Secure Real-Time Transaction Processing

A Thesis
Submitted for the Degree of
Doctor of Philosophy
in the Faculty of Engineering

By
Binto George

Supercomputer Education and Research Centre
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012, INDIA

December 1998

“Dedicated to the dedicated”

Abstract

A **Real-Time Database System (RTDBS)** is a transaction processing system that is designed to handle workloads where transactions have completion *deadlines*. Many real-time database applications arise in safety-critical installations and military systems where enforcing **security** is crucial to the success of the enterprise. Unfortunately, the mechanisms for achieving security and for meeting transaction deadlines often work at cross-purposes, making it difficult to design efficient secure RTDBS. In our study, we have addressed this problem for real-time applications with "firm deadlines", wherein transactions that miss their deadlines are considered to be useless and are therefore "killed", that is, permanently aborted without being executed to completion. Accordingly, the performance metric is the percentage of missed deadlines.

Within the above context, we have focussed on the design of high-performance secure *concurrency control* (CC) protocols and secure *buffer management* (BM) policies, two vital components of database systems. In particular, we present the **S2PL-WAIT** CC protocol, which employs a novel approach that allows the RTDBS to *simultaneously* use different CC mechanisms for guaranteeing security and for improving the real-time performance. Using a detailed simulation model, we evaluate the performance of S2PL-WAIT against other real-time CC protocols for a variety of security-classified transaction workloads and system configurations. The experimental results show that S2PL-WAIT consistently outperforms the existing protocols. We also present the **SABRE** BM policy which ensures efficient usage of buffer resources and incorporates several novel optimizations for improving the real-time performance. Performance evaluation of SABRE against a previously proposed unsecure real-time BM policy shows that SABRE provides security with only a minor drop in real-time performance, especially for applications with few security levels.

A fundamental problem with schemes for ensuring security in RTDBS is that of *fairness* – transactions belonging to higher security classes usually form a disproportionately large fraction of the transactions that miss their deadlines. To address this issue, we present the **GUARD** feedback-based admission control policy which ensures fairness without violating the information leakage bandwidth restrictions specified in military standards. Performance evaluation of GUARD shows that it provides close to ideal fairness without incurring a significant degradation of the overall real-time performance.

In summary, we suggest that designers of secure RTDBS may find the protocols presented in this thesis to be a good choice for achieving high-performance secure real-time transaction processing. To the best of our knowledge, the above results represent the first detailed study in the area of secure firm real-time transaction processing systems.

Publications

- "Secure Buffering in Firm Real-Time Database Systems",
  B. George and J. Haritsa
  Proc. of Intl. Conf. on Very Large Data Bases,
  New York, USA, August 1998

- "Secure Transaction Processing in Firm Real-Time Database Systems",
  B. George and J. Haritsa
  Proc. of ACM SIGMOD Conf.,
  Tucson, Arizona, USA, May 1997

- "Security Protocols for Real-Time Database Systems",
  B. George and J. Haritsa
  Proc. of Natl. Conf. on Software for Real-Time Systems,
  Cochin, India, January 1996

Acknowledgements

I would like to thank my research supervisor Dr Jayant Haritsa who trained me to be scientific, systematic and focussed. If this thesis meets any of the correctness constraints, it is only due to him. My research with him is really an unforgetful experience for the rest of my life. My studentship here was enriched by a great motivator – our chairman, Prof. Balakrishnan. I don't have words to thank him for the support he extended for everything.

I register my thanks to the thesis committee members, Prof. C. E. Veni Madhavan, Dr S. K. Nandy, Dr Anamitra Makur for their invaluable inputs for our research. I am thankful to the referees of my thesis, especially Prof. D. Janaki Ram, for their very useful suggestions. I express my thanks to Dr Ashok Subramanian, Dr Venkateshvaran, Dr Hansdah, Dr P. Venkataram, Dr Matthew Jacob, Dr V. Sriram, and Dr Govindarajan. I am thankful to the faculty of SERC and CSA for their overwhelming support in our research.

I am deeply greatful to Prof. Michael J. Carey for the encouraging e-mail he sent, at the beginning of my research. I am indebted to Dr. Peter Neumann and his secretary Nina for providing me the details of secure alpha work. I express my thanks to Prof. Krithi Ramamritham. I am thankful to Dr Sang H. Son and Mr Rasikan David for giving me the required information on their research. I am thankful to Prof. Sushil Jajodia for spending his valuable time with me during his Madras visit. I register my heart felt gratitude to Dr Vijay Atluri and Prof. Nabil Adam.

I owe a special thanks to my former teachers: Mr Jose K. Peechat, Dr Janahanlal, Prof. Paul Antony, Prof. K. Eapen, Prof. A. K. Menon, Mr Krishna R. Putcha, Dr. K.

Poulose Jacob, Mr Muralidharan and Mr David Peter.

I thank the SERC system staff: Mr R. Krishnamoorthy, Mr. H. Krishnamoorthy, Mr Mohan, Dr Sampath, Dr Sujan, Dr Ghosh, Mr A G Rao, Mr Gopakumar, Rana, Viji, Jaya, Sanoy, Maruthappan, CMC, Mr Gowranga, Ms Jaiganga, Venu, Ajay, Harsha, Naren, Vinod, Kumuda, Jyothi, Hema and Malini.

I thank the office staff: Mr Harish, Ms Rajalakshmi, Mr Balakrishnan, Mr Sasikumar, Ms Sarala, Ms Triveni, Mr Todur, and Ms Neela – your true love and kindness greatly enriched my life here. I particularly thank the Mess, Hostel and Hospital Staff.

My sincere gratitude to Anil Kumar, Narayana Swami, Rahul, and Danny for the lessons I received from them. I cannot forget Ramesh who fought and thought with me every day on simple and subtle issues. I thank all the members and former members of SSL and DSL: Pramod, Ramanan, Raman, S. Balki, MBK, Kalai, FRK, Maya, Priya, Vikram, Nisha, Abhijit, Madhavi, Bhuvana, Krishna, Vidya, Nars, Srikanth, Santhosh, Anurag, Saha, Manjusha, Srikanta and others.

Rajeev, you are my best friend. I express my gratitude to my friends: Ajayan, Anil, Syam, Bsri, Saroja, and Neeharika. Muthu, you have been a great inspiration and support for me all the time. Many thanks to Suresh, Rajesh, Ranjit, Ethiraj, Rohini, Karen, Pai, Paul and Beckey, Suresh, Neethi and KMS.

My sincere thanks to Mr Loverton, Dr Ahammed, Dr Prakash, and Mr. Rageev. The timely help extended by Sreeni for attending the IISc interview is unforgetful.

Finally, I thank Joju, Manju, Boby, Sobers, Anju, Tiju, Geo, Binju, Honey, Manu, Varughese, Leo and Roners. Sarin, I don't have to thank you. You hate that! I remember my parents, then Binu and Cinu. This acknowledgement remains open – I remember all your kind help and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Imagine the situation where an alien bomber could strike a critical target within ten minutes of entering your country's airspace, unless a pre-emptive mission is successfully launched and completed. The pre-emptive mission will involve the following three phases: The first phase consists of analyzing the data from the air force database and the numerous surveillance sensors mounted all around. The second phase involves preparation of a suitable defense plan based on the data analysis. In the third phase, the prepared plan is executed before the expiry of the time limit, that is, within ten minutes. You, as the commander of defense operations, are facing a problem of *three dimensional* complexity: (1) There is an enormous amount of data to be consulted, processed and updated, and while doing so the *database correctness* should be maintained; (2) There is a *time constraint* for the task — the mission must be completed within ten minutes; (3) During every stage of the mission, *data security* must be ensured — giving unauthorized access to the secret military information may be catastrophical. In this thesis we focus on the design of information systems that can simultaneously and effectively meet these three challenges, that is, on the design of **secure realtime database systems (SRTDBS)**.

1.2 Background

Applications that deal with large quantities of information typically use database systems for efficient data storage, processing and retrieval. For data-intensive applications that are faced with timing requirements, the ability of the underlying database system to satisfy transaction timing constraints becomes a key factor in determining both the feasibility of the application and the extent of its use [39]. A system that is specifically designed for efficiently processing such transactions is referred to as a **Real-Time Database System (RTDBS)**. The objective of the system is to meet the transaction deadlines, that is, to process transactions before their deadlines expire.

Many RTDBS applications arise in safety-critical installations and military systems where enforcing **security** is crucial to the success of the enterprise. Given that the US government itself is classifying about seven million new pieces of information every year [82], no further evidence is required for emphasizing the importance of information security. Surprisingly, however, the issue of providing security in RTDBS has received comparatively little attention although real-time database research has been underway for close to a decade now (see [101] for a survey). In our study, we partially address this lacuna by making a detailed investigation of the mechanisms for providing security in the context of real-time applications with "firm-deadlines" [40]. For such applications, completing a transaction after its deadline has expired is of no utility and may even be harmful. Therefore, transactions that miss their deadlines are "killed", that is, immediately aborted and discarded from the system without being executed to completion. Accordingly, the performance metric is *the percentage of missed transaction deadlines.*

For a transaction processing system to be secure, the security of all its components must be ensured. These include the *concurrency control manager*, the *buffer manager*, and the *recovery manager*. The concurrency control manager ensures database consistency in the face of multiple users simultaneously accessing the database. The buffer manager caches a portion of the disk-resident database in main memory and improves the system performance by reducing the number of disk operations. Finally, the recovery manager brings the database to a consistent state after a transaction or system failure. In an

SRTDBS, all these transaction processing components should be specifically designed to both ensure security and deliver good real-time performance. In this thesis we focus on designing high-performance secure concurrency control and secure buffer management protocols for the firm-deadline environment. The problem of designing recovery managers with such features is left for future research.

1.3 Security Violations

Imagine the situation wherein an RTDBS contains both "secret" and "public" data. Security violations in this RTDBS can occur if, for example, information from the "secret" database is transferred by corrupt high security transactions to the "public" database where they are read by conspiring low security transactions. Such *direct* violations can be eliminated by implementing the classical Bell–LaPadula security model [59] which imposes restrictions on the data accesses permitted to transactions, based on their security levels. The Bell–LaPadula model is not sufficient, however, to protect from "covert channels". A covert channel is an *indirect* means by which a high security transaction can transfer information to a low security transaction [57]. For example, if a low security transaction requests access to an exclusive resource, it will be delayed if the resource is already held by a high security transaction, otherwise it will be granted the resource immediately. The presence or absence of the delay can be used as a "signaling" or encoding mechanism by a high security transaction passing secret information to the low security transaction. Note that nothing "obviously illegal" has been done in this process by the conspiring transactions, from the system perspective.

Covert channels can be prevented by ensuring that low security transactions do not "see" high security transactions – this notion is formalized in [30] as *non-interference*, that is, low security transactions should not be able to distinguish between the presence or absence of high security transactions. This can be implemented, for example, by providing higher priority to low security transactions whenever a conflict occurs between a low security transaction and a high security transaction. From a system perspective, it translates to implementing transaction processing components that support the non-interference feature. In this thesis, we address the issue of designing such covert-channel-free components in a manner that minimizes the impact on the real-time performance objectives.

1.4 Design Challenges

An SRTDBS has to *simultaneously* satisfy two requirements, namely, provide security and minimize the number of missed deadlines. Unfortunately, the mechanisms for achieving the individual goals often work at cross-purposes. In an RTDBS, high priority is usually given to transactions with earlier deadlines in order to help their timely completion. On the other hand, in secure DBMS, low security transactions are given high priority in order to avoid covert channels (as described in Section 1.3). Now consider the situation wherein a high security process submits a transaction with a tight deadline in a secure real-time database system. In this case, it becomes difficult to assign a priority since assigning a high priority may cause a security violation whereas assigning a low priority may result in a missed deadline.

In addition to the above issue, there are several related problems that arise while integrating security into the RTDBS framework:

1. Given the paramount importance of security, the database system is forced to assign transaction priorities based primarily on security clearance levels and only secondarily on deadlines. A major problem arising out of this preferential treatment of low security transactions is that of *fairness* – high security transactions usually form a disproportionately large fraction of the transactions with missed deadlines. This is an especially problematic issue because it is the high security transactions, which are supposed to be having higher privileges, that are being discriminated against in favor of the low security transactions.

2. There may be a multitude of covert channel mediums in the transaction processing component, all of which have to be eliminated to make it truly secure. For example, in the Buffer Manager, the presence or absence of a delay in acquiring a free buffer

slot, or the presence or absence of a specific page in the buffer set, or the allocation of a particular (physical) buffer slot, could all be used as channel mediums.

3. Straightforward solutions to eliminate covert channels such as using "static" resource allocation policies where each transaction security class has a pre-assigned set of resources, or using "replication"-based policies wherein multiple copies of data pages are maintained, may result in very poor resource utilization. The goal therefore is to design "dynamic" and "one-copy" policies that are demonstrably secure.

4. Finally, "noise"-based approaches to addressing covert channels, wherein noise is introduced into the signaling channel – by executing *dummy* transactions, for example – are impractical for covert channels that use *data* as the medium (for example, presence or absence of a lock on a pre-determined data item). This is because, unlike physical resources which are typically few in number, the number of data items is usually enormous, especially in a database system. In fact, in heavily loaded systems, noise at the physical resources may be generated "for free", but this will probably never be the case for data since it is trivial to add an additional data item that is of no relevance to other transactions. Therefore, explicitly making data access covert-channel-free is more vital than doing the same for resource access.

In summary, the design of an SRTDBS involves three dimensional complexity as featured in Figure 1.1 since they have to ensure data consistency, enforce security, and provide good realtime performance. Our contributions towards the design of such systems are described in the next section.

1.5 Thesis Contributions

Our contributions are mainly on developing high performance secure concurrency control (CC) and buffer management (BM) techniques for the firm-deadline environment. In particular, we provide the following:

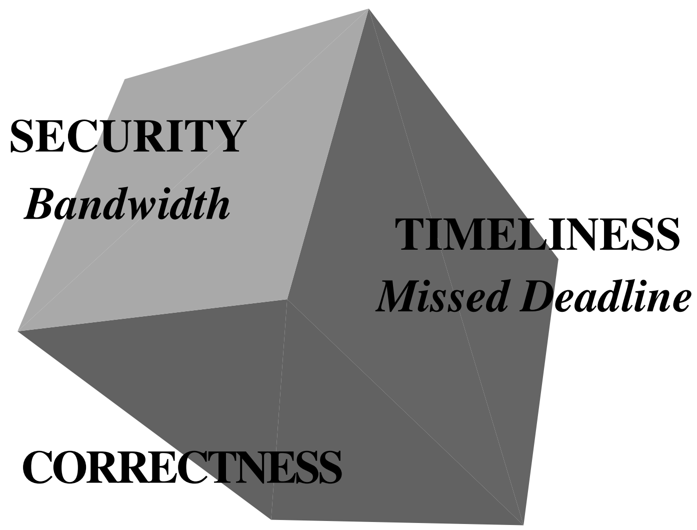


Figure 1.1: Complexity of Transaction Processing

1. We identify which among the previously proposed real-time CC protocols are capable of providing protection against *both* direct and indirect (covert channels) means of unauthorized access to data. That is, which protocols support the concept of *non-interference*. We show that some of the popular real-time CC protocols are inherently unsecure and therefore cannot be easily adapted to the secure environment.

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

3. We evaluate the effectiveness of a novel *dual* approach to secure CC wherein simultaneously *different* CC mechanisms are used for guaranteeing security and for improving real-time performance, respectively. In particular, we investigate the performance of various combinations of CC mechanisms for resolving inter-security-level and intra-security-level data conflicts. Our results show that one of these *hybrid* CC algorithms, **S2PL-WAIT**, performs even better than OPT-WAIT.

4. We evaluate the performance impact of ensuring direct security (by enforcing Bell–LaPadula conditions) as opposed to having no security at all. Our experiments reveal that the performance cost to be paid for ensuring direct security is rather small.

5. We study the performance impact of secure scheduling at the physical resources. Our experiments show that secure resource scheduling has a significant role in causing unfairness across security levels.

6. We examine the possibility of malicious subjects exploiting the *inherent* non-preemptive nature of certain resources such as, for example, the disks, for covert signaling. One solution to this problem that has been suggested in the literature [36] is to introduce an artificial delay in the servicing of disk requests. However, our experiments show that adverse performance implications result with this solution.

7. We identify the importance and difficulties of providing secure buffer management in the real-time transaction processing environment. Our analysis shows that the existing real-time BM protocols are unsuitable for the secure environment since they do not satisfy the non-interference property. To address this lacuna, we present **SABRE (Secure Algorithm for Buffering in Real-time Environments)**, a new BM protocol that provides complete covert-channel-free security. SABRE has been carefully designed to address all three components of buffer management – allocation, replacement and synchronization. Further, it is a dynamic one-copy protocol and therefore ensures efficient usage of buffer resources.

8. Using a detailed simulation model of a firm-deadline RTDBS, the real-time performance of SABRE is evaluated against unsecure conventional and real-time BM algorithms for a variety of security-classified transaction workloads and system configurations. Both inter-transaction locality and intra-transaction locality, as well as restart locality, are modeled in our experiments.

   To isolate and quantify the effects of BM on the system performance, we also evaluate two baselines, ALLHIT and ALLMISS, in the simulations. ALLHIT models an ideal system where every page access results in a buffer pool "hit", while ALLMISS models the other extreme – a system where every page access results in a buffer pool "miss".

   Our experiments show that (a) it is essential to include deadline-cognizance in the buffer manager for good real-time performance, and (b) SABRE efficiently provides security in that its real-time performance is not significantly worse than that of RT, an unsecure real-time BM protocol.

9. Our experiments demonstrate that guaranteeing security can result in considerable unfairness with regard to the distribution of killed transactions across transaction security levels. However, it appears fundamentally impossible to ensure fairness without compromising security to atleast a limited extent. We present **GUARD**, an adaptive admission-control-augmented protocol designed to achieve fairness while remaining within the information leakage bandwidth limits specified in the US military's "Orange Book" [25], which defines security standards. On experimental evaluation, it was found that GUARD ensured close to ideal fairness without really affecting the overall real-time performance.

In summary, we investigate various issues that arise in secure real-time transaction processing systems in this thesis. Our new secure concurrency control and buffer management protocols are shown, through a detailed simulation study, to provide the best real-time performance and maximum fairness as compared to a representative set of other protocols considered in our study. To the best of our knowledge, the results described here represent the *first* detailed study of the SRTDBS environment.

1.6 Organization

The remainder of this dissertation is organized in the following fashion: In Chapter 2, we describe the SRTDBS framework. Chapter 3 reviews the published research on SRTDBS. An SRTDBS performance model is described in Chapter 4. This chapter also includes an overview of the methodology, metrics and workload generator used in the simulation studies that follow. In our initial simulation experiments, we focus on the CC aspects – the performance of existing CC protocols is evaluated in Chapter 5 while new protocols are presented in Chapter 6. We then move on to considering buffer management issues in Chapter 7 – SABRE, our new BM policy, is developed in this chapter. Our second series of simulation experiments estimates the performance cost to be paid for ensuring BM security and is also described in this chapter. The GUARD protocol for improving fairness by relaxing the security requirements to those specified in the Orange security book is presented and evaluated in Chapter 8. Finally, Chapter 9 summarizes the main contributions of our study and outlines future avenues to explore.

Chapter 2

Consistency, Timeliness and Security

2.1 Introduction

As mentioned earlier, the secure real-time transaction processing domain is a complex union of database technology, real-time technology and information security. The purpose of this chapter is to review the basics of these three areas and then to describe the problems that arise in their combination. We begin this chapter with a discussion of traditional transaction processing and then move on to real-time database systems. Subsequently, we present the issues related to secure transaction processing. Finally, we describe the design, architectural and implementation issues for SRTDBS.

2.2 Transaction Processing

A database system consists of a collection of interrelated data and a set of programs that access the data [87]. The goal of the database system is to support efficient storage, retrieval and processing of large amount of data. Applications interact with the database system through *transactions*, which constitute the basic unit of work. Transactions provide the so-called **ACID** semantics, described below:

Atomicity: A transaction is an atomic unit of work comprising of many operations. If all the operations have been performed successfully, the transaction commits. If some operation of the transaction fails and the transaction is aborted, the partial results of the transaction are undone.

Consistency: An individual execution of a transaction transforms the database from one consistent state to another consistent state.

