerial ones could be regarded as OLAP transaction. The scheduler devised for concurrency control in CCTP systems must therefore consider the following factors to satisfy requirements of both OLAP and OLTP transactions. First of all, OLTP transactions must be free from interference from high level OLAP
transactions. Next, even though OLAP transactions usually take a long time to run, the CCTP scheduler has to prevent OLAP transactions from being aborted repeatedly because of interference from OLTP transactions. It is, moreover, necessary to minimize the cost of version management and main memory management.

The rest of this paper is organized as follows. Section 2 presents overviews on previous techniques which are related to our work. In Section 3, we propose a read-down secure single snapshot scheme (RS4) as a new secure real-time concurrency control protocol for CCTP systems. The proofs of RS4 are provided in Section 4. The performance gains of RS4 are analyzed in Section 5. Finally, Section 6 concludes this paper.

2. RELATED WORKS
Lam et al (1998) proposed separate concurrency control algorithms for read-only transactions and for update transactions. Every read-only transaction declares that it will not issue any write operations. The authors demonstrated that data consistency could be preserved successfully by maintaining view consistency instead of imposing conflict-serializability (Bernstein et al, 1987) for read-only transactions. Performance gains of adopting view consistency rather than conflict-serializability as a control criterion are illustrated in Example 2.

Example 2 (Conflict-unserializable but view serializable schedule) (Lam et al, 1998): Let us suppose that there is a schedule which comprises of update transactions U3, U4, and U5 and read-only transactions R1 and R2 (Figure 2).

A conflict-serializability graph (CSG) (Bernstein et al, 1987) for the schedule in Figure 2 contains a cycle (Figure 3(a)). This shows that the schedule in Figure 2 does not satisfy the requirements of conflict-serializability, namely, strong consistency. However, let us consider the partial commitment order of the update transactions only. There are two possible serialization orders, U3 → U4 → U5 and U4 → U3 → U5. When we consider Q1 in conjunction with the update transactions only, we find the serialization order among them as in Figure 3(b). Similarly, if we consider Q2 in conjunction with the update transactions only, we find the serialization order among
Application of Unique View Consistency for Elimination of Covert Channels

them as in Figure 3(c). It should be noted that the execution of Q1 does not have any effect on Q2, and vice versa, because Q1 and Q2 are read-only transactions. Consequently, the schedule in Figure 2 can be regarded as a view consistent one although Q1 and Q2 may be perceived as having different serialization order of update transactions.

To preserve view consistency, read-from graph (RFG) can be used as a tool for validation. As far as the RFG is concerned, every edge should be appended to the graph if and only if there is a write-equal-read-equal conflict, a read-down-write-equal conflict or a write-equal-read-down conflict. That is to say, a read-equal-write-equal conflict or a write-equal-write-equal conflict does not append any edges to RFG. Considerable performance gains could be acquired by adopting view consistency for read-only transactions. RS4 adopts modified view consistency as a control criterion perceiving the fact that a high level transaction can be regarded as a read-only one because it may conflict with low level transactions only by issuing read-down operations.

George and Haritsa (2000) proposed the dual approach, which handles real-time and security requirements separately. The conflict between transactions with the same security levels was named...
*intra-conflict* and the conflict between transactions with different security levels was named *inter-conflict*. Let us define two notations, \( L(T_i) \) and \( L(x) \), as the security level of transaction \( T_i \) and that of data \( x \) respectively. When \( L(T_i) > L(T_j) = L(x) \), an inter-conflict might occur only if there is a conflict between \( T_i \)'s *read-down* operation on \( x \) and \( T_j \)'s *write-equal* operation on \( x \). By utilizing the fact that a covert channel may be opened only via an inter-conflict, the dual approach adopts a non-interference principle for controlling inter-conflict, while it obeys *real-time priority* (RT-Priority) for controlling intra-conflict. However, as shown in Example 3, this approach may not be free from the phenomenon of *priority inversion* when there is a conflict between \( T_i \) and \( T_j \) such that \( L(T_i) > L(T_j) \) and RT-Priority\((T_i) > RT-Priority(T_j)\).