Isolation: A transaction does not make its database modifications visible to other transactions before its commitment. This guarantees the standard notion of database consistency called *serializability* [28], which states that the concurrent execution of a set of transactions should be equivalent to some serial execution of the same set of transactions. That is, the outcome of concurrent execution of transactions, as reflected in the stored data, should be the same as that produced by executing the transactions in some serial order.

Durability: Once a transaction has committed, the effects of the transaction are never lost despite any subsequent system failures.

The database system provides the above semantics through three components working cooperatively, namely, the *Concurrency Control Manager*, the *Buffer Manager*, and the *Recovery Manager*.

2.2.1 Concurrency Control Manager

The CC manager implements a protocol that ensures serializability by resolving concurrent data accesses in a manner that induces a serialization order among the conflicting transactions. Several serialization mechanisms have been proposed [11], including *locking*, *validation* and *timestamping*, and a rich variety of protocols have been developed based on these schemes.

Two methods, *blocking* and *restarts*, are available for resolving data conflicts, and virtually all CC protocols designed for "single copy" databases (where only a single copy of each data item is maintained) use some combination of these methods. A blocking decision causes one of the conflicting transactions to wait for the other to complete, while a restart decision causes one of the conflicting transactions to be rolled-back.

There are also "multiversion" systems where multiple copies of data elements are maintained in an attempt to reduce the data conflicts. Multiversion CC protocols such as *multiversion timestamp ordering*, *multiversion two phase locking*, etc. are available for ensuring serializability in such systems [11].

2.2.2 Buffer Manager

In most computer setups, the operating system is responsible for the management of physical resources. However, the policies used by the OS are often unsuitable to meet database objectives [97]. Therefore, some OS functionalities are re-implemented by database system builders to suit their requirements. In particular, database systems reserve a region of main memory to serve as a "database cache" of the persistent data on disk. This memory space is usually referred to as the *buffer pool* and is organized as a set of *slots*, each of which can hold a single data page.

A buffer page that matches the corresponding page on the disk is said to be "clean" whereas a modified page is said to be "dirty". Dirty pages have to be flushed to disk before they can be replaced by other pages. Sometimes, even without the need for replacement, dirty pages have to be "force-written" to disk – for example, at transaction commit time, all buffer pages containing log records related to the committing transaction have to be forced to the disk.

Transactions that intend to access the data in a buffer slot are required to first *pin* the slot in the appropriate mode (read or write). The pin ensures the physical consistency of the page during the access period. Multiple read pins may simultaneously exist on a page but only a single write-pin is permitted at a time. When the transaction completes its current usage of the pinned page, it issues the corresponding *unpin* command. Of course, it may access this same page again later following the pin–unpin procedure as before.

If a requested page is found to be already in the buffer pool, it is termed a "buffer hit", whereas if it has to be brought in from the disk, it is termed a "buffer miss". Buffer managers attempt to utilize the locality typically exhibited in database reference patterns to maximize the number of buffer hits. Three kinds of reference locality are usually observed: *inter-transaction locality* (i.e.“hot spots”), *intra-transaction locality* (transaction’s internal reference locality), and *restart locality* (restarted transactions usually make the same set of accesses as their original incarnation).

There are three components commonly associated with buffer management: *allocation*, *replacement* and *synchronization*. The allocation algorithm determines the assignment of buffers to transactions. The replacement algorithm determines which page is to be replaced in order to accomodate a new page if no empty buffers are currently available. Lastly, the synchronization algorithm ensures that conflicting pins are not held on a page at the same time.

2.2.3 Recovery Manager

The recovery manager is responsible for ensuring that the database contains all the effects of the committed transactions and none of the effects of aborted ones [11]. Most importantly, this guarantee is valid even in the presence of transaction or system failures.

Two well-known mechanisms for achieving the recovery objective are *shadow-paging* [63] and *write-ahead-logging (WAL)* [34]. In addition, for *distributed* databases, a *commit protocol* has to be implemented to ensure that *all* sites participating in the distributed execution of the transaction agree on the final decision (commit or abort). A large number of such commit protocols have been designed including the classical *Two Phase Commit (2PC)* protocol [64, 33], its variations such as *Presumed commit (PC)* and *Presumed Abort (PA)* [58, 71], and *Three Phase Commit (3PC)* [86].

2.2.4 Admission Control

A system that is overloaded with transactions may often deliver less than its best possible performance. Therefore, for performance improvement, some systems employ an *admission control* policy that controls the entry of transactions into the system. The decision to admit or deny a transaction into the system is taken based on the current transaction population and the “desired” population in the system. The desired population is determined based on some heuristics and the computation of the desired population is often based on the past history of the system performance. A transaction that is denied admission at the entry point maybe discarded from the system.

2.3 Real-Time Database Systems

Having discussed the basics of database transaction processing, we now move on to **Real-Time Database Systems (RTDBS)**. A Real-Time Database System (RTDBS) is a transaction processing system that is designed to handle workloads where transactions have timing constraints. A time constraint is, typically, expressed in the form of a **deadline**, that is, the application submitting the transaction would like it to be completed before the deadline expires. Accordingly, the objective of the RTDBS is to complete transactions before their deadlines expire. How soon a transaction completes with respect to its deadline is of no relevance at all – what is important is *whether or not* the transaction completes before its deadline. That is, having a transaction complete just before its deadline is no different to having it finish earlier. Therefore, in contrast to a conventional DBMS where the goal usually is to minimize transaction response times, the emphasis in an RTDBS is on meeting transaction deadlines.

Note that an RTDBS has to *simultaneously* satisfy two different kinds of constraints: data integrity constraints and real-time constraints (see Figure 2.1). This *dual* requirement makes real-time transaction scheduling more complex than task scheduling in conventional real-time systems or transaction scheduling in conventional database systems.

Based on how the application is impacted by the violation of time constraints, real-time applications can be grouped into three categories: 1. Hard Deadline; 2. Firm Deadline; and 3. Soft Deadline. These three categories are shown in Figure 2.2; in this figure, transaction T arrives into the system at time A_T with an associated deadline D_T. A transaction that completes successfully before its deadline is *InTime* and fails to complete before the deadline is *Late*. For a hard deadline application, missing a deadline is equivalent to a catastrophe symbolized by the exploding bomb shown in Figure 2.2a. Life-critical applications, such as flight control systems or missile guidance systems, belong to this category. For firm deadline or soft deadline applications, however, missing deadlines

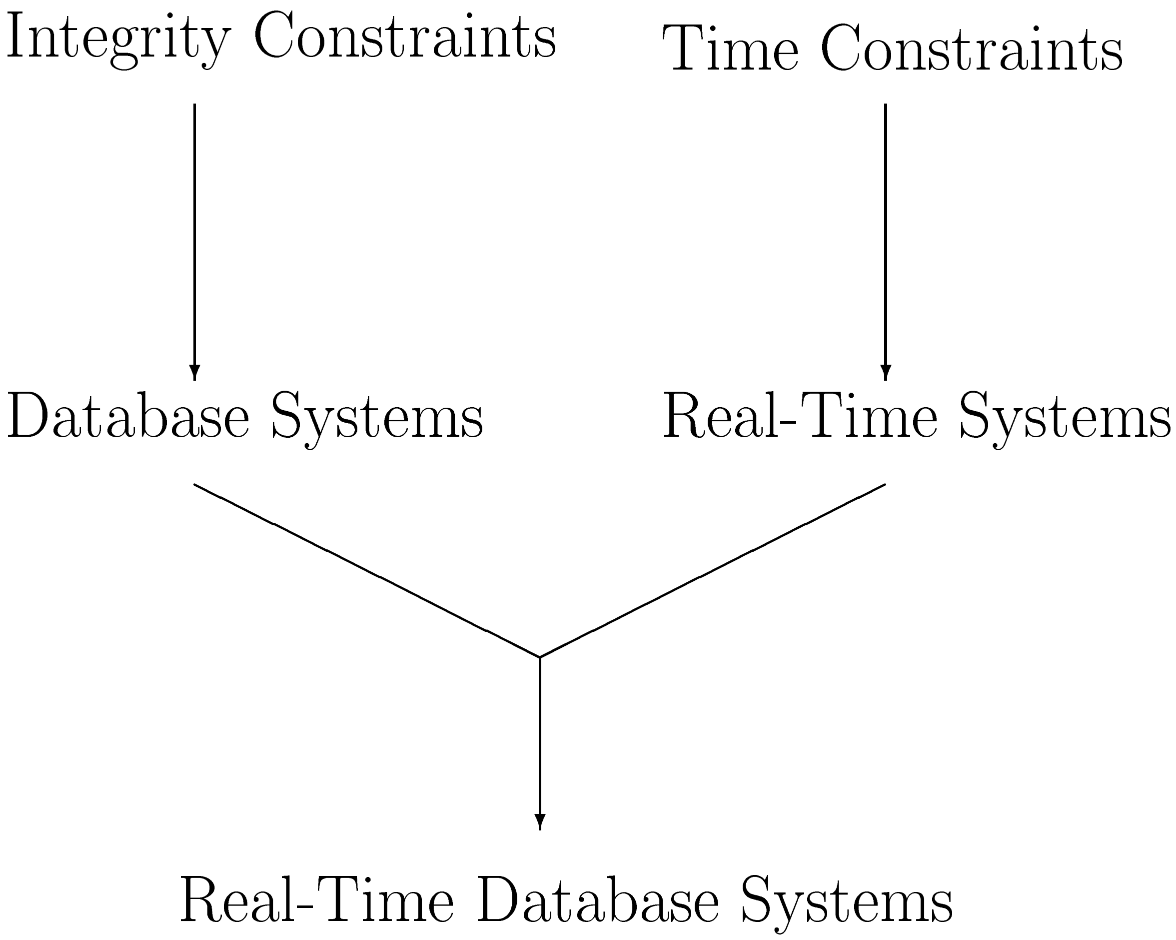


Figure 2.1: Constraints in Real-Time Database System

leads to a performance penalty but does not entail catastrophic results. The distinction between firm deadline and soft deadline applications lies in their view of late tasks. For firm deadline applications, completing a task after its deadline has expired is of no utility and in fact may be harmful. Therefore, late tasks are required to be permanently aborted and discarded as shown in Figure 2.2b. For soft deadline applications, however, there is some diminished utility to completing tasks even after their deadlines have expired (Figure 2.2c). Financial and manufacturing applications usually fall under either firm deadline or soft deadline categories.

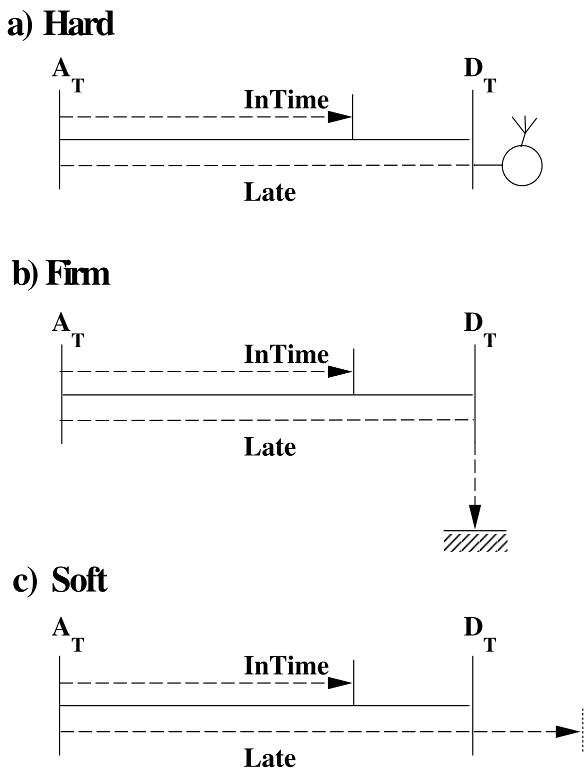


Figure 2.2: Deadline Types

Database systems for efficiently supporting hard deadline real-time applications, where *all* transaction deadlines have to be met, appear infeasible [96]. This is because there is usually a large variance between the average case and worst case execution times of a transaction. The large variance is due to transactions interacting with the operating system, the I/O subsystem, and with each other in unpredictable ways. Guaranteeing transactions under such circumstances requires an enormous excess of resource capacity to account for the worst possible combination of concurrently executing transactions [96].

Database systems for firm deadline or soft deadline applications, however, appear feasible since the system is only expected to make a "best effort" to meet transaction deadlines [39]. In this dissertation, we restrict our attention to <u>firm deadline</u> applications, and late transactions are therefore considered to be worthless and are discarded by the database system.

2.3.1 Priority Assignment

In an RTDBS, improving the response time of one transaction at the expense of another *can* result in a system performance improvement [39]. Imagine the case where two transactions arrive at the database system, one with a tight deadline and the other with a loose deadline. If the transactions share the system resources on an equal basis, the urgent transaction misses its deadline while the slack transaction completes in time. Alternatively, if the urgent transaction progresses at the expense of the slack transaction, it completes before its deadline. The slack transaction then, by virtue of its loose deadline, completes in time despite its retarded progress. From the above discussion, it is clear that providing differential service to transactions in an RTDBS can result in a performance improvement.

The priority assignment policy decides the service precedence among transactions to the database system resources. The goal of a priority assignment policy is to improve the system performance. Popular examples of such priority assignment policies include *Earliest Deadline* and *Least Slack*. The Earliest Deadline priority mapping [69] assigns higher priority to transactions with earlier deadlines. The Least Slack [4] policy gives maximum priority to the transaction that has least time to finish its remaining execution. However, this priority assignment requires apriori knowledge of the transaction's execution time and obtaining such knowledge maybe difficult in a general real-time transaction processing scenario. Therefore, in our work, we use the Earliest Deadline priority assignment policy in all our experiments.

2.4 Database Security

Having discussed both basic and real-time transaction processing, we now move on to discussing database security aspects. A database system typically has *objects* and *subjects*. An object is a data item, whereas a subject is a process that requests access to an object. For example when a process requests access to a file, the process is the subject and the file is the object. A subject can access an object in different access modes (e.g., read, write, append), governed by the system security policy. Database security deals with the measures, policies and mechanisms used to provide *availability*, *secrecy* and *integrity* in a database system [16]. The *availability* ensures that unauthorized subjects do not deny system services to genuine subjects. *Secrecy* deals with the prevention of unauthorized subjects from accessing information from or about the objects, and *integrity* deals with the prevention of the corruption of system objects either directly or indirectly by unprivileged subjects.

Ensuring security in databases translates to implementing a mechanism for monitoring and controlling the transaction data accesses. There are two types of data access control methods widely used for providing security:

Discretionary Access Control: Here the the data access control is left to the user's discretion. If a user wishes his/her data to be protected, the system offers a facility for doing so.

Mandatory Access Control: Here security is the responsibility of the system and it is required to prevent subjects (e.g., users, programs) from accessing objects (e.g., files, directories, devices, windows, sockets) that they are not trusted to handle.

The discretionary protection models are often simple and less secure whereas the mandatory protection models are more difficult to implement but guarantee certain levels of security. In this thesis, we focus on the secrecy aspects of SRTDBS that enforce Mandatory Access Control.

2.4.1 Mandatory Access Control (MAC)

Implementing mandatory protection requires assigning of sensitivity labels to all subjects and all objects in the system. Each object in the system has a *classification level* based on the security requirement. Similarly, each subject has a corresponding *clearance* level based on the degree to which it is trusted by the system. For convenience we assume, from here on, that the "level" of an object means its security classification level and the "level" of a subject means its security clearance level. Mandatory Access Controls use

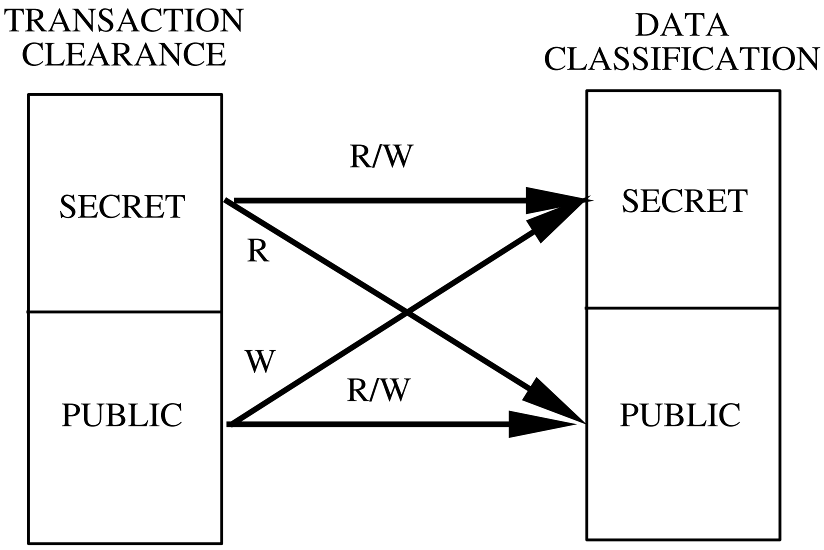


Figure 2.3: Bell-LaPadula access restrictions

sensitivity labels to determine who can access what information in the system. In a *closed system* (from a security view point), only explicitly authorized object accesses are allowed, whereas in an *open system*, accesses that are not explicitly forbidden are allowed. The studies reported in this thesis were conducted on a closed system.

2.4.2 Security Models

A security model provides a high level picture of the protection requirements in a system. In recent years, many security models have been proposed and some of these models enforce mandatory access control. In the remainder of this section, we discuss two important security models included in our studies, namely, *Bell–LaPadula* and *Restricted Write* – both these models support mandatory access control.

2.4.3 The Bell–LaPadula Model

The Bell–LaPadula model [59] imposes two restrictions on all data accesses:

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

- Star Property: A subject is allowed *write* access to an object only if the former's clearance is <u>identical to or lower than</u> the latter's classification.

The Bell–LaPadula conditions effectively enforce a "read below, write above" constraint on transaction data accesses (an example is shown in Figure 2.3) and thereby prevent flow of information from lower security levels to higher security levels.

2.4.4 The Restricted Write Model

In the Restricted Write model, the read access condition is identical to that of the Bell–LaPadula model, but the write access condition is further restricted to the following:

- A subject is allowed *write* access to an object only if the former's clearance is <u>identical to</u> the latter's classification.

Note that the above requirement disallows *blind writes* [11] by low security transactions to high security data. Therefore, the model prevents lower level subjects from corrupting the secret data.

2.4.5 Covert Channels

The security models discussed above are sufficient to prevent *direct* unauthorized access to secure data. They are not sufficient, however, to protect from "covert channels". A covert channel is an *indirect* means by which a high security clearance transactions can transfer information to a low security clearance transaction [57]. There are two types of covert channels [25]: *storage channels* and *timing channels*. Covert storage channels include all mediums that would allow the writing of a storage location by one transaction and the reading of it by another. Covert timing channels include all mediums that would allow the high security clearance transaction to signal information to another low security clearance transaction by modulating its own use of system resources in such a way that the change in response time observed by the low security clearance transaction would provide information. In this thesis, we focus our attention exclusively on <u>timing channels</u>.

Transaction Class	Time (ms)		
	05	10	20
Secret			
Public		Lock(d)	

Table 2.1: a) Secret sending 0 to Public

Transaction Class	Time (ms)		
	05	10	20
Secret	Lock(d)		UnLock(d)
Public		LockWait(d)	Lock(d)

Table 2.1: b) Secret sending 1 to Public

We explain the timing covert channels further with the help of Table 2.1a and 2.1b, where the request/grant sequence of a Secret class transaction and a Public class transaction with regard to an exclusive resource d is shown. In Table 2.1a, where the Public class transaction requests exclusive access to the resource d at time $t = 10ms$, the request is granted immediately since no transaction is holding the resource d. In contrast, in Table 2.1b, the same request is not immediately granted since the Secret transaction is already holding the resource from $t = 5ms$ onwards. The Public transaction, in this case, is made to wait until the Secret transaction releases the resource at $t = 20ms$. The above example illustrates that the *delay* experienced by the Public transaction for accessing the data item d can be *decided* by the Secret transaction (by choosing to access or not access the data item d). The presence or absence of the delay can be used as a "signaling" or encoding mechanism by the Secret transaction to pass on information to the Public transaction – in Table 2.1, a single bit of information (*one* or a *zero*) is transmitted in this manner.

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

Covert channels based on data can be prevented by providing *higher* priority to the low security transaction whenever a data conflict occurs between a low security transaction and a high security transaction. Taking this approach ensures that low security transactions do not "see" high security transactions and are therefore unable to distinguish between their presence or absence. This notion is formalized in [30] as *non-interference*. From a database system perspective, it translates to implementing transaction processing components that support the non-interference feature.

Some physical resources (e.g. the buffer pool), however, require careful handling due to the multitude of covert signaling mechanisms possible in such resources (see Section 2.5.4).

2.4.6 Covert Channel Security Requirements

The necessary and sufficient conditions for a secure interference-free scheduler are given in [53], and this framework is applicable to the secure firm real-time transaction processing environment. There are three requirements to be satisfied for a system to be covert channel secure:

Value Security: A system is Value Secure if the value of objects accessed by a subject are not changed by actions of subjects belonging to higher clearance levels.

Delay Security: A system is Delay Secure if the delay experienced by a subject is not affected by actions of subjects having higher clearance levels.

Recovery Security: A system is Recovery Secure if the occurrence of a transaction restart appears the same for a subject regardless of the presence or absence of higher level subjects in the system.

2.4.7 Channel Bandwidth

The maximum information leakage possible through a covert channel is often quantified in terms of the bandwidth of the channel. The higher the covert channel bandwidth, the less secure is the system. In this thesis we explore two scenarios: (1) *Full Security* environment, where the covert channel bandwidth is *zero*, and (2) *Orange Security* environment where the covert channel bandwidth is bounded to a maximum of *1 bit/second*.

Full Security: For many applications, security is an "all-or-nothing" issue, that is, it is a *correctness* criterion. In such "full-secure" applications, metrics such as the number of killed transactions or the fairness across security levels are secondary *performance* issues. The CC and buffer management protocols introduced in this thesis are designed to meet the full security requirement.

Orange Security: For some applications it is acceptable to have well-defined *bounded-bandwidth* covert channels in exchange for performance improvement. For example, the US military's security standards, which are defined in the so-called "Orange Book" [25] specify that covert channels with bandwidth of less than *one bit per second* are typically acceptable.

The underlying premise of such "orange-secure" applications is that covert channels based on such low bandwidths are acceptable because (a) these channels will take a long time to transfer useful information, (b) the time taken to transfer useful information is long enough to considerably increase the possibility of detection, (c) by the time useful information is transferred it may well be obsolete, and (d) the performance or implementation cost associated with eliminating the covert channel is much higher than the cost of any lost information.

Our transaction processing protocols are originally designed for meeting full security requirements. But we also show that in the "orange-secure" environment it is possible to tremendously improve the *fairness* performance of these protocols by including an adaptive feedback based admission control mechanism (this mechanism cannot be used in the full-secure environment since it introduces a covert channel).

2.5 Real-Time Database Security

We have discussed above, individually, database processing, real-time processing, and database security. We now shift our attention to the combination of these three areas, namely, secure real-time database systems.

Besides meeting the traditional database correctness requirements, an SRTDBS has to *simultaneously* satisfy two additional requirements, namely, provide security and minimize the number of missed transaction deadlines. Unfortunately, the mechanisms for achieving the individual goals often work at cross-purposes [36]. In an RTDBS , high priority is usually given to transactions with earlier deadlines in order to help their timely completion. On the other hand, in secure database systems, low security transactions are given high priority in order to avoid covert channels. Now consider the situation wherein a high security process submits a transaction with a tight deadline in a SRTDBS. In this case, it becomes difficult to assign a priority since assigning a high priority may cause a security violation whereas assigning a low priority may result in a missed deadline.

2.5.1 Secure Transaction Scheduling

An SRTDBS transaction scheduling algorithm (see Figure 2.4) includes the components of Service Eligibility, Priority Assignment, Resource Service, Concurrency Control and Recovery, just as in (unsecure) RTDBS [39]. The *Service Eligibility* component decides which transactions are allowed to execute in the database system. All other transactions are immediately (and permanently) discarded from the system. The *Priority Assignment* component assigns priorities to transactions that are eligible to execute in the system. The *Resource Service* component maps a scheduling discipline to each hardware resource. Finally, the *Concurrency Control* component maintains database consistency and resolves data conflicts. In addition to these four components, a new component that is specific to the secure environment is *Access Control* which enforces the security constraints for transaction data accesses.

A sample transaction scheduling algorithm, described in terms of the above components, is the following: { Feasible Only; Earliest Deadline; Priority PR (CPU), Priority LRU (Memory), Priority HOL (Disk); 2PL-HighPriority; Bell-LaPadula }. This algorithm implements the Feasible Only eligibility test [4] which discards late transactions and permits only feasible transactions to execute in the system. The Earliest Deadline priority mapping [69] assigns higher priority to transactions with earlier deadlines. The Priority Pre-emptive Resume [76] and Priority LRU [47] scheduling disciplines provide priority-based preemptive service at the processors and memory buffers, respectively, while Priority HOL [56] provides priority-based non-preemptive service at the disks. The 2PL-HighPriority CC protocol [4] maintains database consistency with a two-phase locking scheme and resolves data conflicts in favor of higher priority transactions. Finally, the Bell–LaPadula access restrictions prevent direct unauthorized access to secret data.

Note that the above system is still vulnerable to the indirect means of information transfer through covert channels. In this thesis we explore ways to make the components of the transaction processing system covert channel secure.

Priority Assignment

The priority assignment in a secure RTDBS should be such that the low security level transactions should not be able to feel the presence of higher security level transactions in the system. Therefore, whenever there is a data or resource conflict between a high security level transaction and a low security level transaction, the low security level transaction should be given priority over the high security level transaction. Note that with a secure priority assignment policy in place, security violations through covert channels are possible only at non-pre-emptable non-sharable resources.

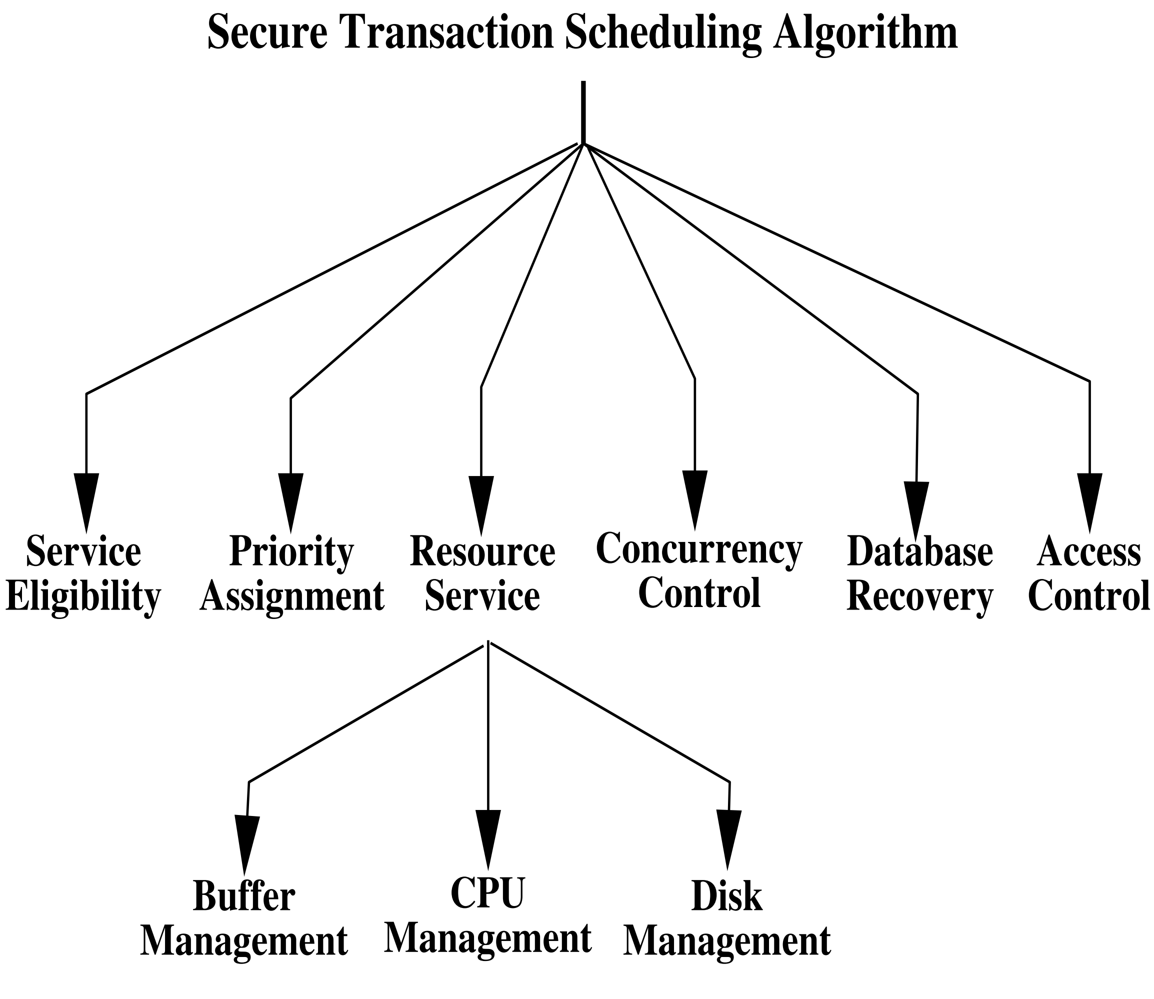


Figure 2.4: Components of Secure Transaction Scheduling Algorithm

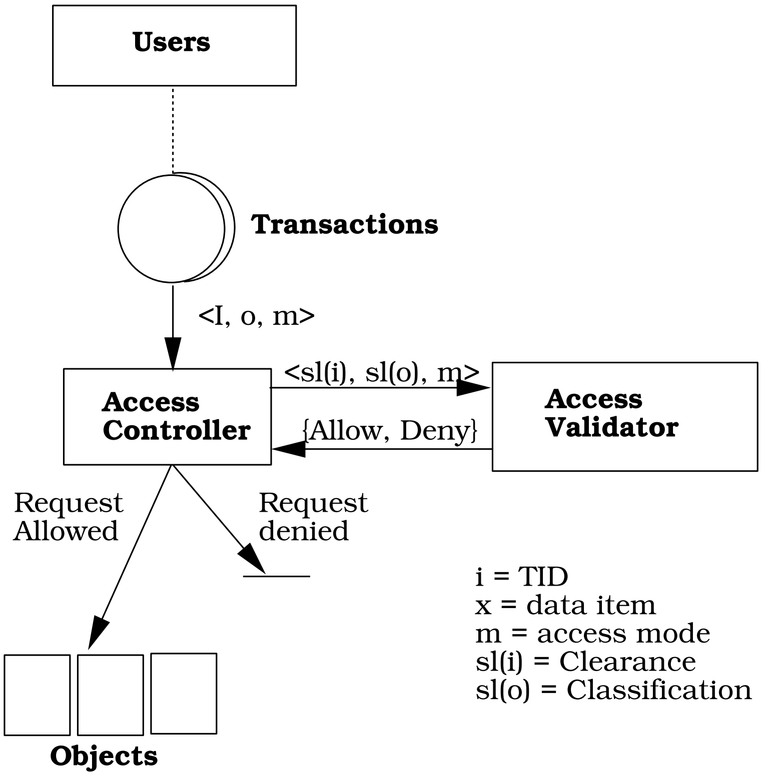


Figure 2.5: Access Control in Databases

2.5.2 Access Control

For security reasons, every transaction data access request must be validated and permitted by the access control unit. Upon receiving a validation request, the access control algorithm compares the security classification level of the transaction and the security clearance level of the object to which the access is requested. If the requested operation could be permitted as per the security policy, the request is allowed. On the other hand, if a request fails in the validation process, the requester (transaction) is killed and discarded from the system. Figure 2.5 shows the access control unit in a secure transaction processing system. The request received from the transaction is of the form $< I, o, m >$, where m is the operation the transaction I wishes to perform on the object o. The access controller consults the authorization database (not shown in the figure) and obtains the security clearance level of the transaction I (indicated as $sl(I)$) and the security classification level of the object o (indicted as $sl(o)$). The tuple $< sl(I), sl(o), m >$ is passed to the access validator for validation. Depending on the return value of the access validator, the access controller either allows or denies the request.

2.5.3 Concurrency Control

In an RTDBS, the decision on which transaction's progress is to be hindered is usually based on the transaction priority. A high priority transaction should not be either restarted or blocked by a transaction with a low priority. In conjunction with a **secure** priority assignment, it would seem at first glance that, in principle, any real-time CC protocol could be used in a secure RTDBS and that the actual choice of protocol would be based only on the relative performance of these protocols. However, *not all the previously proposed real-time CC algorithms are amenable to supporting security requirements*. For example, consider the 2PL Wait Promote algorithm proposed in [4]. This protocol, which is based on 2PL, incorporates a *priority inheritance* mechanism [84] wherein, whenever a requester blocks behind a lower-priority lock holder, the lock holder's priority is promoted to that of the requester. In other words, the lock holder *inherits* the priority of the lock requester. The basic idea here is to reduce the blocking time of high priority transactions by increasing the priority of conflicting low priority lock holders (these low priority transactions now execute faster and therefore release their locks earlier).

The Wait Promote approach is *not* suitable for SRTDBSs. This is because it permits the blocking of high priority transactions by low priority transactions which violates the requirement of *non-interference* between the transactions of different security levels (as described in Section 2.4.5, non-interference means that low security transactions should not be able to distinguish between the presence or absence of high security transactions).

To generalize the above observation, a real-time CC protocol that permits, to even a limited extent, high priority transactions to be adversely affected by low priority transactions, a phenomenon known as *priority inversion* in the real-time literature [84], cannot be used in a secure RTDBS. Apart from Wait Promote, other examples of real-time CC algorithms that fall into this category include 2PL-CR [4], 2PL-OS/BI [3] and WAIT-50 [40].

2.5.4 Resource Service

The main physical resources in a database management system are the processors, the memory buffers, and the disks. In a secure RTDBS, service should be provided in a manner that reflects transaction priorities as well as security objectives. One approach is to make the resources behave like *priority preemptive resume* servers [47]. In such servers, the highest-priority requester is always the one receiving service. This is achieved by preempting lower priority requesters from use of the resource. Hardware restrictions, however, prevent certain actions such as disk service, from being preemptable. This may open up covert channel possibilities and for fully addressing this issue, special hardware solutions may have to be developed.

In addition to the standard buffer management components of allocation, replacement and synchronization, described earlier in Section 2.2.2, an additional buffer manager component that arises specifically in the secure real-time domain is *pin preemption.* Consider the case, for example, where a low security transaction requests buffer space but the pages currently held by high security transactions which would normally be candidates for replacement (to prevent covert channels) cannot be forced out because they are all pinned. In this case, one option for the low security transaction is to wait for pages to become unpinned but this would immediately result in covert channel possibilities. The other option is to permit the replacement with the proviso that the high security transaction holding the original pin is immediately notified that the page it had assumed to be stable had actually been replaced – the high security transaction can then either abort or, preferably, redo the corrupted operation at a later time.

2.5.5 Database Recovery

The role of the secure recovery mechanism is to ensure transaction atomicity and durability without opening up the possibility of covert channels. Secure recovery has received less attention from researchers than concurrency control [10]. The main reason for this is that most recovery algorithms do not require trust (i.e., privilege). It is possible to maintain a separate recovery log for each active security level and the recovery at each level can

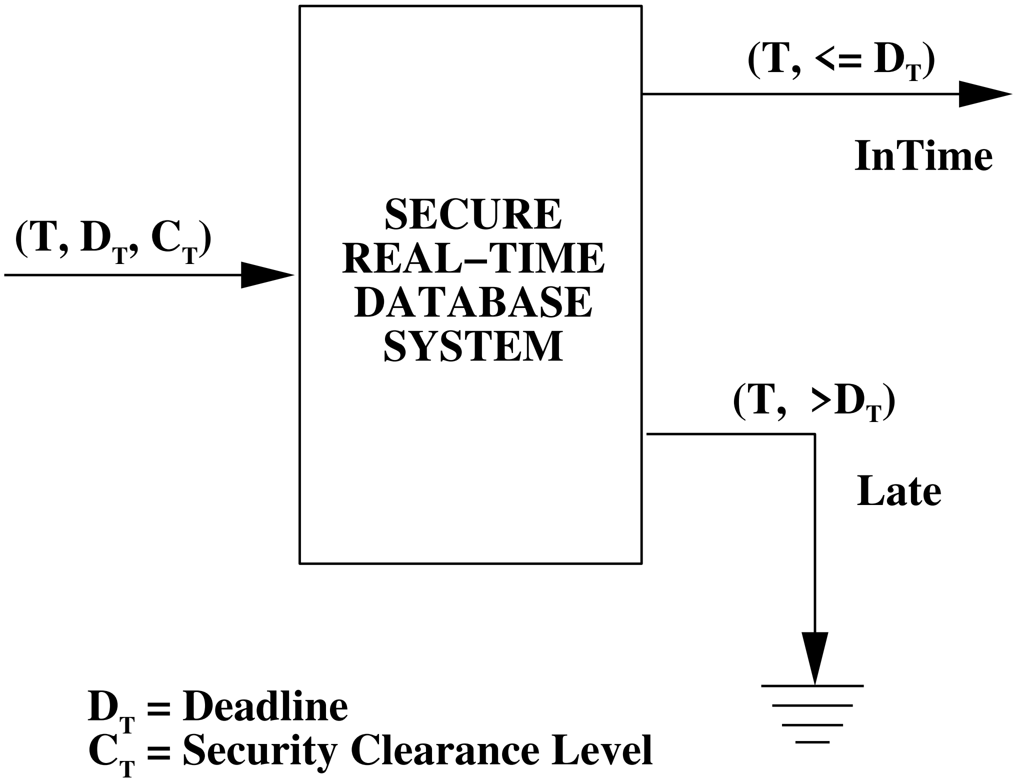


Figure 2.6: Black-Box View of the secure RTDBS

proceed independently. Therefore, the interference channels are easily eliminated.

However, ensuring the security of recovery protocols in *distributed* database systems is more complex as compared to centralized database systems. In distributed systems, even after a site decides to locally commit a cohort (the fragment of the global transaction executed at that site), that is, after it has entered the so-called *prepared state*, the local transaction manager has to retain some or all of its locks until the global decision on the transaction is available – this retention is fundamentally necessary to provide atomicity. In such a scenario, if a cohort requests a conflicting lock on the data element locked by a higher security level prepared cohort, the requester *has to* wait till the lock holder aborts or commits. However, note that this mechanism could be exploited by malicious high priority transactions as a signaling mechanism. Therefore, special mechanisms are required to ensure security of distributed commit protocols.