**Example 3 (Priority inversion problem):** There is a schedule comprising of transactions \( T_C \) and \( T_U \) and data \( x \) such that \( L(T_C) > L(T_U) = L(x) \) and RT-Priority\((T_C) > RT-Priority(T_U)\). Let us suppose that a *write* lock for data \( x \) has already been acquired by a transaction \( T_U \). Then, a *read* lock for data \( x \) is requested by transaction \( T_C \). If \( T_C \)'s *read-down* operation is executed immediately and \( T_U \) aborted, a covert channel may be opened by interference of \( T_C \) by \( T_U \). To block this covert channel, the dual approach schedules the *read-down* operation of \( T_C \) after \( T_U \) has released its lock on data \( x \). As a result, for the benefit of preserving security, \( T_C \) ought to be delayed by \( T_U \) which has lower RT-Priority than \( T_C \).

Priority inversion phenomenon is caused by contradictory requirements between non-interference and RT-Priority. By noting that every inter-conflict can be prevented in advance by maintaining one additional copied version as well as a working database, RS4 can separate the requirements of real-time from those of security.

Mukkamala and Son (1995) have devised the *Secure Real-Time Two-Phase Locking protocol* (SRT-2PL) for secure real-time transaction management. To block covert channels, SRT-2PL maintains an additional copied version of the database, which is named a secondary copy, as well as the primary copy. The secondary copy could be accessed only by transactions with a higher security level than that of data objects, while the primary copy may be accessed in every other case. The newly updated data in the primary copy should be stored in an extra queue temporarily. With the passage of time, the data values in the secondary copy need to be replaced with the value of the queued data. This data structure enables SRT-2PL to meet secure real-time requirements by removing inter-conflict in advance and by adopting RT-Priority to handle intra-conflict. SRT-2PL, however, shows degradation in concurrency of transactions, mainly due to the fact that every transaction must acquire locks on all data, which are planned to be accessed by the transaction, at the beginning of its execution. Furthermore, as soon as a *write* operation is executed, SRT-2PL appends the newly updated data in the primary copy into the temporary queue in main memory. This policy may produce the side effect that queue might be lengthened gradually, which is caused by augmentation in the duration of updated data’s stay in queue. Example 4 illustrates this phenomenon (\( x \) is assumed to have a value of \( x_0 \) as an initial value). 

**Example 4 (Overheads of augmented queue length):** Let us suppose that there is a schedule which consists of data \( x \) and transactions \( T_{C1}, T_{C2}, \) and \( T_U \) such that \( L(T_{C1}) = L(T_{C2}) > L(T_U) = L(x) \) (Figure 4).

In the SRT-2PL schedule, \( T_{C2} \) reads \( x_U \) which has been written by \( T_U \) while \( T_{C1} \) reads \( x_0 \) in Figure 4. For the purpose of making \( T_{C2} \) able to access \( x_U \), SRT-2PL enforces that \( x_U \) must be appended into the queue directly after the execution of \( T_U \)'s *write* operation. \( x_U \) can be removed from the queue...
only after $T_{C2}$, which has read $x_{U}$, is committed. As a result, $x_{U}$ ought to remain in the queue from $t_{2}$ till $t_{6}$. On the contrary, RS4 appends the newly updated value into the queue only after the commit of the transaction which has issued the write operation. Consequently, $x_{U}$ remains in the queue from $t_{4}$ till $t_{6}$ by RS4 in Figure 4. RS4 could make the duration of the temporary data's stay in the queue shorter than that of SRT-2PL though it provides $x_{0}$ to the read-down operation of $T_{C2}$.

3. READ-DOWN RELATIONSHIP-BASED SECURE ONE-SNAPSHOT PROTOCOL

3.1 Data Structure and Control Criteria

To block covert channels without causing a priority inversion problem, RS4 maintains a snapshot (SS) as well as a working database (WDB). When $L(T_{i}) = L(x)$, $T_{i}$'s read-equal or write-equal operations on $x$ ought to be executed in the WDB. On the contrary, $T_{i}$’s read-down operation on $x$ should read the value of $x$ from SS when $L(T_{i}) > L(x)$. According to the restricted BL model, an inter-conflict may occur only if there is a conflict between a read-down operation of a high level transaction and a write-equal operation of low a level transaction. RS4 enforces that the data $x$ in the SS should be retrieved by a read-down operation of $T_{i}$ when $L(T_{i}) > L(x)$, while $x$ in the WDB is updated by a write-equal operation of $T_{j}$ when $L(T_{j}) = L(x)$. Consequently, RS4 is able to block covert channels without a priority inversion problem by removing the causal elements of inter-conflict. In the case of intra-conflict, previous concurrency control techniques can be used to manage transactions without the threat of a covert channel because they belong to the same security class. Data values in the SS cannot be updated directly by write operations because the SS could be accessed only by read-down operations. Data in the SS should be, therefore, replaced with recent data from the WDB in order to prevent read-down operations from reading excessively stale data from the SS. In this paper, the term publish stands for such replacements and the notation $Pub(\cdot)$ denotes its operation. $Pub(T_{i})$, for example, means the operation of publishing all data, which has been written by the committed transaction $T_{i}$, from the WDB to the SS. RS4 utilizes a publishing order graph (POG) to arrange publication order while preserving data consistency.

View consistency is a weaker requirement than conflict-serializability because read-only transactions are not required to observe the same serialization order of update transactions. Read-only transactions can each see different serialization orders. We propose a new control criterion by utilizing the fact that every high level transaction can be regarded as a read-only one from the viewpoint of the SS. However, it should be noted that high level transactions may actually execute write operations and therefore may have an influence on each other. It means that high level transactions should each see the same serialization order. In this paper, we propose a Unique View consistency (UVC) as a new control criterion. UVC has a weaker requirement than conflict-serializability while it has a stronger one than view consistency. UVC allows transactions with
different security levels to each have different serialization orders. However, UVC enforces that transactions with the same security levels have the same serialization order.

3.2 Rules for arranging publication order

The POG specifies the publication order of transactions, and it is constructed on the basis of the read-from relationship (Lam et al., 1998). When a transaction is committed, an edge should be appended to the POG only if a read-from dependency is detected. Every publication ought to be executed in accordance with the sequence of the POG. For example, Pub(Ti) could be executed after Pub(Td) when \( T_d \rightarrow T_i \) appears in the POG. In this subsection, we define Rules 1, 2, and 3 for the case of appending edges to the POG and Rules 4 and 5 for processing publications in a predefined order. Every Rule should be invoked by commit operations of transactions and applied in order of Rules 1, 2, 3, 4, and 5.

Data consistency may not be preserved if publications are executed as soon as their transactions are committed (Figure 5). To begin with, let us take only TS and TC into consideration. TS reads data k which has been written and published by TC while it reads data t which has not been affected by TC yet. In this case, TS has read inconsistent data because it has been under the influence of the partial effect of TC. Data consistency can be validated by confirming that there is no cycle in the read-from graph (RFG) (Lam et al., 1998). Note that a RFG is used only for the purpose of checking the data consistency of produced schedule while a POG is used for arranging publication order while preserving the consistency. In the RFG for Figure 5, we can detect a cycle between TS and TC which consists of edge \( TS \rightarrow TC \) on data t and edge \( TC \rightarrow TS \) on data k. To prevent this type of cycle from being created, RS4 formulates a definition of Rule 1.

**Rule 1 (Edge for direct publication dependency by inter-conflict):** For committed transaction \( TU \), active transaction \( TC \), and data \( x \) such that \( L(TC) > L(TU) = L(x) \), append \( TC \rightarrow TU \) to POG if \( r_c(x) < w_u(x) \) either or \( w_u(x) < r_c(x) \) is detected in a schedule.