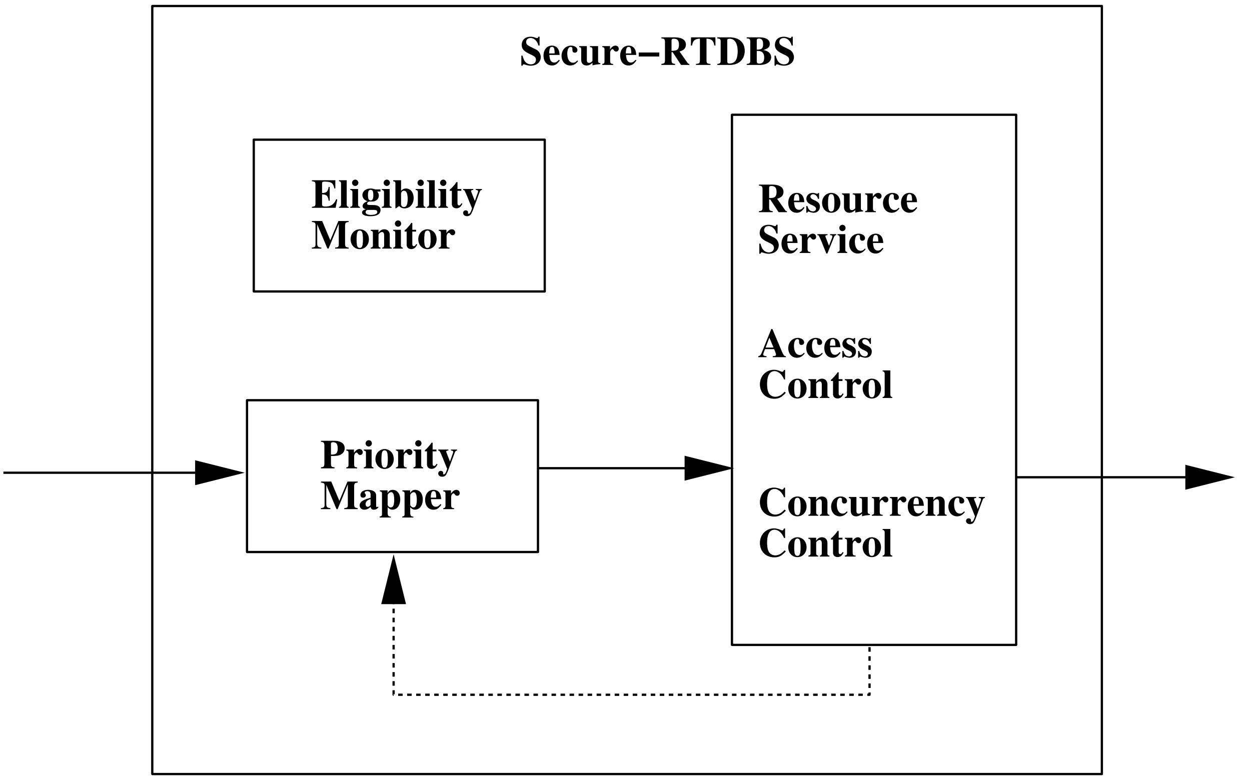


Figure 2.7: Transaction Scheduling Architecture

2.6 Secure RTDBS Architecture

A high-level "black-box" view of a SRTDBS is shown in Figure 2.6. In this model, the input process to the SRTDBS is an arrival stream of transactions with each transaction T having an associated arrival time A_T, deadline D_T, and security clearance level C_T. There are two transaction output streams from the secure RTDBS, *InTime* and *Late*. Transactions that complete at or before their deadlines join the InTime stream, while transactions that miss their deadlines join the Late stream. Transaction scheduling within the SRTDBS incorporates the architecture shown in Figure 2.7. The *eligibility monitor* tracks the eligibility status of all transactions executing in the system. A transaction that becomes ineligible is immediately removed from the system and discarded. The *priority mapper* unit assigns a priority vector P to each transaction on its arrival, and this priority is used by the hardware resource schedulers and the CC mechanism to distinguish among transactions. As mentioned earlier, we restrict our attention in this study to applications that have firm deadlines. This means that late transactions are immediately discarded and do not execute to completion. A second restriction is that the SRTDBS has no a-priori knowledge of either transaction processing requirements or transaction data accesses. This means that, from the secure RTDBS perspective, transactions are distinguished only by their arrival time, deadline and security classification level characteristics. It also means that transactions are detected to be late only when they *actually* miss their deadline since the secure RTDBS cannot estimate the remaining service requirements of a transaction (i.e. transactions remain eligible until their deadlines expire).

2.7 Summary of Framework of Study

In this chapter, we have described the detailed framework of our study. We now summarize the salient features of this framework: We concentrate on the mandatory secrecy of firm real-time applications operating on disk resident database systems that specify one copy serializability as the database correctness criterion. For simplicity, we assume that the security levels form a full order and that there is a one-to-one correspondence between data classification levels and transaction clearance levels. Applications can opt for either an absolute security guarantee or for an orange security guarantee with respect to timing-based covert channels. No apriori knowledge is available to the system about either the transaction workload or the transaction resource requirements.

For the above framework, we design and evaluate in this thesis efficient protocols for eliminating security violations that arise through the concurrency control manager and the buffer manager.

Chapter 3

Related Work

3.1 Introduction

Many real-time database applications arise in safety-critical installations and military systems where enforcing security is crucial to the success of the enterprise. Surprisingly, however, the issue of providing security in RTDBS has received comparatively little attention although realtime database research has been underway for close to a decade now (see [101] for a survey). Consequently, there have been only a handful of published papers on SRTDBS. In this chapter, a brief summary of relevant research on SRTDBS and related areas is presented. First, we explore the related research on secure transaction processing. We then shift our attention to real-time transaction processing. Finally, we consider secure real-time processing.

Another major issue arising in SRTDBS, apart from ensuring security and improving performance, is that of fairness – high security transactions usually form a disproportionately large fraction of the transactions with missed deadlines. The issue of providing fairness in RTDBSs has been addressed earlier, but in a different context. We briefly review the relevant fairness research. Finally, we conclude the chapter with a survey of major implementation projects and commercial database products.

3.2 Secure Transaction Processing

In the last chapter we discussed different types of access control mechanisms commonly used in database systems (see Section 2.4). The mandatory access control is ideal for the environments where the users and objects can be classified and the application demands a high level of security. Some of the works we discuss here may not provide complete mandatory access control. However, they find relevance here due to their significance as some of the interesting research attempts that lead to the evolution of the modern MAC models.

A *need-to-know* policy for limiting information flow in process execution in the operating system environment was suggested in [102]. In their scheme, a process is allowed to copy information from one object to another only if the set of users allowed to read the second object is a subset of the users allowed to read the first object. Several other need-to-know policies were later proposed. However, many of these policies could not control the dissemination of information fully. An information flow based access control was proposed in [66] and this policy allows the owner of the file to retain control of dissemination of the information obtained in the file. The approach consists of associating a data object with an access control list, which is propagated through subject and object labels, to all objects into which the contents of the object may flow. Similarly, in the model proposed by [98], each object is associated with two protection attributes. The current access attribute describes what can be done by whom on the object. Potential access attributes provide a means of propagating possible access restrictions to the information, once it has been released. A lattice model of flow control is proposed in [23] to formulate the requirements needed for secure information flows. In this model the flow relations are specified by organizing information into classes and indicating secure flows among these classes.

The Bell-LaPadula model described in Chapter 2 is one of the early models that support mandatory access control which aims at achieving secrecy by preventing unauthorized access to the secure data. This model, however, is not sufficient to protect the information integrity (as per the definition in Section 2.4), since the low level subjects are freely allowed to pass on information to the higher levels.

A security model that can ensure information integrity called Biba model [13] was later suggested – this model is specified in terms of subjects and objects. Each subject in the system is assigned an integrity level reflecting its trustworthiness for inserting, modifying or deleting information. Each object in the system has an integrity level that reflects both the degree of trust that can be placed in the information stored in the object and the potential damage that could result from the unauthorized modification of information. Biba model contains both mandatory and discretionary access policies to protect the integrity of information.

The Dion model [26] proposes a mandatory policy that protects the secrecy as well as the integrity of data. This model basically combines the principles for controlling secrecy of the Bell-LaPadula model with the principles of controlling integrity of the Biba model. Dion is an *object-to-object* model; in other words, the data flow is allowed only between objects, never between subjects and objects.

In the last ten years, transaction processing in secure DBMS has been receiving attention from academia as well as industry [10]. Various models for secure transaction processing have been proposed. The Sea View (SEcure dAta VIEW) model [24] is a security model for the protection of relational database systems. The model governs access to the data stored in the database on the basis of mandatory as well as discretionary policies. Another model for the application of mandatory policies in relational database systems has been proposed in [46]. The model extends the standard relational model to the consideration of security classifications and provides a formalization of such an extended model. The model is based on the security classifications introduced in the Bell-LaPadula model. Another model which considers the application of mandatory policies in relational databases has been proposed in [88, 94]. This model treats a secure database as a set of ordinary relational databases.

The implementation of secure database models, including those mentioned above, can be classified into two types : *replicated* and *kernelized*. The idea behind the replicated models is to construct a secure database from several single-level DBMSs. The DBMS of a particular level contains all the data items that a subject at that level can access. While the replicated models are simple and able to reuse the DBMSs in their entirety, the major disadvantages with them are the inflexible and inefficient use of resources. Kernelized models, where only one copy of each data page is maintained, provide a way to improve the situation, but the basic problem arising in such systems is the synchronization of readers and writers. In this thesis, we concentrate on the development of transaction processing protocols for kernelized models.

3.2.1 Concurrency Control

Several secure CC mechanisms for replicated models have been proposed [17, 6, 45]. However, in replicated models, updating replicas incurs significant overheads with no clear benefits. Also, the resources are allocated among the DBMSs when the system is configured and therefore cannot adapt to the changes at a later point of time.

A multi-version time stamp ordering (MVTO) protocol for kernelized models was suggested in [52]. The algorithm assigns timestamps such that high-level readers are pushed behind writers in the serialization order. However, their protocol cannot prevent a high level transaction from reading arbitrarily stale data. Also, the number of versions required by the protocol (for ensuring liveness) does not have an upper bound, and the version pool being physically separate from the primary version pool can reduce the overall physical clustering and therefore affect the system performance. A snapshot algorithm was proposed in [7] to overcome the disadvantages of MVTO protocols. As with schedulers based on MVTO, even in snapshot algorithms, there is a basic conflict between long-lived high security transactions and the freshness of the data that they read [10].

Jajodia et al study secure locking protocols in [49]. In particular, they modify strict 2PL to a covert-channel-free 2PL, called Secure 2PL. In their protocol, a high security transaction must release its lock on a data item when a low security transaction requests an exclusive lock on the same data item. Since a low security transaction is never blocked or restarted by a high security transaction, this protocol satisfies the non-interference requirement.

A secure locking-based protocol, also called Secure 2PL (S2PL), but based on a completely different approach, was proposed in [89]. The basic principle behind S2PL is to try to simulate the execution of conventional 2PL *without* blocking the actions of low security transactions by high security transactions. This is accomplished by providing a new lock type called *virtual lock*, which is used by low security transactions that develop conflicts with high security transactions. The actions corresponding to setting of virtual locks are implemented on *private versions* of the data item (similar to optimistic CC). When the conflicting high security transaction commits and releases the data item, the virtual lock of the low security transaction is upgraded to a real lock and the operation is performed on the original data item. To complete this scheme, an additional lock type called *dependent virtual lock* is required apart from maintaining, for each executing transaction T_i, lists of the active transactions that precede or follow T_i in the serialization order.

It has been argued that the serializability requirement is overly restrictive for secure databases [65]. Therefore, alternative and weaker notions of database correctness for secure databases, such as *levelwise serializability*, *one-item read serializability*, and *pairwise serializability*, have been proposed in [44].

Another solution that has been proposed is to allow users to read and write information at multiple classification levels by decomposing the original transaction into multiple subtransactions, each of which is assigned a single classification level, and all actions performed by subtransactions obey the Bell-LaPadula properties. However, even in such a scenario, it is impossible to simultaneously guarantee both transaction atomicity and absence of covert channels [14].

3.2.2 Buffer Management

Buffer management in traditional database systems has been extensively studied (see [27, 35] for surveys). In comparison, however, very little work has been done with regard to secure buffer management. The only prior research we are aware of on secure buffer management is the recent study by Warner et al [104]. A number of design alternatives for secure buffer allocation, replacement and synchronization were explored in this study. In particular, they *statically* divided the buffer pool among the various security levels,[1] but also presented a *dynamic* scheme called "slot stealing" wherein buffers, currently under-utilized at the low security level, could be borrowed by high security transactions. Later, if needed, these borrowed slots could be reclaimed by the low security transactions. They also considered replication-based schemes wherein multiple copies of the same disk page could be present in the buffer pools of various security levels. Finally, the performance of these policies was evaluated on a simulated model of a buffer manager.

While their dynamic allocation scheme is an interesting approach, it suffers from two problems: First, it is only *partially dynamic*, since low security transactions are not allowed to utilize the currently unused buffer slots of high security transactions. Therefore, resource wastage could still result. Second, it must be ensured that the number of unpinned clean buffer slots in the high security buffer pool must *at all times* be as large as the number of slots borrowed from the low security pool. This is necessary to support the immediate return of slots reclaimed by the low security transactions and thereby to prevent covert channels. The utility of the slot stealing is diminished by this constraint since it places restrictions on the high security accesses.

A simulation-based performance evaluation of their various policies was also presented in [104] but this evaluation considered the buffer management component in isolation, not as part of an entire secure system. It is possible that the relative performance behavior of the policies may be impacted when integrated into a complete system.

3.2.3 Recovery

The role of the recovery manager is to ensure transaction atomicity and durability. Secure recovery processing has received less attention as compared to secure CC in the literature. This is mainly due to the fact that the recovery algorithms need not be "trusted", since the recovery can be performed on a per security level basis, and it is possible to maintain a separate recovery log for each active security level [10]. A drawback of this

[1] A scheme where the static buffer allocation is periodically reviewed and altered is also presented, but this scheme can result in covert channels.

scheme, however, is that although the process of recovery can proceed at each security level independently of others, the new transactions cannot begin execution until recovery is completed at all dominated security levels. An Undo/No Redo protocol is proposed in [50] for solving this issue, but this strategy requires a separate disk be maintained for each security level.

As mentioned in Section 2.5.5, recovery in distributed systems is more complex than in centralized systems. A protocol to address this problem, called Secure Early Prepare (SEP), was proposed in [8] which supports lock pre-emption by low security transactions at any stage of the high security transction's execution, including the prepared state. This is achieved, however, at the expense of introducing an additional *confirmation phase* in the protocol. An improved version of SEP called ASEP was proposed in [79] – ASEP supports language primitives to provide features like rollback, forward recovery, etc.

3.3 Real-Time Processing

The problem of scheduling real-time transactions was first addressed, almost exactly a decade ago, in [4]. They evaluated a variety of real-time scheduling algorithms that considered different combinations of eligibility policies, priority assignments, resource scheduling disciplines, and locking-based CC protocols. Since that time, there has been a significant body of research on various aspects of real-time transaction processing (see [101] for a survey), which we cover in the remainder of this section.

3.3.1 Concurrency Control

The 2PL High Priority (2PL-HP) scheme, a prioritized modification of 2PL [28], was suggested in [4] for CC in RTDBS. Subsequently, the prioritized variants of optimistic CC (OPT) [51] such as Sacrifice and Wait [39], hybrid algorithms [68], etc. were also suggested. These algorithms were primarily designed to minimize the number of missed transaction deadlines and were evaluated on this basis [4, 95].

A phenomenon peculiar to priority-driven systems is *priority inversion* [84]. This refers to the situation where a high-priority task is blocked by a low-priority task, and typically occurs from synchronized access to shared resources [39]. This situation can arise in an RTDBS, for example, if a conventional locking protocol is used to control data access – a high priority transaction can be "locked out" by a low priority transaction that has accessed the object earlier. One solution to this problem is *priority inheritance* [84], where the low priority transaction inherits the priority of the high priority waiter. Another solution is *priority abort* [4] where the low priority transaction is aborted and the lock is granted to the high priority transaction, as in the 2PL-HP algorithm mentioned above. A conditional priority inheritance scheme that attempts to capitalize on the advantages of each of the above two schemes is described in [42].

3.3.2 Buffer Management

A scheme for pre-fetching data pages in an RTDBS was presented in [105], but its applicability is limited since it assumes complete apriori knowledge of the workload on which static transaction pre-analysis techniques are applied. In [15], buffer management policies for prioritized transaction processing were introduced. In particular, prioritized versions of the popular LRU [27] and DBMIN [18] policies were presented and evaluated in conjunction with a prioritized allocation policy that involved suspension of low priority transactions to create buffer space for higher priority transactions. These policies were designed for systems with *fixed* transaction priority classes and are therefore not applicable directly to the real-time environment where priority is usually a dynamic assignment. To address this lacuna, real-time versions of the allocation and replacement polices of [15] were developed and evaluated in [42] and in [55]. The results of these studies were inconclusive, however, since they arrived at opposing conclusions regarding the utility of adding real-time information to buffer management.

3.3.3 Recovery

Research in real-time recovery processing has received comparatively less attention than concurrency control. The only papers that we are aware of in this area are [22, 93, 106, 32].

A *centralized timed* 2PC protocol is described in [22] that guarantees that the fate of a transaction (commit or abort) is known to all the participants of the transaction before the expiry of the deadline, when there are no processor, communication, or clock faults. In case of faults, however, it is not possible to provide such guarantees, and an *exception state* is allowed which indicates the violation of the deadline. Further, the protocol assumes that it is possible for the DRTDBS to guarantee allocation of resources for a duration of time within a given time interval. Moreover, the protocol is predicated upon the knowledge of worst-case communication delays.

The [93] and [106] papers are based on a common theme of allowing individual sites to *unilaterally* commit – the idea is that unilateral commitment results in greater timeliness of actions. If it is later found that the decision is not consistent globally, "compensation" transactions are used to rectify the errors.

A variety of distributed commit protocols were recently evaluated in [32] and they showed, using a simulation-based study, that distributed commit processing can have considerably more effect than distributed data processing on the real-time performance. This highlights the need for developing commit protocols tuned to the real-time domain. They also proposed a new commit protocol, called OPT, that allows prepared cohorts to "lend" their uncommitted data to other executing transactions, and showed that this protocol can provide considerably improved performance as compared to the traditional commit protocols. OPT also features several other optimizations that are tuned to the real-time domain.

3.3.4 Fairness

It has been found that the Earliest Deadline priority assignment policy, described in Section 2.3.1, discriminates against long transactions in that they form a disproportionately large fraction of the transactions with missed deadlines. To solve this problem, a dynamic priority assignment scheme called *Adaptive Earliest Virtual Deadline* was described in [73] – this scheme ensures fairness by assigning a sequence of "virtual deadlines" to transactions, with the relative distances between virtual deadlines being tuned to match the estimated length of the transaction – longer transactions have more frequent virtual deadlines.

There has also been fairness-related work reported in the context of priority memory management for multiclass workloads [74]. Their work employs admission control as a fairness mechanism, and the desired per class multi-programming level at any moment is calculated, based on the system-wide performance measures.

Fairness from a CC perspective was addressed in [43]. Here, a two-class workload was used to show that, similar to conventional DBMS, real-time optimistic CC also discriminates against long transactions. Their solution to the problem was to assign a weighting factor to the priority of each class and with a performance study showed that the bias could be overcome by suitable tuning of these weights.

3.4 Secure Real-Time Processing

As discussed above, the issues of designing transaction processing protocols for the (unsecure) real-time databases and for secure (non-realtime) databases have been addressed to some extent. Little attention appears to have been given, however, to developing transaction processing components for SRTDBS.

3.4.1 Concurrency Control

The only work that we are aware of in the secure real-time concurrency control area is a series of papers by Son et al [21, 72, 90, 91, 92]. In particular, a concurrency control protocol that attempts to *balance* the dual requirements of security and timeliness is presented in [21, 90, 91]. In their scheme, transactions *dynamically* choose between 2PL-HP [4] and S2PL [89], the previously described (unsecure) real-time and secure (non-real-time) versions of 2PL, respectively. The goal of the protocol is to *tradeoff* security for real-time performance with the tradeoff depending on the state of the system and the application's requirements.

Since the protocol is based on trading off security for real-time performance improvements, they suggest a way to measure the degree to which the security is being satisfied by the system [21]. Their analysis is based on information theoritic concepts and defines the capacity of the covert channel as the maximum amount of information that can be communicated through the covert channel, measured in bits per second. Capacity is specified in terms of two quantities: a parameter that quantifies the interference from other transactions or in other words the "noise" generated due to other transactions (q_1), and the probability that a secure choice is made when a data conflict occurs (q_2). The value of q_1 depends on the workload conditions and the duration of the system clock tick, and as they point out, it is difficult to control. Therefore, they consider only tuning the parameter q_2. The system will swing from fully secure to unsecure for different values of q_2 ranging from 0 to 1.

For a desired value of covert channel capacity, *given* the current q_1, the value of q_2 is calculated. Then the algorithm observes the resulting missed deadlines percentage over an interval, and if the kill percentage exceeds a threshold, the database administrator is intimated for readjusting the value of the channel capacity and/or desired kill percentage. If the current killpercentage is far below the threshold value, the parameter q_2 is decreased towards zero, such that the security is improved. This protocol, while a welcome first step, suffers from several problems:

1. Imagine the case when a high priority secret transaction requests a data item. Assume that the algorithm chooses the real-time option, and grants the lock to the requester aborting the low security holder. Now if a new low security transaction with a slack deadline makes a conflicting request on the same data item, and the protocol this time decides to follow the secure option, the transaction that was assisted earlier based on deadline considerations, now gets hindered on security considerations. Also, since the protocol takes only data conflicts into account, it cannot control the conflicts at other resources. Therefore again, a transaction that was assisted to make its deadline at the data may be hindered at the resources.