In accordance with Rule 1, edge \( TS \rightarrow TC \) should be appended to the POG when \( TC \) is committed. Therefore, the publication of \( TC \) ought to be postponed until \( TS \) published. Data consistency is not violated because \( TS \) reads t and k which have not been affected by \( TC \). Again, let us take not only \( TS \) and \( TC \), but also \( TU_1 \) into consideration. At the stage of \( TU_1 \)’s commit, there is no active transaction which has executed a read-down operation on x and y. Therefore, \( TU_1 \) can be published immediately after its commit because no edges need to be appended to the POG in accordance with
Rule 1. However, such an immediate publication may induce a cycle in the RFG because it makes $T_S$ reads that was $x$ written by $T_{U1}$. Accordingly, Rule 2 needs to be defined for the purpose of postponing the execution of $Pub(T_{U1})$.

**Rule 2 (Edge for indirect publication dependency by inter-conflict):** For committed transaction $T_C$, $T_{U1}$, and data $x$ such that $L(T_C) > L(T_{U1}) = L(x)$, append $T_C \rightarrow T_{U1}$ to POG if $r_C(x) <_H w_{U1}(x)$ either or $w_{U1}(x) <_H r_C(x)$ is detected in a schedule and a node for $T_C$ already exists in POG.

Rule 2 states that a read-from relationship between $T_C$ and $T_{U1}$ ought to be examined before $T_{U1}$’s publication, when $T_C$ represents a transaction that has been committed after executing a read-down operation but is not published yet. In this situation, let us take $T_{U2}$ into consideration together with $T_S$, $T_C$, and $T_{U1}$. At the stage of $T_{U2}$’s commit, there is no active or committed transaction which has executed a read-down operation on $z$. Therefore, $Pub(T_{U2})$ can be executed as soon as $T_{U2}$ is committed because no edges need to be appended to the POG in accordance with Rule 1 or Rule 2. Nevertheless, a cycle may be induced again in the RFG when $T_S$ executes a read-down operation on $z$ which has been written by $T_{U2}$. In this case, the read-from dependency of $T_S \rightarrow T_C$ has been delivered to $T_{U2}$ via dependencies of $T_C \rightarrow T_{U1}$ and $T_{U1} \rightarrow T_{U2}$. Accordingly, the execution of $Pub(T_{U2})$ should be postponed until $T_{U1}$ is published. RS4 formulates a definition of Rule 3 to prevent such a dependency from occurring.

**Rule 3 (Edge for publication dependency by intra-conflict):** For committed transaction $T_{U1}$, $T_{Uk}$, and data $x$ such that $L(T_{U1}) = L(T_{Uk}) = L(x)$, append $T_{Uk} \rightarrow T_{U1}$ to POG if $w_{Uk}(x) <_H r_{U1}(x)$ is detected in a schedule and a node for $T_{Uk}$ already exists in POG.

In Rule 3, it ought to be noted that $T_{Uk} \rightarrow T_{U1}$ should be appended to the POG only if $w_{Uk}(x) <_H r_{U1}(x)$ is detected in a schedule when $L(T_{U1}) = L(T_{Uk})$. That is to say, no edges or nodes need to be appended to the POG if $w_{Uk}(x) <_H w_{U1}(x)$ or $r_{Uk}(x) <_H w_{U1}(x)$ is detected when $L(T_{U1}) = L(T_{Uk})$. Rule 3 was formulated based on the UVC. Because every transaction can execute only read-down operations on low level data, it is justifiable for RS4 to regard high-level transactions as read-only ones from the viewpoint of low level data. During the publication process, from the viewpoint of the SS, every high level transaction can be regarded as a read-only one regardless of whether it has executed a write operation or not. In the case of $T_{U3}$, it can be published immediately after its commit because it participated in the conflict only by issuing a write operation on $z$. The schedule in Figure 5 should be replaced with Figure 6 when Rules 1, 2, and 3 are applied. A modified schedule is assured to preserve the unique view consistency because there is no cycle in its RFG. Detailed proofs are provided in Section 4.
Next, we define the rules for executing publications in order of dependency in the POG. Rules 1, 2, and 3 should be invoked when a transaction $T_i$ is committed. $Pub(T_i)$ can be executed immediately after $T_i$’s commit, if there is still no node for $T_i$ in the POG after the above rules have been applied. We define Rule 4 for such a case. Moreover, $T_i$ can be published if there is no incoming edge to node $T_i$ in the POG, even though $T_i$ already exists in the POG. Rule 5 should be applied in this case. Rule 5 is defined by a recursion with the intention of cascading publications being enabled.