2. The protocol provides the database administrator the facility for "tuning" the database by readjusting the channel capacity and desired killpercentage. This facil-

ity, however, may not be very helpful under a real-time workload.

3. For applications where full security is a fundamental requirement, the protocol reduces to plain S2PL, which does not have any real-time features.

In [72], a CC protocol that ensures *both* security and timeliness is proposed. For this scheme, however, the RTDBS is required to maintain *two* copies of each data item. Further, transactions are required to obtain *all* their data locks *before* starting the execution (i.e., strict static locking). These requirements limit the applicability of the protocol.

3.4.2 Buffer Management

We are unaware of any prior research work on buffer management in SRTDBS. We take a first step to address this lacuna in this thesis.

3.4.3 Recovery

To the best of our knowledge, specialized recovery solutions are yet to be developed for SRTDBS. We plan to address this issue in our future work.

3.4.4 Fairness

A fundamental problem with schemes for ensuring security in SRTDBS is that of *fairness* – transactions belonging to higher security classes usually form a disproportionately large fraction of the transactions that miss their deadlines. To the best of our knowledge, this problem has not been considered before for SRTDBS and, in this thesis, we present an algorithm that attempts to provide a solution.

3.5 Our Research

Our research is directed towards the development of efficient CC and buffer management protocols tailored for secure firm deadline real-time transaction processing environments.

We also propose and evaluate an admission control protocol to ensure fairness across security classification levels.

3.5.1 Concurrency Control

In our work, we consider database environments where all data items are single-copy and transactions acquire data locks dynamically. Further, unlike [89], which is primarily addressed towards “soft-deadline” applications, that is, real-time applications in which there is value to completing tasks even *after* their deadlines have expired, we have concentrated on firm-deadline applications. The type of deadline has a significant impact on both the performance evaluation model and on the interpretation of the results, as observed earlier for (unsecure) real-time transaction CC [4, 40].

3.5.2 Buffer Management

We have conducted a detailed study for designing buffer managers that address the design challenges unique to SRTDBS. Our work differs from the earlier work of Warner et al [104] in that, apart from addressing real-time applications, we consider (a) fully dynamic and symmetric policies, (b) the buffer manager's performance role in the context of an <u>entire</u> system.

3.5.3 Fairness

In this thesis, we don't address the fairness issues arising out of the transaction type (e.g., query or update) or size. Our work concentrates on the fairness with respect to missed deadlines across security classes. Simultaneously achieving full security and fairness appears to be fundamentally impossible unless static resource reservation policies are implemented, which could lead to poor system utilization. We have therefore taken the approach of developing dynamic fairness mechanisms for *orange secure* applications, and show that with careful design, it is possible to develop protocols that deliver almost ideal fairness with minimal impact on the real-time performance.

3.6 Products

Having discussed the research literature, we move on now to reviewing some products that are commercially available. Examples of commercial secure database products [10] include Sybase Secure SQL server, Trusted Oracle and Informix-Online/Secure. For CC, Sybase uses 2PL and therefore is not covert channel secure. Trusted Oracle uses a combination of locking and multiversioning techniques for CC. Although this algorithm is secure, it does not produce one copy serializable histories. Informix uses an approach by which a low-level transaction can acquire a write lock on a low data item, even when a high level transaction holds read lock on this data item. The low level transaction is, therefore, not delayed by the high level transaction. However, the high level transaction receives a warning that the lock is broken. Despite the broken lock the high level transaction could still be committed.

Though there are a few real-time relational databases available in the market (e.g., Polyhedra [77]), they mainly provide discretionary security. We are unaware of any real-time database product that can fullfill the mandatory security requirement. However, the feasibility of a secure real-time operating system that enforces mandatory security has been demonstrated by SRI International [25].

Chapter 4

SRTDBS Model

We developed a simulation model of a shared-memory multiprocessor database management system using primarily Simscript [60, 83], a discrete-event simulation language. Some routines were, however, implemented in C [54] for efficiency reasons. This simulator is the main tool employed for our performance studies of secure realtime transaction processing protocols, detailed in the next few chapters. In this chapter, we describe the model and the performance metrics.

4.1 Model

Our model is loosely based on the single-site realtime database model of [39], and is shown in Figure 4.1. There are six components in our model and four of those components (*Database*, *Transaction Manager*, *Resource Manager* and *Concurrency Control Manager*) represent the database services. The other two modules, *Source* and *Sink*, capture the application utilizing the database services. The functionality of the individual components is as follows:

1. Database models the data and its organization
2. Source generates the transaction workload of the system
3. Transaction Manager models the execution of transactions

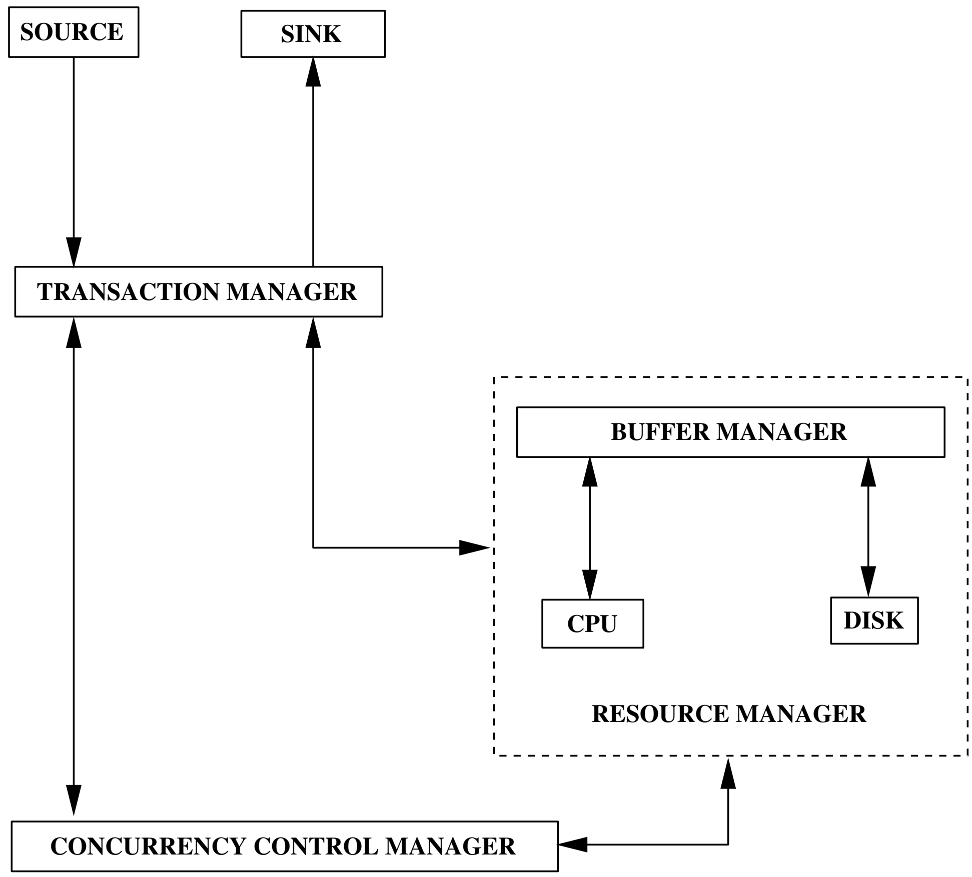


Figure 4.1: SRTDBS model

4. Resource Manager models the hardware resources

5. Concurrency Control (CC) Manager controls access to shared data

6. Sink collects the statistics on exiting transactions

The interface between modules primarily consist of service requests and completion replies. One may note that our model does not explicitly have any secure domain specific components. We don't model the access controller (see Chapter 2) explicitly because of two reasons: First, we are interested in the performance of the transactions that perform *legal* operations in the system. Second, we don't foresee any difference in the relative performance of the protocols profiled with the addition of the access controller. We also feel that the performance overhead due to the access controller is negligible. Although there are no security specific components in our model, the modeling of each component in the secure domain is quite different from their modeling in the non-secure RTDBS domain. In the reminder of this chapter, we describe the parameters and internals of each of the model components.

4.1.1 Database

The database is modeled as a collection of pages, and all database operations are modeled at the page level of granularity. The parameter $DBSize$ specifies the number of pages in the database. The data pages are uniformly randomly distributed across all of the disks. The database is equally partitioned into $ClassLevels$ security classification levels (for example, if the database has 1000 pages and the number of classifications is 5, pages 1 through 200 belong to level 1, pages 201 through 400 belong to level 2, and so on).

4.1.2 Source

The Source component represents the real-time application that utilizes the database services. It generates a stream of transactions that comprise the workload of the SRTDBS. Transactions are generated in a Poisson stream with rate $ArrivalRate$, and each transaction has an associated security clearance level and a firm completion deadline. Table 4.1

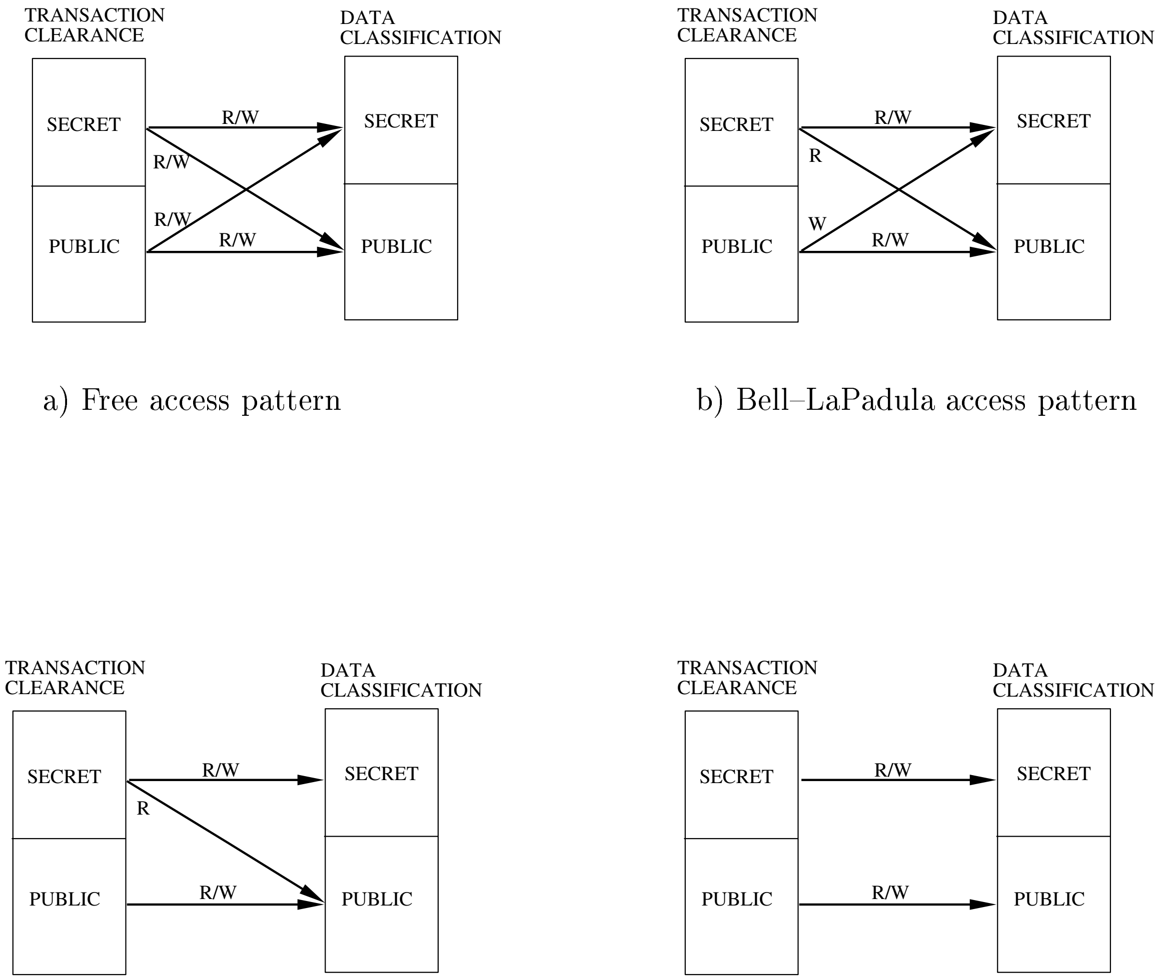
Table 4.1: Simulation Model Parameters

$DBSize$	Number of pages in the database
$ClassLevels$	Number of Classification Levels
$ArrivalRate$	Transaction arrival rate
$ClearLevels$	Number of Clearance Levels
$LowSlackFactor$	Lowest Slack Factor in Deadline assignment
$HighSlackFactor$	Highest Slack Factor in Deadline assignment
$TransSize$	Average transaction size (in pages)
$WriteProb$	Page write probability
$NumCPUs$	Number of processors
$NumDisks$	Number of disks
$NumBuffers$	Number of buffers
$PageCPU$	CPU time for processing a data page
$PageDisk$	Disk service time for a data page
$MinPin$	Minimum buffer pin time
$MaxPin$	Maximum buffer pin time
$CReqCPU$	Concurrency Control Overhead
$NumGPageSets$	Number of Global pagesets
$SizeGPageSet$	Size of Global pageset
$GRefCount$	Number of references from current global pageset
$InterLoc$	Inter-transaction locality Factor
$IntraLoc$	Intra-transaction locality Factor
$LocalProb$	Local Pageset Selection Probability

summarizes the key workload parameters. Each transaction consists of a sequence of pages to be read and a set of pages which are to be written. The mechanisms for determining a transaction's security clearance level, generating its pageset and assigning its deadline are described below:

Transaction Security Clearance Level Assignment

When a transaction is generated by the Source module a clearance level is assigned to it. For simplicity, we assume that the clearance level remains constant throughout the life of the transaction inside the system. The probability that a transaction belongs to the security level i is decided by the parameter λ_i, the levelwise transaction arrival rate, and is given by: $p(i) = \frac{\lambda_i}{\Sigma_1^n \lambda_i}$, where n is the total number of security clearance levels in the system.



a) Free access pattern

b) Bell–LaPadula access pattern

c) Restricted access pattern

d) Partitioned access pattern

Figure 4.2: Data access patterns

Transaction Page Assignment

A transaction consists of a sequence of page read and page write accesses. The number of pages accessed by a transaction varies uniformly between half and one-and-a-half times the value of $TransSize$. The number of pages accessed by the transaction varies uniformly between 0.5 * $TransSize$ and 1.5 * $TransSize$ pages. The $WriteProb$ parameter determines the probability that a transaction operation is a write. Each transaction can only access data from a specific segment of the database. Note that since concurrency control is implemented at the page granularity, the page size is not modeled in our experiments.

Table 4.2: Pattern summary

Pattern	Intra-Class	Inter-Class
Free	$\{R, W\}$	$\{R \updownarrow, W \updownarrow\}$
Bell–LaPadula	$\{R, W\}$	$\{R \downarrow, W \uparrow\}$
Restricted	$\{R, W\}$	$\{R \downarrow\}$
Partitioned	$\{R, W\}$	$\{\}$

The segment of the database to which a transaction is given access depends on the security clearance level of the transaction, the operation it wishes to perform and the type of data access pattern the workload generator generates. Our workload generator can generate four different data access patterns that place varying degrees of restrictions on data access operations – examples of these patterns are shown in Figure 4.2a–d for a two security level system. In these figures, the symbols R and W represent read and write operations, respectively.

Figure 4.2a shows the data access pattern that is normally seen in un-secure databases. All transactions can access all data elements without any restrictions. We refer to this pattern as *Free*, since transactions are free to access the entire database. Figure 4.2b shows the *Bell–LaPadula* data access pattern that implements the Bell-LaPadula security constraints. Figure 4.2c shows the *Restricted* data access pattern generated for the systems that follow the Restricted Write model described in Section 2.4.4. Finally, Figure 4.2d shows the *Partitioned* pattern, where interclass operations are completely absent. A system that has this data access pattern is equivalent to a collection of single level database systems.

Table 4.2 summarizes the above-mentioned data access patterns. The Intra-Class column specifies the operations allowed between a subject and object of identical security levels. The Inter-Class column specifies the operations permitted between a subject and object of different security levels. A $\uparrow$ indicates that a subject can perform the operation on objects of classification levels higher than its own clearance level. Similarly, a $\downarrow$ indicates that a subject can perform the operation on objects of classification levels lower than its own clearance level. A $\updownarrow$ indicates that the operation could be performed on objects belonging to either higher classification levels or lower classification levels.

Access Locality

In the *no locality* case, that is, when there is no explicit locality present in the workload, the page requests are generated by uniformly randomly sampling (without replacement) from the database over the region of access. The generation of inter-transaction and intra-transaction reference locality is handled in a manner similar to that used in [104]. In this scheme, two types of pagesets are created, *global pagesets* and *local pagesets*. The inter-transaction and intra-transaction locality is associated with the global pagesets and local pagesets respectively.

Global pagesets are generated by sampling (with replacement) from the database using identical (truncated) normal distributions with variance $1/InterLoc^2$ – only the page location of the centrepoint of the distribution is a function of the individual global pageset. The $NumGPageSet$ parameter specifies the number of global pagesets generated and the size of each pageset is given by the $SizeGPageSet$ parameter. At the outset, one of the global pagesets is designated as the *current* global pageset and subsequent page references are generated from this pageset. After $GRefCount$ number of references have been generated from this pageset, another pageset is uniformly randomly chosen from the remaining pagesets to become the current pageset.

The pageset for a transaction T is created in the following manner: Initially a local pageset is created. This local pageset is obtained by uniformly randomly choosing pages from the current global pageset and the number of pages chosen is given by the formula $LocalPageSetSize_T = (1 - IntraLoc) * TransSize_T$. After this, the transaction's pageset is created by successively choosing pages randomly from within the local pageset or from within the current global pageset. The probability that a given page access is chosen from the local pageset is specified by the $LocalProb$ parameter.

Finally, restart locality is modeled by ensuring that a transaction that is restarted due to a data conflict has the same clearance level and makes the same data accesses as its original incarnation.

Transaction Deadline Assignment

We used two different transaction deadline assignment policies (similar to [40]) in our studies. Each experiment uses only one formula for assigning deadlines. We will use D_T, A_T and R_T to denote the deadline, arrival time and resource time of transaction T, respectively. The resource time of a transaction is the total service time at the resources that it requires to commit. In other words, it is the response time of the transaction if it were to execute in isolation in the system. Using the above notation, we define the term *slack ratio* as $\frac{D_T - A_T}{R_T}$. The physical interpretation of this ratio is that it is the number of completion opportunities provided to the transaction by the application. (A slack ratio of less than 1 implies that it is impossible for the transaction to complete before its deadline.) The deadline formulae employed in our studies primarily differ in the slack ratios that they assign to transactions.

The first deadline assignment that we use is

DF1: $D_T = A_T + SlackFactor * R_T$

The *SlackFactor* parameter controls the tightness/slackness of deadlines. Note that with this formula, the deadline of a transaction is *linearly* correlated with its execution time.

The second deadline assignment that we use is

DF2: $D_T = A_T + Uniform(LowSlackFactor, HighSlackFactor) * R_{max}$

The resource time used in this assignment, R_{max}, is the resource time of the *largest* workload transaction that could be generated by the Source. The largest resource time is evaluated based on the maximum possible transaction size. Transaction slack factors are uniformly chosen over the range set by $LowSlackFactor$ and $HighSlackFactor$. With this assignment, transaction slack ratios are spread over a range of values, based on the ratio of R_{max} to the R_T's and the spread in slack factors. Note that with this assignment, short transactions (i.e. those with smaller resource times) tend to have greater slack ratios than long transactions. This formula also makes the deadline of a transaction *independent* of its execution time.

The $LowSlackFactor$ and $HighSlackFactor$ workload parameters set the slack factors to be used in each of the deadline formulas. (For DF1, both the parameters are set to the same number and SlackFactor takes on this value.) An important point to note here is that while the workload generator utilizes information about transaction resource requirements in assigning deadlines, the SRTDBS itself has no access to such information.

4.1.3 Transaction Manager

Transactions generated by the Source are delivered to the Transaction Manager of the SRTDBS. The Transaction Manager controls the execution of transactions. It implements the Service Eligibility and Priority Assignment components of the SRTDBS transaction scheduling algorithm. Upon arrival, transactions receive a priority from the priority mapper unit, which is embedded in the transaction manager. The priority mapper assigns priorities according to the Priority Assignment component.

The transaction manager executes each transaction until it either completes or is deemed ineligible by the Service Eligibility test. Transactions that complete are marked as such and forwarded to the Sink module. A transaction that becomes ineligible is immediately "killed"; killing a transaction consists of aborting its execution, marking it as killed and forwarding it to the Sink module. As described earlier, each transaction execution consists of a sequence of read and write page accesses. For a read page access, the Transaction Manager requests access permission from the Concurrency Control Manager. When permission is received, the Transaction Manager requests the Resource Manager to read the corresponding disk page. The Resource Manager in turn sends the requests to the Buffer Manager that brings the page into memory. After the page has been loaded to the memory, the Transaction Manager requests CPU time to process the page from the Resource Manager. When page processing is complete, the Transaction Manager then begins executing the next page access. A write page access is executed in similar fashion to a read page access.

When all the page accesses of a transaction have been completed, the Transaction Manager initiates commit processing for the transaction. Transactions may sometimes have to be aborted due to data conflicts. In this case, the Concurrency Control Manager decides that the transaction should be aborted and informs the Transaction Manager.

The Transaction Manager then invokes the abort procedure for the transaction. After the abort procedure is completed, the transaction is restarted and follows the same data access pattern as the original transaction.

Transactions are also aborted when they are killed due to failing the Service Eligibility test. In this case, the Transaction Manager is the initiator of the abort process, and the aborted transaction is not restarted but sent, instead, to the Sink.

4.1.4 Resource Manager

The Resource Manager represents the SRTDBS operating system and controls access to the physical resources of the database system. In our model, the SRTDBS physical resources consist of multiple CPUs and multiple disks. There is a single common queue for the CPUs and the service discipline is Pre-emptive Resume, with preemption based on transaction priorities. Each of the disks has its own queue and is scheduled according to a Head-Of-Line (HOL) policy, with the request queue ordered by transaction priority.

The NumCPUs and NumDisks parameters specify the hardware resource composition, while the PageCPU and PageDisk parameters capture CPU and disk processing times per data page. We assume that preemption costs at the CPU are negligible compared to page processing times.

A single level buffer comprising of $NumBuffers$ identical page-sized buffer slots is modeled and the slots are allocated, replaced and synchronized according to the buffer management protocol in use. Each buffer slot has a dirty field, and if the dirty bit is set, the page is saved to the disk before replacement. The buffer wait queues are maintained in the order specified by the buffer management protocol. A page that is brought into the buffer pool is pinned for a uniformly distributed duration in the range $[MinPin, MaxPin]$. In our simulator, a separate instance of buffer manager was created for each of the buffer management protocols evaluated in our experiments.

4.1.5 Concurrency Control Manager

The Concurrency Control Manager maintains database consistency by regulating transaction access to data pages. It implements a concurrency control protocol, servicing concurrency control requests received from the Transaction Manager. These requests include requests for read access, write access, and to commit or abort a transaction. For processing each of these requests, the Concurrency Control Manager requests a period of CPU service from the Resource Manager. A parameter called *CReqCPU* specifies the CPU overhead required to process each request. A separate instance of the Concurrency Control Manager was created for each of the concurrency control protocols evaluated in our experiments.

As mentioned earlier, the concurrency control is implemented at the page granularity.

4.1.6 Sink

The Sink module receives both completed and killed transactions from the Transaction Manager. It gathers statistics on these transactions and measures the performance of the system from the application's perspective.

4.1.7 Priority Assignment

In our study, since covert-channel-free security is a correctness requirement, the database system is forced to assign transaction priorities based primarily on security clearance levels and only secondarily on deadlines. In particular, we assign priorities as a vector P = (LEVEL, INTRA), where LEVEL is the transaction security clearance level and INTRA is the value assigned by the priority mechanism used *within* the level. We assume that security levels are numbered from zero upwards, with zero corresponding to the lowest security level. Further, priority comparisons are made in lexicographic order with lower priority values implying higher priority (as in the UNIX operating system).

With the above scheme, transactions at a lower security level have higher priority than all transactions at a higher security level, a necessary condition for non-interference. For the intra-level priority mechanism, any priority assignment that results in good real-time performance can be used. For example, the classical Earliest Deadline [69] assignment where transactions with earlier deadlines have higher priority than transactions with later deadlines. In this case, the priority vector would be $\mathsf{P} = (\mathsf{LEVEL}, \mathsf{DEADLINE})$.

To help isolate and understand the performance cost that occurs due to having to eliminate covert channels, we have also simulated the performance achievable in the *absence* of covert channel security requirements. That is, the performance achievable if only Bell–LaPadula conditions had to be satisfied. For this scenario, a priority assignment of $\mathsf{P} = \mathsf{DEADLINE}$ is used.

4.2 Experimental Methodology

Our performance study of SRTDBS transaction scheduling algorithms consists of two major parts: concurrency control and buffer management. Chapters 5 and 6 focus on the Concurrency Control component, while Chapter 7 deals with Buffer Management. Our transaction processing protocols were originally designed for the full secure environment. However, in combination with the GUARD protocol described in Chapter 8, they are amenable even to the orange secure environment where it is acceptable to have well-defined bounded-bandwidth covert channels in exchange for performance improvement.

In the first phase of our work, in Chapter 5, we determine the performance penalty to be paid in terms of real-time performance, in order to ensure security in a real-time transaction processing system. To help isolate and understand the performance cost that occurs due to having to eliminate covert channels, we simulate the performance achievable in the *absence* of covert channel security requirements. That is, the performance achievable if only Bell–LaPadula conditions had to be satisfied. In that chapter we also determine which among the set of secure real-time CC protocols that we consider exhibits the best real-time performance. In Chapter 6, with insight obtained from our experiments on secure real-time CC protocols, we develop a novel dual protocol for simultaneously ensuring security and achieving the real-time performance objectives.

Having covered the concurrency control aspects of secure transaction processing, we shift our attention to Buffer Management in Chapter 7. There we evaluate the performance of a variety of buffer management protocols ranging from unsecure non-real-time policies to fully secure real-time policies. Then, in Chapter 8, we present an adaptive admission-control based protocol that ensures fairness while guaranteeing orange security.

In all of our experiments, the deadline assignments are such that each transaction is guaranteed to make its deadline if it were to execute alone in the system. This means that the only reasons for transactions to miss their deadlines are resource contention and data contention. The level of resource contention in the system is determined by the quantity of resources, the transaction processing costs, number of security levels, data access pattern, and the mean number of transactions in the system. Similarly, the level of data contention is determined by the database size, the transaction size distribution, the frequency of updates, number of security levels, data access pattern, and the mean number of transactions. The processing costs, the database size and the transaction size distribution are kept constant across almost all of our experiments, while the resource composition, data update probability, number of security levels, data access pattern and number of transactions are varied.

4.3 Performance Metrics

The performance metric of our experiments is $KillPercent$, which is the percentage of input transactions that the system is *unable* to complete before their deadlines. We compute this percentage also on a *per-clearance-level* basis. KillPercent values in the range of 0 to 20 percent are taken to represent system performance under "normal" loads, while KillPercent values in the range of 20 percent to 100 percent represent system performance under "heavy" loads. A long-term operating region where the kill percentage is high is obviously unrealistic for a viable SRTDBS. Exercising the system to high miss levels (as in our experiments), however, provides valuable information on the response of the algorithms to brief periods of stress loading [39]. All the KillPercent graphs of this thesis show mean values that have relative half-widths about the mean of less than 10

Table 4.3: Default Parameter Settings

$DBSize$	1000 pages	$NumCpus$	10
$ClassLevels$	2 (Secret, Public)	$NumDisks$	20
$ClearLevels$	2 (Secret, Public)	$PageCPU$	10 ms
$LowSlackFactor$	4.0	$PageDisk$	20 ms
$HighSlackFactor$	4.0	$NumBuffers$	50
$TransSize$	16 pages	$MinPin$	0 ms
$WriteProb$	0.5	$MaxPin$	100 ms
$NumGPageSet$	100	$SizeGPageSet$	200
$InterLoc$	0.14	$IntraLoc$	0.8
$LocalProb$	0.8	$GRefCount$	500
$CReqCPU$	1 ms	–	–

percent at the 90 percent confidence level, with each experiment having been run until at least 10000 transactions were processed by the system. Only statistically significant differences are discussed here.

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

The simulator was instrumented to generate a host of other statistical information, including resource utilizations, number of transaction restarts, buffer hit ratios, etc. These secondary measures help to explain the KillPercent behavior of the CC and BM protocols under various workloads and system conditions.

4.4 Default Parameter Settings

In most of our experiments, we model a two-security-level environment where the levels are called Secret and Public. The average transaction size is 16 pages and individual transaction sizes are uniformly distributed over the range (8,24) pages. The database size is set to 1000 pages. Though it may appear that this database size is rather small, we have chosen this parameter so as to represent only heavily accessed data, that is, "hotspots" [1]. Such hotspots are often observed in real life applications where typically 90% of accesses are made to 10% of the data items.

The resource parameter settings are such that the CPU time to process a page is 10 milliseconds while disk access times are 20 milliseconds. For experiments that were intended to factor in the effect of resource contention on the performance of the algorithms, the number of processors and disks were set to 10 and 20, respectively. For experiments that were intended to isolate the effects of data contention, an "infinite resource" situation [2], where there is no queuing for the processors and disks was simulated. A point to note here is that while abundant resources are usually not to be expected in conventional database systems, they may be more common in SRTDBS environments since many real-time systems are sized to handle transient heavy loading. This directly relates to the application-domain of RTDBSs, where functionality, rather than cost, is usually the driving consideration [39].

The other default parameter settings used in our experiments are given in Table 4.3. These parameters are set in accordance with the values used in earlier published simulation studies [40, 103]. While the absolute performance profiles of the protocols would, of course, change if alternative parameter settings were used, we expect that the *relative* performance of the protocols will remain qualitatively similar since the model parameters are not protocol-specific.

Chapter 5

Secure Real-Time CC

5.1 Introduction

In this chapter we address the issue of designing secure CC protocols for real-time transaction processing systems. As a first step to achieving this goal, we analyze which of the real-time CC protocols already available in the literature can be adapted to the secure environment. We find that while some protocols can be easily adapted to the secure environment, many others are *inherently* unsecure. We then profile the performance of the secure protocols and evaluate the performance penalty, in terms of real-time performance, to be paid for ensuring covert channel free security in an SRTDBS. Our studies show that the performance penalty in terms of the overall killpercent is marginal.

5.2 Secure CC Protocols

In combination with simple priority-based scheduling schemes such as Pre-emptive Resume and Head-of-Line, the secure priority assignment mentioned in Section 4.1.7 is adequate to securely handle conflicts for the physical resources such as the processors and disks. For resolving transaction *data* conflicts, however, an additional design decision has to be made – the choice of CC protocol.

At first glance it would seem that, in principle, any real-time CC protocol could be used in a SRTDBS and that the actual choice of protocol used in the system would be based only on the relative performance of these protocols. However, *not all the previously proposed real-time CC algorithms are amenable to supporting security requirements*. For example, consider the 2PL Wait Promote algorithm proposed in [4]: This protocol, which is based on 2PL, incorporates a *priority inheritance* mechanism [84] wherein, whenever a requester blocks behind a lower-priority lock holder, the lock holder's priority is promoted to that of the requester. In other words, the lock holder *inherits* the priority of the lock requester. The basic idea here is to reduce the blocking time of high priority transactions by increasing the priority of conflicting low priority lock holders (these low priority transactions execute faster and therefore release their locks earlier).

The Wait Promote approach is *not* suitable for SRTDBSs. This is because it permits the blocking of high priority transactions by low priority transactions which violates the requirement of *non-interference* between the transactions of different security levels (as mentioned earlier, non-interference means that low security transactions should not be able to distinguish between the presence or absence of high security transactions).

To generalize the above observation, a real-time CC protocol that permits *priority inversion* [84] to even a limited extent cannot be used in a SRTDBS. Apart from Wait Promote, other examples of real-time CC algorithms that fall into this category include WAIT-50 [39] and 2PL-OS/BI [3].

In the remainder of this section, we present a representative set of CC protocols that by virtue of being free from *priority inversion* could be used to resolve conflicts in an SRTDBS. These protocols use either locking or optimistic CC [11] as the basic regulatory mechanism.

5.2.1 2PL High Priority

The 2PL High Priority (2PL-HP) scheme [4] modifies the classical strict two-phase locking protocol (2PL) [28] by incorporating a high priority conflict resolution scheme which ensures that high priority transactions are not delayed by low priority transactions. In 2PL-HP, when a transaction requests a lock on a data item that is held by one or more higher priority transactions in a conflicting lock mode, the requesting transaction waits for the item to be released (the wait queue for a data item is managed in priority order). On the other hand, if the data item is held by only lower priority transactions in a conflicting lock mode, the lower priority transactions are *restarted* and the requesting transaction is granted the desired lock. Note that if priorities are assigned uniquely (as is usually the case in RTDBSs), 2PL-HP is inherently deadlock-free. A new reader can join a group of lock-holding readers only if its priority is higher than that of all the writers waiting for the lock.

5.2.2 OPT-SACRIFICE

The OPT-SACRIFICE algorithm [39] modifies the classical forward (or broadcast) optimistic CC protocol (OPT) [80, 70, 38] by incorporating a *priority sacrifice* mechanism. In this algorithm, a transaction that reaches its validation stage checks for conflicts with currently executing transactions. If conflicts are detected and one or more transactions in the conflict set is a higher priority transaction, then the validating transaction is restarted – that is, it is sacrificed in an effort to help the higher priority transactions make their deadlines.

5.2.3 OPT-WAIT

The OPT-WAIT algorithm [39] modifies the forward OPT protocol by incorporating a *priority wait* mechanism. In this algorithm, a transaction that reaches validation and finds higher priority transactions in its conflict set is "put on the shelf", that is, it is made to wait and not allowed to commit immediately. This gives the higher priority transactions a chance to make their deadlines first. While a transaction is waiting, it is possible that it will be restarted due to the commit of one of the conflicting higher priority transactions.

5.2.4 SOPT

A secure version of the classical forward OPT algorithm, called Secure OPT (SOPT), is described in [19, 90]. Here, whenever a conflict is detected between a high security transaction in its validation phase and a low security transaction, the high security transaction is aborted, while the low security transaction is not affected.

The SOPT protocol is similar to the OPT-SACRIFICE protocol described above wherein validating transactions are restarted if high priority transactions are present in their conflict set. However, in SOPT, transaction priorities are based *solely on security levels*, that is, it lacks real-time characteristics. Therefore, we have not included this algorithm in our performance study.

5.2.5 S2PL

A secure locking-based protocol called Secure 2PL (S2PL) is proposed in [20, 89]. The basic principle behind S2PL is to try to simulate the execution of conventional 2PL without blocking the actions of low security transactions by high security clearance transactions. This is accomplished by providing a new lock type called *virtual lock*, which is used by low security transactions that develop conflicts with high security transactions. The actions corresponding to setting of virtual locks are implemented on *private versions* of the data item (similar to optimistic CC). When the conflicting high security transaction commits and releases the data item, the virtual lock of the low security transaction is upgraded to a real lock and the operation is performed on the original data item. To complete this scheme, an additional lock type called *dependent virtual lock* is required apart from maintaining, for each executing transaction T_i, a list of the active transactions that precede T_i in the serialization order and a list of the active transactions that follow T_i in the serialization order. The details are given in [20, 89, 21, 91]. In conjunction with using secure two phase locking, the Undo/Redo algorithm [11] used for recovery must be modified to ensure the database correctness [19].

We have found that in certain situations S2PL *fails* to ensure security. More details on S2PL and the problem situations are given in Appendix B. In our implementation, we have fixed the loopholes in S2PL. Note that S2PL may not perform well in the real-time domain since it does not include any real-time-specific features. We include it here for two reasons: First, it serves as a baseline against which to compare the real-time CC algorithms. Second, we use it in one of the "dual approach" protocols that we evaluate in this study.

5.3 Performance Study

In this section we present the performance results from our experiments comparing the various secure CC protocols using the SRTDBS simulation model described in Chapter 4. All the experiments described in this chapter and the next chapter were performed on a system in which the buffer manager is not explicitly modeled. We assume that all data is accessed from disk and buffer pool considerations are therefore ignored. The default parameter settings used in experiments described in this chapter are as given in Table 4.3. Note, however, that the buffer related parameters are irrelevant here. The settings of the workload parameters were chosen with the objective of having significant data contention and resource contention in the system, thus helping to bring out the performance differences between the various CC protocols. For each experiment, any parameters different from the default settings are mentioned explicitly in the corresponding experimental section.

5.3.1 Comparative Protocols

To help isolate and understand the performance cost that occurs due to having to eliminate covert channels, we have also simulated the performance achievable in the *absence* of covert channel security requirements. That is, the performance achievable if only Bell–LaPadula conditions had to be satisfied. For this situation, a priority assignment of $P_T = DEADLINE$ is used.

In the following experiments, we will refer to the performance achievable under the above scenario as **DIRECT** since the Bell–LaPadula conditions prevent direct unauthorized access to data.

5.4 Overview of Experiments

We begin our performance evaluation by first developing a baseline experiment on an RTDBS having both data and resource contention. Further experiments are constructed around the baseline experiment by varying a few parameters at a time. After exploring the combined effect of resource and data contention, we isolate the effect of data contention. We then explore the dependence of performance on various workload parameters - the page update frequency, the ratio of the transaction arrival rates of various classes, the slack ratios assigned to transaction deadlines, etc. We also evaluate an alternative data access model wherein writes to higher security levels are not permitted.

Through a series of experiments, we assess the cost to be paid, in terms of killed transactions, of ensuring direct security in the entire RTDBS, of ensuring covert channel security at the physical resources, of ensuring covert channel security at the data, respectively. We then evaluate the degree of interference that could arise due to current technology restrictions such as the non-preemptability of disks, and we also compare the secure RTDBS with a traditional multiclass RTDBS. Finally, while all the above experiments are in the context of a a two-security-level system, our last experiment analyzes the performance of systems with five security levels.

5.5 Experiment 1: Resource and Data Contention

The settings of the workload parameters and system parameters for the baseline experiment are listed in Table 4.3. These settings were chosen with the objective of having significant data and resource contention in the system, thus helping to bring out the performance differences between the various concurrency control protocols.

In this experiment, there are two security levels: *Secret* and *Public*. For this system, Figures 5.1a and 5.1b show the KillPercent behavior as a function of the overall transaction arrival rate. In Figure 5.1a, the overall kill percentages of the secure algorithms and their

DIRECT (Bell–LaPadula) counterparts is profiled. We see here that at normal loads the performance of the secure algorithms is worse than that of their DIRECT counterparts – this is especially so for OPT-SACRIFICE for reasons explained later. In contrast, under heavy loads the performance of the secure algorithms is actually better than that of the DIRECT algorithms. The reason for this is that while the Earliest Deadline priority assignment is excellent for a set of tasks that *can* be completed before their deadlines, it becomes progressively worse as the task set overloads its capacity [48, 39]. In this situation, the secure protocols feature of grouping the transactions into prioritized levels means that Earliest Deadline is operational among smaller sets of transactions, leading to improved performance at higher loads. In summary, although elimination of covert channels results in performance degradation at low loads, it is helpful under heavy loads.

Focusing on the secure algorithms, we observe first in Figure 5.1a that the performance of OPT-SACRIFICE and 2PL-HP is almost the same and significantly worse than that of OPT-WAIT. In fact, their performance is no better than that of S2PL which, as mentioned in Section 5.2.5, is a non-real-time protocol!

The results for 2PL-HP and OPT-WAIT are similar to those observed for conventional real-time transaction concurrency control [39]. The poor performance of 2PL-HP is because it suffers from wasted restarts (restarts caused by a transaction that is eventually killed). In contrast, in OPT-WAIT, all restarts are made “on demand” and at the commit time of a higher priority transaction.