**Rule 4 (Immediate publication rule):** Execute $Pub(T_i)$ if a committed transaction $T_i$ does not exist in POG.

**Rule 5 (Recursive publication rule):** If a committed transaction $T_i$ exists in POG but there is no incoming edge to it, publish it and remove all outgoing edges of $T_i$ from POG. Rule 5 could be recursively applied to $T_i$’s dependents with modified POG.

Operations of the five rules for Figure 6 are as follows. Edges, $T_S \rightarrow T_C, T_C \rightarrow T_{U1},$ and $T_{U1} \rightarrow T_{U2},$ should be appended to the POG in accordance with Rules 1, 2, and 3 respectively. As a result, $T_C, T_{U1},$ and $T_{U2}$ ought to postpone their publication until $T_S$ is published. On the contrary, $T_{U3}$ can be published immediately in accordance with Rule 4 because no edges for $T_{U3}$ need to be appended to the POG in accordance with Rules 1, 2, and 3. When $T_S$ is committed, Rule 5 should be invoked by $T_S$ because $T_S$ has no incoming edge to it in the POG. By adopting Rule 5 recursively, $T_C, T_S$’s dependent, can be published after $T_S$’s publication. In this way, total order of publications may be arranged to $T_{U3} \rightarrow T_S \rightarrow T_C \rightarrow T_{U1} \rightarrow T_{U2}$.

### 3.3 Algorithm for RS4 Protocol

The Rules 1, 2, 3, 4, and 5 could be summarized in a following pseudo-code.

```plaintext
when transaction $T_i$ is committed{
    for every active transaction $T_d$ such that $L(T_d) > L(T_i)$
        if $r_d(x) < H w_i(x)$ or $w_i(x) < H r_d(x)$ exists in schedule H, then add $T_d \rightarrow T_i$ to POG //Rule 1//
    for every committed transaction $T_d$ in POG such that $L(T_d) > L(T_i)$
        if $r_d(x) < H w_i(x)$ or $w_i(x) < H r_d(x)$ exists in schedule H, then add $T_d \rightarrow T_i$ to POG //Rule 2//
    for every committed transaction $T_d$ in POG such that $L(T_d) = L(T_i)$
        if $w'_d(x) < H r_i(x)$ exists in schedule H, then add $T_d \rightarrow T_i$ to POG //Rule 3//
        if there is still no node for $T_i$ in POG, then execute $Pub(T_i)$ //Rule 4//
    else if there is no incoming edge to the node for $T_i$, then execute recursive_Pub($T_i$) //Rule 5//
}
recursive_Pub($T_i$){
    for every transaction $T_k$ such that $T_i$ à $T_k$ exists in POG{
        delete edge of $T_i \rightarrow T_k$ from POG
        if there is no more incoming edge to $T_k$, then execute recursive_Pub($T_k$)
    }
    delete node for $T_i$ from POG
    execute $Pub(T_i)$
}
```
4. PROOFS OF CORRECTNESS

In this section, we provide theoretical bases for RS4 by means of semiformal proofs.

**Property 1 (Covert channel-free property):** A low-level transaction is never interfered by high-level one due to inter-conflict.

*Proof:* By restricting the BL model, an inter-conflict may occur only between T_i’s *read-down* operation on x and T_j’s *write-equal* operation on x such that \( L(T_i) > L(T_j) = L(x) \). Because RS4 offers two separate versions to *read-down* and *write-equal* operations, there is no possibility of inter-conflict being occurred.

**Property 2 (Priority inversion-free property):** There is no priority inversion in RS4 schedule.

*Proof:* Let us suppose that a transaction T_j requests a lock on x which has been already acquired by T_i and at least one of those locks is a *write* lock. Only in the following two cases may a phenomenon of priority-inversion arise.

**Case 1 (Issuing write-equal lock on already read-down scheduled data):** \( L(T_i) > L(T_j) = L(x) \) and RT-Priority(T_i) > RT-Priority(T_j)

**Case 2 (Issuing read-down operation on already write-equal locked data):** \( L(T_i) = L(x) < L(T_j) \) and RT-Priority(T_i) < RT-Priority(T_j)

Case 1 describes the situation where T_j requests a *write-equal* operation after T_i’s execution of a *read-down* operation. In the data structure of RS4, a high-level transaction need not acquire any locks on low level data because the *read-down* operation of a high level transaction reads the data from the SS. Therefore, T_j can acquire a *write* lock on x without being delayed by or aborting T_i. Much the same proofs can be extended to case 2. Therefore, no type of priority-inversion can be detected in the schedule produced by RS4.

**Lemma 1 (Conflict-serializability among the same level transactions):** All transactions with the same security level are conflict-serializable.

*Proof:* RS4 has no additional restrictions for scheduling transactions with the same security level. Therefore, conflict-serializability among same level transactions can be preserved by applying traditional concurrency control schemes. More formal proofs could be found in (Bernstein *et al.*, 1987).

**Property 3 (Unique view consistency among all committed transactions):** All committed transactions are unique view consistent regardless of their security levels.

*Proof:* Let us suppose the schedule which comprising transaction T_{C1}, T_{C2}, ..., and T_{Cn} and T_{U1}, T_{U2}, ..., and T_{Um} such that \( L(T_{C1}) = L(T_{C2}) = ... = L(T_{Cn}) > L(T_{U1}) = L(T_{U2}) = ... = L(T_{Um}) \). The sequential order of \( T_{C1} \rightarrow T_{C2} \rightarrow ... \rightarrow T_{Cn} \) and \( T_{U1} \rightarrow T_{U2} \rightarrow ... \rightarrow T_{Um} \) could be assumed without damaging generality because all transactions with the same security level are conflict-serializable by Lemma 1. A schedule can be regarded as a unique view consistent one if its corresponding RFG has no cycle. We can, with generality, assure that the only following types of cycle might be constructed in RFG.
Case (Cycle with inter-conflicts): $T_{Ci} \rightarrow T_{Cj} \rightarrow T_{Ut} \rightarrow T_{Uk} \rightarrow T_{Ci}$ (for $1 \leq i \leq j \leq n$, and $1 \leq t \leq k \leq m$)

In this case, $T_{Uk} \rightarrow T_{Ci}$ describes that $T_{Ci}$ reads the data, which has been written by $T_{Uk}$, from the SS after the execution of $Pub(T_{Uk})$. However, $Pub(T_{Uk})$ ought to be postponed until after $Pub(T_{Cj})$ owing to the dependencies of $T_{Cj} \rightarrow T_{Ut}$ and $T_{Ut} \rightarrow T_{Uk}$. It means that $Pub(T_{Uk})$ should be executed after $T_{Ci}$’s commit. Thus, it is not possible for $T_{Ci}$ to read the data from the SS which has been written and published by $T_{Uk}$, because $T_{Ci}$ must precede $T_{Cj}$ according to the sequential order assumption among same level transactions. Hence, the cycle in this case cannot be constructed by RS4. In conclusion, all committed transactions are unique view consistent regardless of their security levels.

5. EXPERIMENTS AND PERFORMANCE EVALUATIONS

We evaluated the performance of RS4 and SRT-2PL using a simulation. In addition to SRT-2PL, there are several other secure real-time concurrency control mechanisms such as the ones presented in (Ahmed and Vrbsky, 2002). In this simulation, SRT-2PL was chosen for the performance comparison because of its following characteristics. First of all, SRT-2PL tries to block covert channels completely without bringing about priority inversion problems. Its aim is exactly the same as that of RS4. Furthermore, SRT-2PL has a similar data structure (i.e. two data versions) to RS4. However, the mechanism in Ahmed and Vrbsky (2002) considers covert channels among adjacent security levels as less dangerous while the ones among distant security levels as more dangerous ones. This approach cannot be directly compared to RS4 using the same criterion for performance evaluation. For that reason, we evaluated the performance of RS4 and SRT-2PL only.