With respect to OPT-SACRIFICE, however, there is a change in performance behavior in the secure environment since in the conventional real-time domain, OPT-SACRIFICE used to provide a performance that was *intermediate* to that of OPT-WAIT and 2PL-HP. Its main problem was that it suffered from wasted sacrifices (sacrifices for a transaction that is eventually killed). The effect of this problem is *magnified* in the secure domain for the following reason: In conventional real-time CC, a transaction that reached validation close to its deadline would usually not have to sacrifice itself since it would have high priority. However, in the secure model, where the transaction *level* is also a factor in the priority assignment, Secret transactions that manage to reach their validation stage close

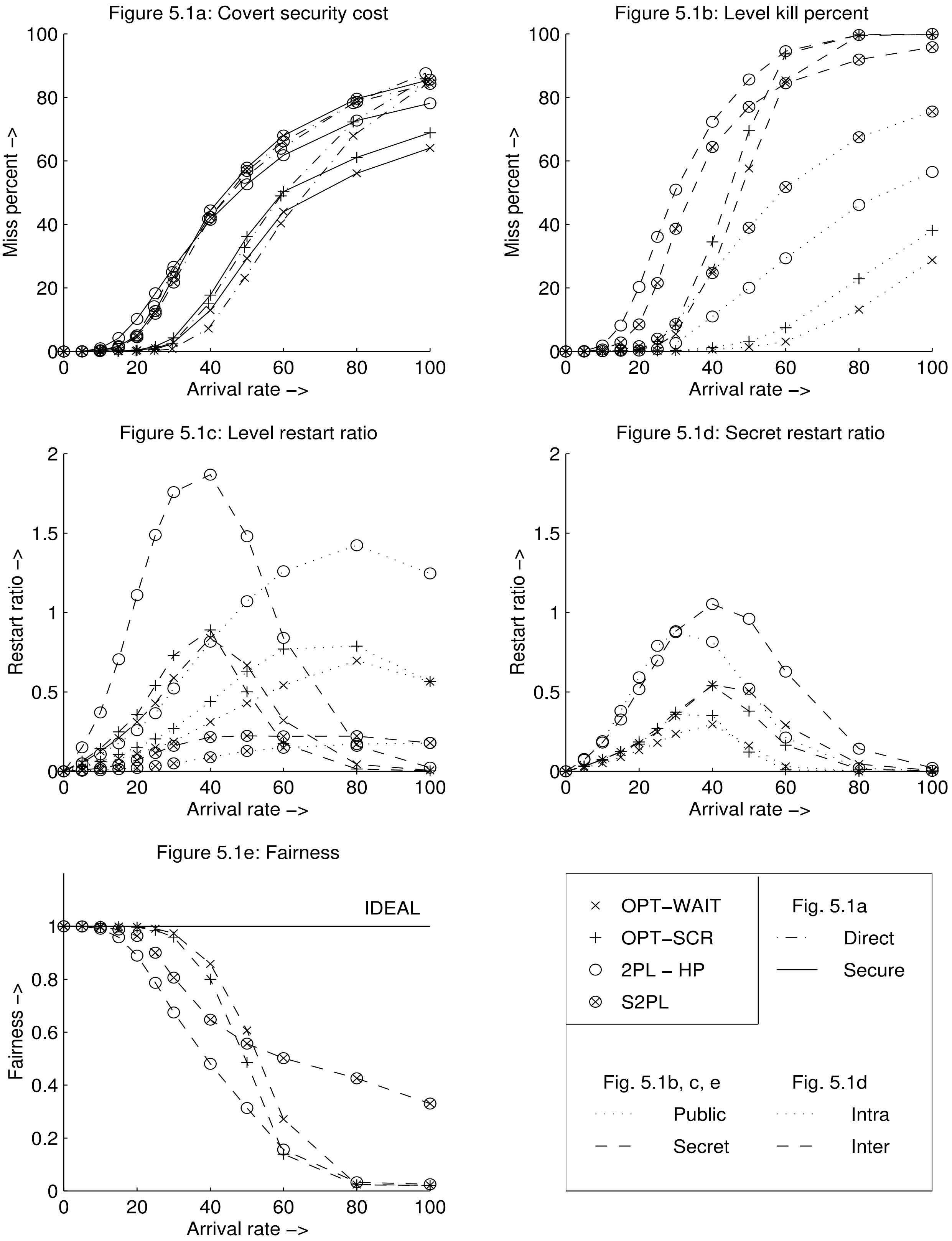


Figure 5.1: Resource and Data contention

to their deadline may still be restarted due to conflict with a Public transaction. The negative impact of this restart is much more than that in 2PL-HP since the transaction is restarted only after having used up its entire resource requirement. This was confirmed by measuring the CPU utilizations of the various protocols – we found that OPT-SACRIFICE had noticeably higher CPU utilization than the other algorithms.

Another interesting feature that arises here is due to the access pattern restrictions of the Bell-LaPadula model. The definition of conflict in forward optimistic concurrency control is that a conflict exists between the validating transaction V and an executing transaction E if and only if the intersection of the write set of V and the current read set of E is non-empty. For the LaPadula model, where *blind writes* are permitted due to the "read-below, write-above" paradigm, optimistic algorithms will correctly conclude that there is no conflict between items that are in the intersection of the write set of V and the write set of E but not in the read set of E. However, for 2PL-HP such conflicts show up as write-write conflicts and result in either blocking or restarts, thereby further deteriorating the performance of the locking protocol.

Moving on to S2PL, we find that, although it is based on basic 2PL, which performs very poorly in the real-time environment [39], it manages to perform on par with 2PL-HP and OPT-SACRIFICE here in spite of not being deadline cognizant. This is due to its "optimistic-like" feature of virtual commit, which considerably reduces the amount of blocking associated with 2PL. This phenomenon is similar to that seen in [39], wherein a conventional optimistic protocol performed better than locking-based real-time protocols.

In Figure 5.1b, we present the kill percentages of the various concurrency control protocols on a *per-security-level* basis. This graph clearly shows how the high-security Secret transaction class (dashed lines) suffers much more than the Public transaction class (dotted lines) to satisfy the goal of avoiding covert channels. Figure 5.1c provides statistics about the corresponding breakup of the "restarts ratio" (the average number of restarts of a transaction) on a level basis. We see here that Secret transactions are restarted much more often than Public transactions under normal loads. Under heavy loads the number of restarts decrease for Secret transactions since resource contention, rather than data contention, becomes the more dominant reason for these transactions missing their deadlines.

In Figure 5.1d, we present a different view of the transaction restarts picture. Here, the restarts of Secret transactions are categorized into those caused by Public transactions (i.e. inter-level restarts) and those caused by Secret transactions (i.e. intra-level restarts). An important observation here is that *only inter-level data conflicts result in security violations* and have to be resolved always in favor of public transactions. Therefore, the restart breakup is meaningful only for Secret transactions and all restarts are intra-level for Public transactions. The graph clearly shows that the Secret transactions suffer to a greater extent from inter-level conflicts (dashed lines) than from intra-level conflicts (dotted lines).

Finally, in Figure 5.1e, we plot the *fairness factor* of each protocol, that is, the ratio of the number of Secret transaction commits to the overall number of transaction commits. This measure quantifies the extent to which Secret algorithms are discriminated against in a covert-channel-free system with respect to an ideal fairness factor of 1.0. We observe that at low loads when virtually all transactions make their deadlines, all the concurrency control protocols are fair. However, as the loading increases, all the protocols become increasingly unfair. The point to note, however, is that OPT-WAIT provides greater fairness over most of the loading range as compared to OPT-SACRIFICE and 2PL-HP.

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

5.6 Experiment 2: Pure Data Contention

The goal of our next experiment was to isolate the influence of data contention on the performance of the concurrency control protocols. For this experiment, therefore, the resources were made "infinite", keeping all the other parameter values the same as those used in the baseline experiment. The performance results for this system configuration are presented in Figures 5.2a through 5.2c. We observe in these figures that the differences in the relative performance of the various protocols increases as compared to those seen in the previous experiment. The overall (Figure 5.2a) as well as the per-level (Figure 5.2b) kill percentages of OPT-WAIT are considerably better than those of 2PL-HP. Here, OPT-WAIT does better than 2PL-HP for two reasons: First, the basic wasted restarts problem of 2PL-HP occurs here too and is magnified due to the higher level of data contention. Second, the blocking component of 2PL-HP reduces the number of transactions that are executing and making progress. This blocking causes an increase in the average number of transactions in the system, thus generating more conflicts and a greater number of restarts. With OPT-WAIT, however, transactions are always executing and are never blocked.

Moving on to the performance of OPT-SACRIFICE, we observe that, unlike the previous experiment, it is somewhat improved as compared to 2PL-HP. The reason for this is that the wasted utilization caused due to its wasted sacrifices has less of an impact here since resource contention is not an issue.

Finally, in Figure 5.2c, OPT-WAIT shows close to ideal fairness over most of the loading range since it misses very few deadlines overall. In contrast, 2PL-HP and OPT-SACRIFICE are noticeably unfair with increasing loading levels.

In the remaining experiments, we do not present S2PL since it performed significantly worse than OPT-WAIT in all of these experiments. However, we use S2PL in one of our "dual" CC protocols described in Chapter 6 for improving the system performance.

5.7 Experiment 3: The Effect of Write Probability

This experiment was conducted to profile the effect of the write probability on the real-time performance for three representative mean transaction arrival rates: LOW (arrival rate = 15), MEDIUM (arrival rate = 50) and HIGH (arrival rate = 80). For each of these arrival rates, the write probability was varied from 0.0 to 1.0 in steps of 0.1 and the resulting performance was observed. The rest of the parameters were kept the same as those of Experiment 1.

Figure 5.3 shows the KillPercent values plotted against write probability for MEDIUM

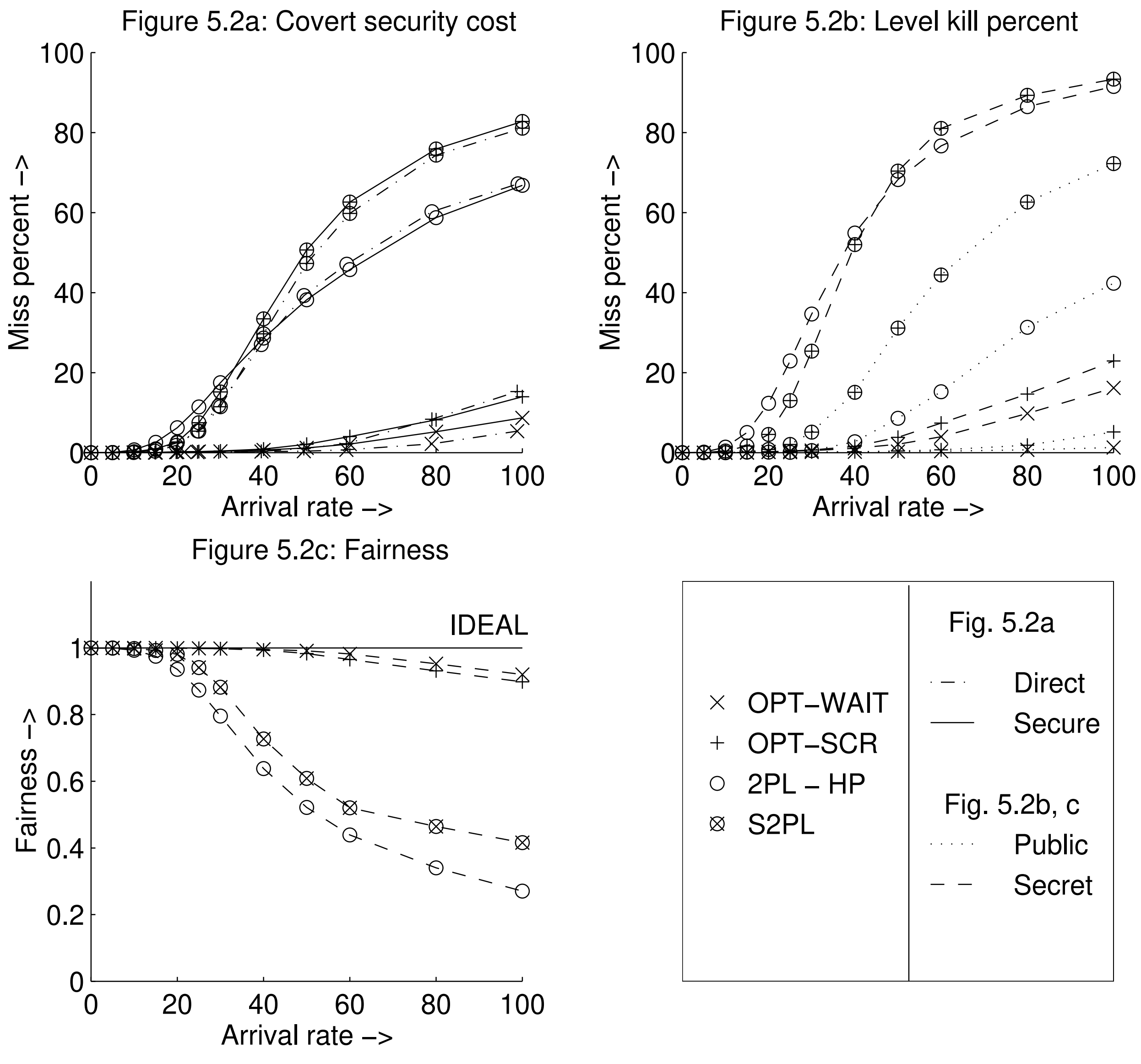


Figure 5.2: Pure data contention

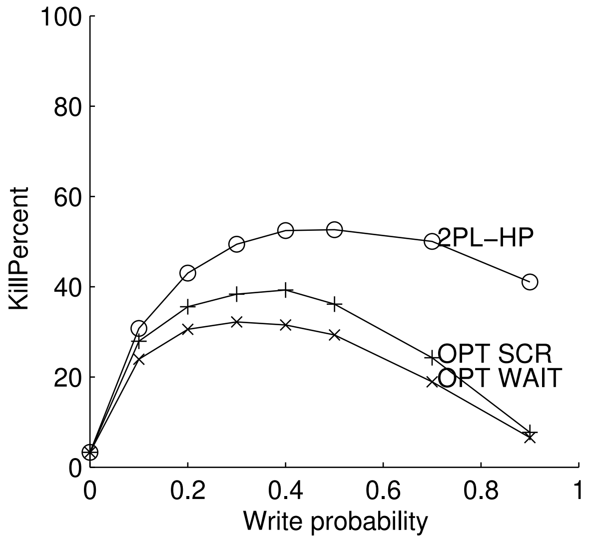


Figure 5.3: Effect of write probability

loads (we omit the HIGH and LOW performance graphs for brevity as the results are qualitatively similar). The locking protocol 2PL-HP shows poor performance at all write probability levels compared to the optimistic protocols and this difference is magnified at high write probability levels. In optimistic protocols, since the inter-level writes are blind writes (due to Bell–LaPadula conditions), write-write conflicts do not appear in the conflict set as explained in Experiment 1. Therefore, at high write probabilities the data conflicts in fact reduce for optimistic protocols, whereas, for locking protocols such as 2PL-HP, high write probability means blocking or restarting of many transactions (because of the exclusive nature of write locks).

5.8 Experiment 4: Unequal Arrival Rates

In the above experiments, a transaction is equally likely to belong to any of the security clearance levels. In other words, there is an equal contribution to the system workload from all security classification levels. However, in practice, there could be imbalances in the per-class workload and the purpose of this experiment is to capture such scenarios. For instance, in a system there can be more Public transaction arrivals than Secret transaction

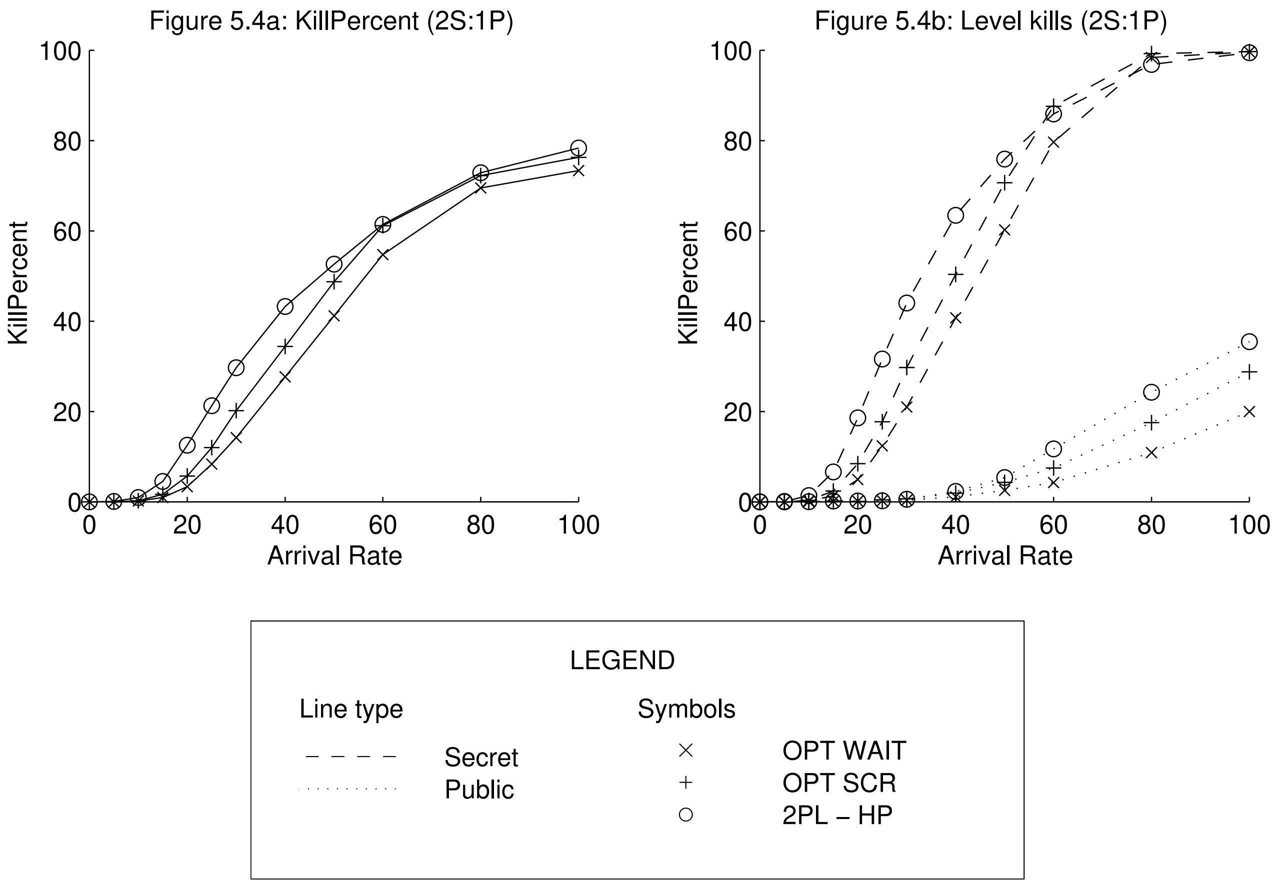


Figure 5.4: Unequal arrival rate (2S:1P)

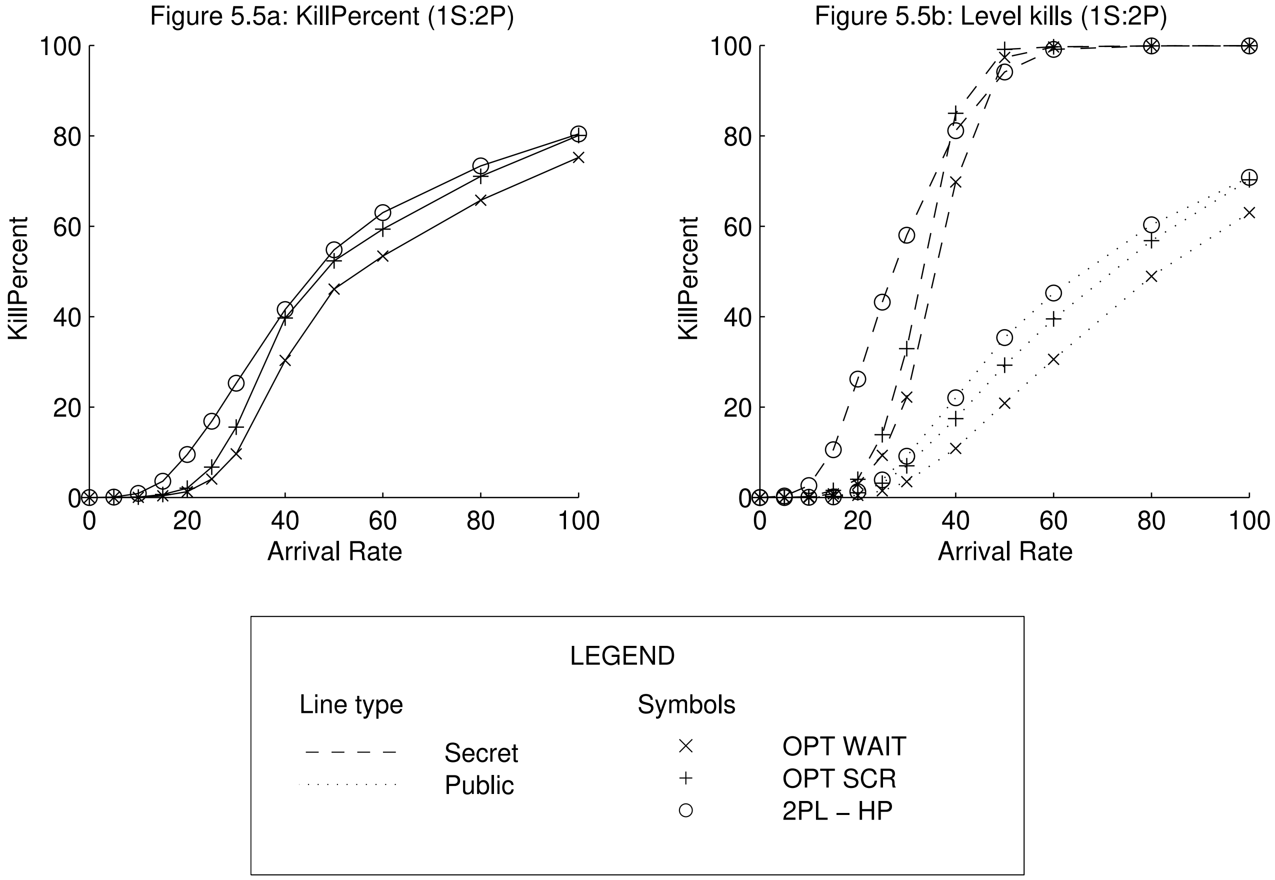


Figure 5.5: Unequal arrival rate experiment (1S:2P)

arrivals or vice-versa. We explore two scenarios here:

2S:1P – The arrival rate of Secret transactions is double that of the Public transactions.