5.1 Input parameters and performance indices

A list of parameters and their values are presented in Table 1. The values in Table 1 have been set in accordance with (Agrawal et al, 1987) for the purpose of realistic comparisons with previous studies.

<table>
<thead>
<tr>
<th>System Parameters</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>num_cpus</td>
<td>2 CPUs</td>
<td>Number of cpus</td>
</tr>
<tr>
<td>num_disks</td>
<td>4 Disks</td>
<td>Number of disks</td>
</tr>
<tr>
<td>cpu_cc_delay</td>
<td>7.5 milliseconds</td>
<td>CPU time for controlling concurrent execution</td>
</tr>
<tr>
<td>obj_io_delay</td>
<td>35 milliseconds</td>
<td>I/O time for accessing an object</td>
</tr>
<tr>
<td>obj_cpu_delay</td>
<td>15 milliseconds</td>
<td>CPU time for accessing an object</td>
</tr>
<tr>
<td>S_min</td>
<td>4</td>
<td>Minimum slack factor</td>
</tr>
<tr>
<td>S_max</td>
<td>10</td>
<td>Maximum slack factor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database and Transaction Parameters</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>db_size</td>
<td>600, 800, 1,000, 1,200, 1,400 and 1,600 pages</td>
<td>Number of objects in database</td>
</tr>
<tr>
<td>num_terms</td>
<td>30, 60, 90, 120, 150, 200 and 250 terminals</td>
<td>Number of terminals</td>
</tr>
<tr>
<td>tr_size_min</td>
<td>3 pages</td>
<td>Size of smallest transaction</td>
</tr>
<tr>
<td>tr_size_max</td>
<td>4, 6, 8, 10, 15, 20 pages</td>
<td>Size of largest transaction</td>
</tr>
<tr>
<td>update_pct</td>
<td>5, 10, 15, 20, 25, 30, 40 and 50%</td>
<td>Percentage of update operations</td>
</tr>
<tr>
<td>max_e_levels</td>
<td>2, 4, 6, 8, 10 and 12 levels</td>
<td>Number of security levels</td>
</tr>
</tbody>
</table>

Table 1: List of parameters and their values
In Table 1, default values for simulation are marked with underlines. For example, when we investigate the effect of $db\_size$ on performance indices, we set $num\_terms = 200$, $tr\_size\_min = 3$, $tr\_size\_max = 10$, $update\_pct = 25$, and $max\_c\_level = 8$. Each transaction has the size between $tr\_size\_min$ and $tr\_size\_max$. We ran 2000 transactions for each simulation. Each simulation was executed three times with different random seeds. The average values of the three executions were used for performance analysis. We defined the following indices to evaluate the performance gains of each protocol.

- $\text{resp\_time}$: Average response time for each transaction
- $\text{queue\_ratio}$: Average ratio of temporary queue in main memory to database size
- $\text{deadline\_miss\_ratio}$: Average ratio of the number of deadline missed transactions to that of total committed transactions

This simulation was performed using a Microsoft Visual C++ 6.0 compiler and CSIM version 18 (Schwetman, 1992) on the Windows XP operating system.

5.2 Simulation results and their interpretations

In this subsection, we evaluate performance gains of RS4 and SRT-2PL in respect of $\text{resp\_time}$, $\text{queue\_ratio}$, and $\text{deadline\_miss\_ratio}$.

5.2.1 Effect of version management policy on response time

In all cases of this simulation, RS4 showed shorter average response times than that of SRT-2PL. Furthermore, the difference in $\text{resp\_time}$ increased when there was heavy contention among transactions. That is mainly due to the fact that the heavy contention may bring about a more serious delay of transactions in the case of SRT-2PL because SRT-2PL enforces every transaction to acquire all potential locks at the beginning of its execution. Figure 7 illustrates effect of $tr\_size\_max$ on $\text{resp\_time}$ for SRT-2PL and RS4.

![Figure 7: resp\_time with tr\_size\_max varied](image)
5.2.2 Effect of update timing on queue length
Both SRT-2PL and RS4 should store updated data values in a temporary queue for version management. SRT-2PL appends the newly updated data of the primary copy into the queue as soon as a write operation is executed. On the contrary, RS4 appends newly written value into the queue only after commit of the transaction. Consequently, RS4 holds temporary data which remains in the queue for shorter periods than SRT-2PL. In all cases of this simulation, RS4 had shorter average queue lengths than SRT-2PL. Furthermore, such difference in queue_ratio increases when there is heavy contention among transactions. Figure 8 shows the effect of update_pct on queue_ratio.

5.2.3 Effect of real-time policy on deadline observance.
It is known that the aggregate response time cannot effectively represent individual transaction deadlines. Therefore, deadline_miss_ratio should be examined to analyze deadline observance property of real-time transactions. For this paper, the deadline of a transaction is determined by Definition 1.

**Definition 1 (Deadline of a transaction):** The deadline of a transaction $T_i$ is defined as follows:

$$DL(T_i) = ARR(T_i) + SLACK \times EXP(T_i)$$

where $DL(T_i)$ is deadline of $T_i$, $ARR(T_i)$ is the arrival time of $T_i$, $EXP(T_i)$ is the expected execution time of $T_i$, and SLACK is the slack factor. In our simulation, SLACK is uniformly chosen from the range $[S_{min}, S_{max}]$.

Figure 9 illustrates the effect of num_terms on deadline_miss_ratio for SRT-2PL and RS4. For both protocols, the deadline_miss_ratio increases as the num_terms increases. The rate of increase of SRT-2PL is, however, shown to be larger than that of RS4. This means that RS4 can meet deadline constraints better than SRT-2PL when a large number of terminals are contained in the system. CCTP usually assumes a heavy congestion of transactions. Therefore, RS4 can be considered as a more appropriate protocol for a CCTP environment than SRT-2PL in the aspect of deadline observance.
6. CONCLUSIONS

In this paper, we proposed the RS4 protocol as a new secure real-time concurrency control protocol for CCTP. RS4 can block covert channels without giving rise to priority inversion problems by removing causal elements of inter-conflicts. Moreover, the flexibility of adopting concurrency control algorithms enables RS4 to improve concurrency among transactions. RS4 can, in addition, minimize the cost of version management by maintaining only one additional copied version, and reduce the overheads of main memory management by keeping the temporary queue short during the updating process of the copied version.

We also introduced UVC as a new control criterion in this paper. As a matter of fact, a part of the performance gain comes from a weakened control criterion. UVC has weaker requirements than conflict-serializability while it has stronger ones than view consistency. However, UVC requires that transactions with the same security levels have the same serialization order. UVC utilizes the fact that a high level transaction can be regarded as a read-only one because it may conflict with low level transaction only by issuing read-down operations. UVC can be considered to have adequate properties to be applied to multilevel secure database management systems.

REFERENCES


**BIOGRAPHICAL NOTES**

**Namgyu Kim** received his BSc in Computer Engineering from Seoul National University, Korea in 1998 and MSc in Management Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2000. He is a PhD candidate in Management Engineering at KAIST. His research interests include database security, data modeling and transaction processing.

**Songchun Moon** received his PhD in Computer Science from the University of Illinois at Urbana-Champaign in 1985. He has been working for KAIST since then. He has developed a multi-user relational database management system, IM, which is the first prototype in Korea in 1990 and also a distributed database engine, DIME, in 1992, which is another first prototype ever in Korea. His research interests include database security, data modeling and transaction processing.

**Yonglak Sohn** received his BSc in Computer Science from Kyeongbuk National University, Korea in 1986, his MSc in Computer Science from Korea University, Korea in 1988, and the PhD in Information and Communication from KAIST in 2000. Since 1995, he has been with the Department of Computer Engineering at Seokyeong University, Seoul, Korea where he is now an Assistant Professor. His current research interests include database security, XML, and data warehousing.