1S:2P – The arrival rate of Public transactions is double that of the Secret transactions.

For the first case (2S:1P), Figure 5.4a shows the overall kill percentage characteristics and Figure 5.4b shows the level-wise split-up of the kill percentage, as a function of the <u>overall</u> arrival rate. Note that the increase in Secret level arrivals to the system does not affect the Public class performance. In fact, the Public class performance obtained in this experiment is better than the Public class performance obtained in the equal arrival rate case (Experiment 1) because of the reduction in intra-level conflicts – the Public class transaction arrival rate in this experiment is only two third that of the baseline experiment.

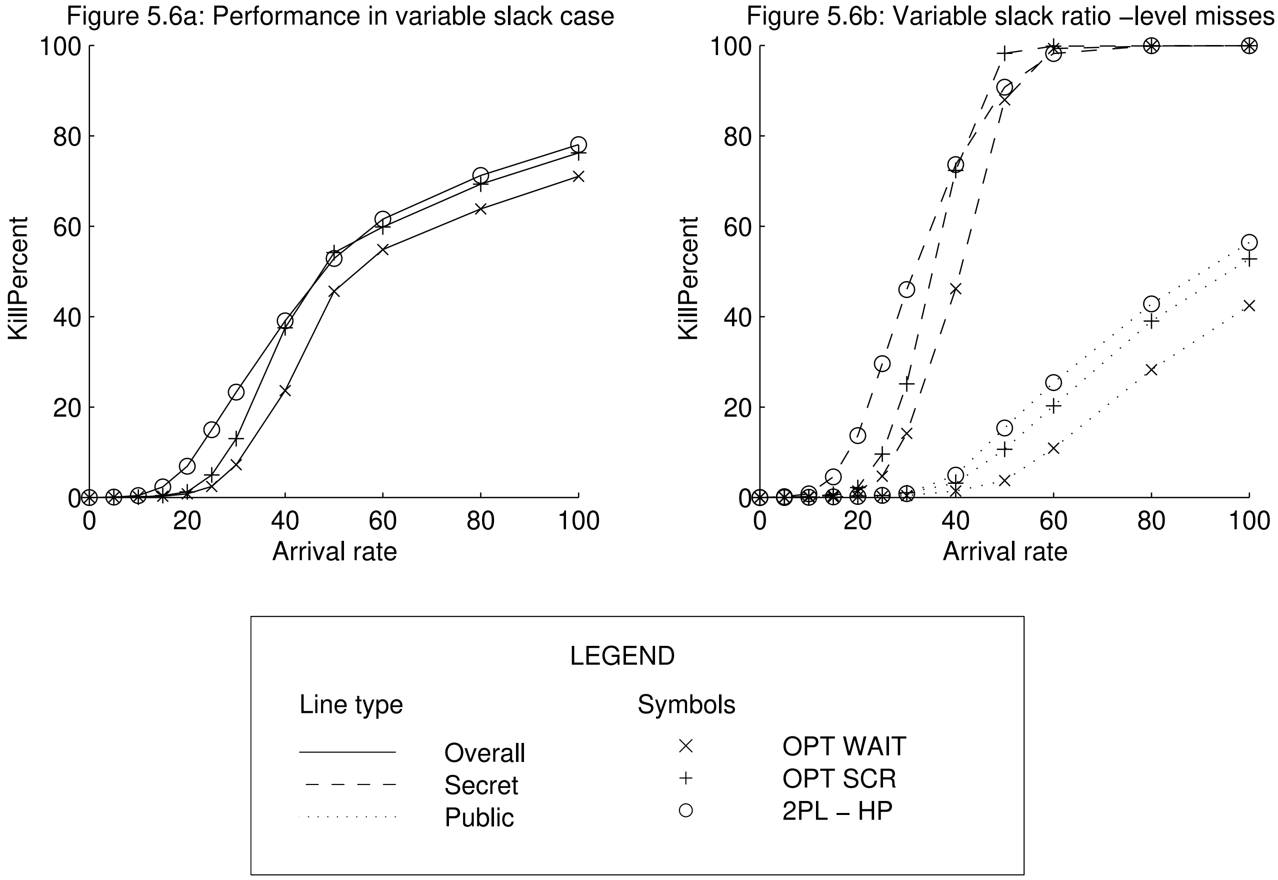


Figure 5.6: Variable slack experiment

In fact, the Secret class performance also improves (compared to the Secret performance in Experiment 1) due to the reduction in competition from Public transactions.

The results for the second workload scenario (1S:2P) are shown in Figures 5.5a and 5.5b. The increase in intra-level conflicts (due to the large Public class population) is the reason for the large number of Public class deadline misses. The inter-level conflicts also increase, causing the Secret transactions to lose heavily. In this experiment too, overall OPT-WAIT performed better than the other protocols compared.

5.9 Experiment 5: Variable Slack Ratio

In our experiments so far, we investigated the case where all transactions have the same slack ratio. We now move on to study the situation where different transactions may have different slack ratios. In this experiment, the transaction deadline is assigned by the formula DF2 described in Section 4.1.2 : $D_T = A_T + Uniform(LowSlackFactor, HighSlackFactor) * R_{max}$. The $LowSlackFactor$ and $HighSlackFactor$ parameters were set to 1.33 and 4.0, respectively, with the remaining parameters being the same as that of Experiment 1. These slack parameter settings were chosen to ensure that the *mean* slack ratio is the same as that of the baseline experiment (i.e., 4.0).

The results of this experiment are shown in Figures 5.6a and 5.6b respectively. One noticeable difference here is that at medium loads, the OPT-SACRIFICE kill percentage curve rises rapidly as the arrival rate increases, and it becomes almost as bad as 2PL-HP (Figure 5.6a). The reason for this behavior is evident from the Secret class performance (Figure 5.6b) in this range. Unlike the previous experiments, here there are transactions with less slack present in the workload and due to the heavy shortage of resources experienced by Secret class transactions many of these transactions miss their deadlines. This trend is observable in the case of Public class transactions though its not as predominent as in the case of Secret class transactions. This characteristic is visible even in OPT-WAIT as can be observed from Figure 5.6b. Nevertheless, OPT-WAIT does not suffer from the wasted restarts problem as explained in Experiment 1 and therefore it performs better than OPT-SACRIFICE.

5.10 Experiment 6: Restricted Write Model

In this experiment, transaction accesses are according to the Restricted Write model (see Figure 4.2c) with the remaining parameter values being the same as those of Experiment 1. The results are shown in Figures 5.7a through 5.7c. Note that the kill percentage behavior (Figures 5.7a, 5.7b) is similar to what was obtained in Experiment 1. One noticeable difference, however, is that Public class kill percentages are high for all the protocols, compared to the corresponding values obtained in the baseline experiment. This is due to the increased intra-level data conflicts arising in the system due to the confinement of Public class writes to the Public partition of the database. This is further evident from

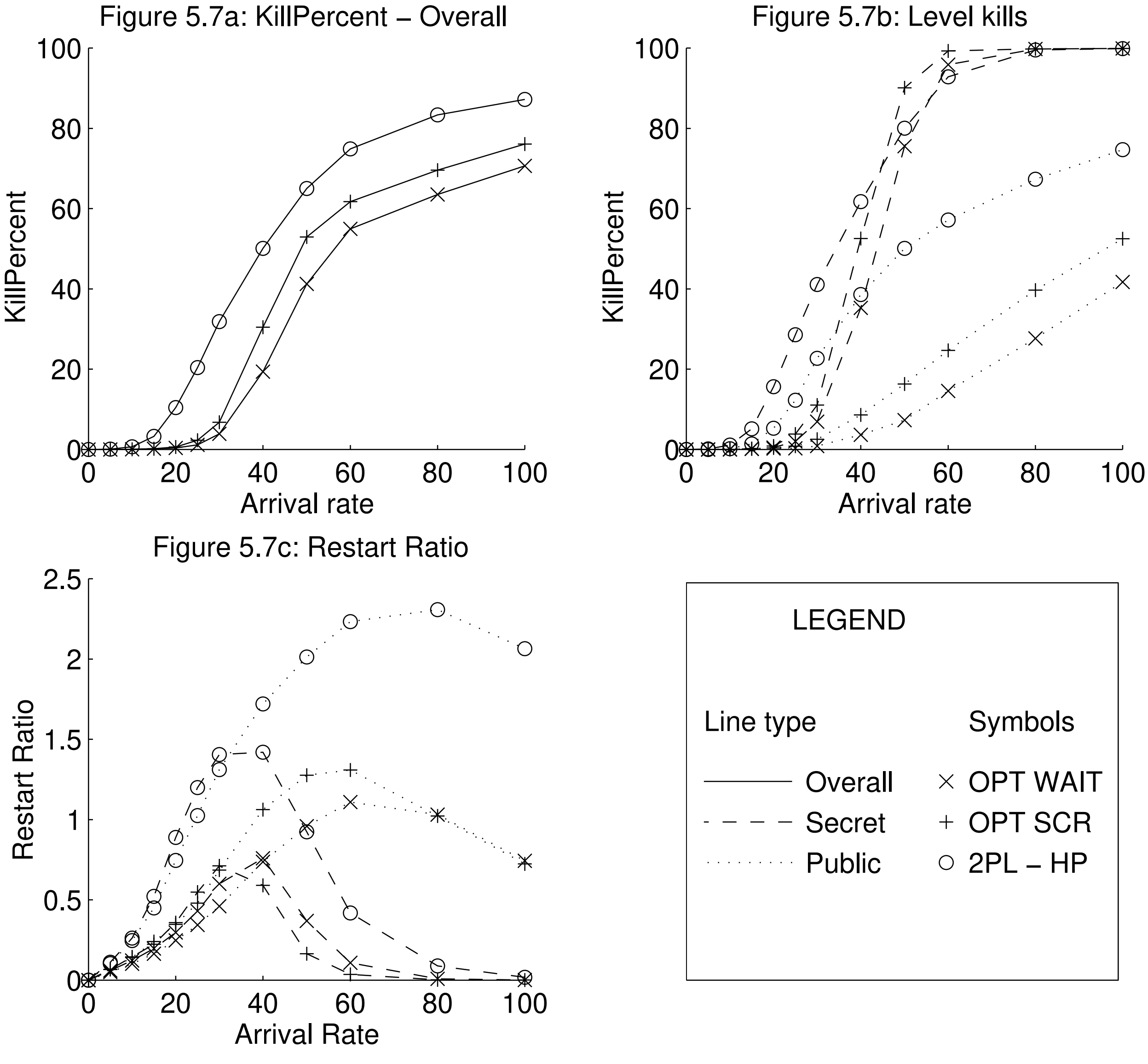


Figure 5.7: Restricted Write

Figure 5.7c, where we observe that the Public class restart ratios – for all the protocols – are noticeably higher compared to their correspoding values in Experiment 1.

The inter-level data conflicts reduce because the inter-level writes are disallowed. Consequently, the Secret class performance improves, especially at low loads. At high loads, however, this improvement is not noticeable because the physical resources are mainly hogged by the Public transactions and therefore Secret transactions miss their deadlines due to resource contention, rather than data contention.

5.11 Experiment 7: Direct Security Experiment

In Experiment 1, we investigated the performance cost of providing covert channel security for a system that enforces the Bell–LaPadula access restrictions. Here, we shift our focus to the evaluation of the real-time performance cost to be paid for enforcing direct (Bell–LaPadula) security in a conventional single level RTDBS (hereafter referred to as UNSECURE). That is, we compare DIRECT against UNSECURE.

The system model and parameters are kept identical to those of Experiment 1 except that the UNSECURE system has only a single security level. In both systems, the transaction priority for data and resources is assigned as P = DEADLINE. In the UNSECURE model, a transaction can access all data elements in the system. On the other hand, in the DIRECT model transaction data accesses are constrained by the Bell–LaPadula data access restrictions. Note that the difference in performance between DIRECT and UNSECURE can only arise out of the enforcement of direct security access constraints.

The performance results for this experiment are shown in Figures 5.8a–e. The kill percent curves (Figure 5.8a) show that there is no significant performance cost, in terms of missed deadlines, to be paid for enforcing direct security. Further, from Figure 5.8b, where the level wise split-up of DIRECT kill percentage is given, it can be seen that the Secret class performance is poorer than the Public class performance regardless of the CC protocol in use. This is because there are more write operations performed on the Secret partition of the database than there are at the Public partition. As a result, more Secret transactions get restarted, as is evident from the restart ratio characteristics shown in

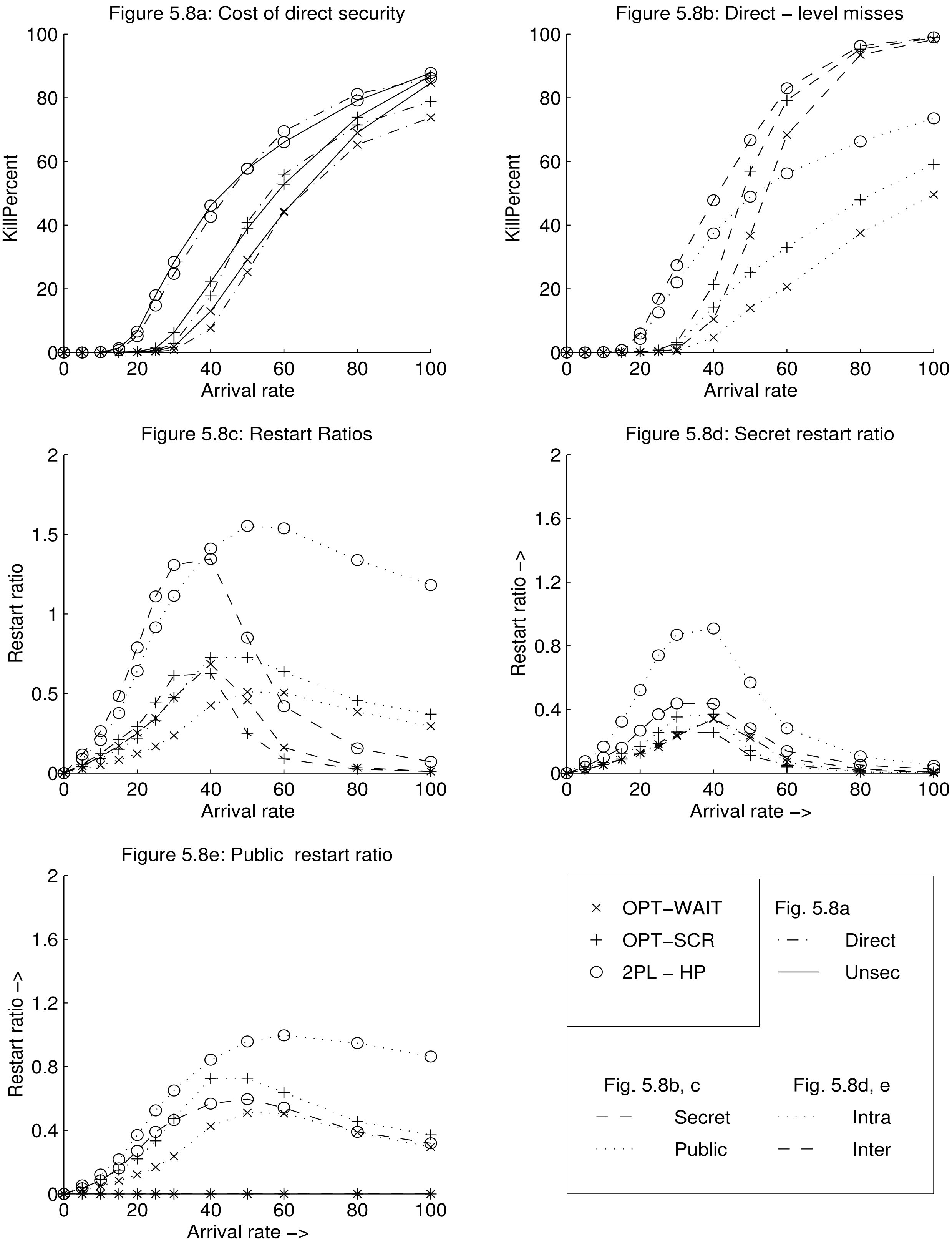


Figure 5.8: Direct Security

Figure 5.8c.

The Secret restarts can occur due to the intra-level conflicts or inter-level conflicts. Figure 5.8d shows the split-up of Secret restarts into inter-level restarts and intra-level restarts. Similarly, Public transactions could also be restarted due to any of these two conflict types. One interesting point to note here is that, in optimistic protocols, the Public transactions do not get restarted by Secret transactions due to Bell–LaPadula access restrictions. This is further evident from Figure 5.8e where the Public restarts statistics due to inter-level conflicts (conflicts from Secret class transactions) and intra-level conflicts (conflicts from other Public transactions) are shown. This explains the reason for optimistic protocols being more unfair (to the Secret class) than 2PL-HP.

To summarize, there is no significant performance cost to be paid, in terms of kill percentage, for providing direct security in an RTDBS . Also it was observed that the CC protocols in general and the optimistic protocols in particular are unfair to Secret class transactions even where only direct security is provided.

5.12 Experiment 8: Secure Resource Scheduling

This experiment was conducted for isolating the performance impact of the preferential physical resource access given to Public class transactions over Secret class transactions in order to prevent covert channels. The model used here captures the hypothetical situation wherein the inter-level data conflicts are completely absent and only inter-level resource conflicts are present in the system.

In this experiment, transactions follow the Partitioned data access model (see Figure 4.2d). All other parameter values for this experiment are the same as those of Experiment 1.

The result of this experiment is shown in Figure 5.9, which shows the level-wise split up of KillPercent performance. The difficulty experienced by Secret transactions in obtaining resources in the presence of competition from the Public transactions makes them vulnerable to missing their deadlines. This is true for all protocols including OPT-WAIT. Though the waiting strategy of OPT-WAIT benefits Public transactions, it hardly helps

Secret transactions since they are discriminated against at the physical resources.

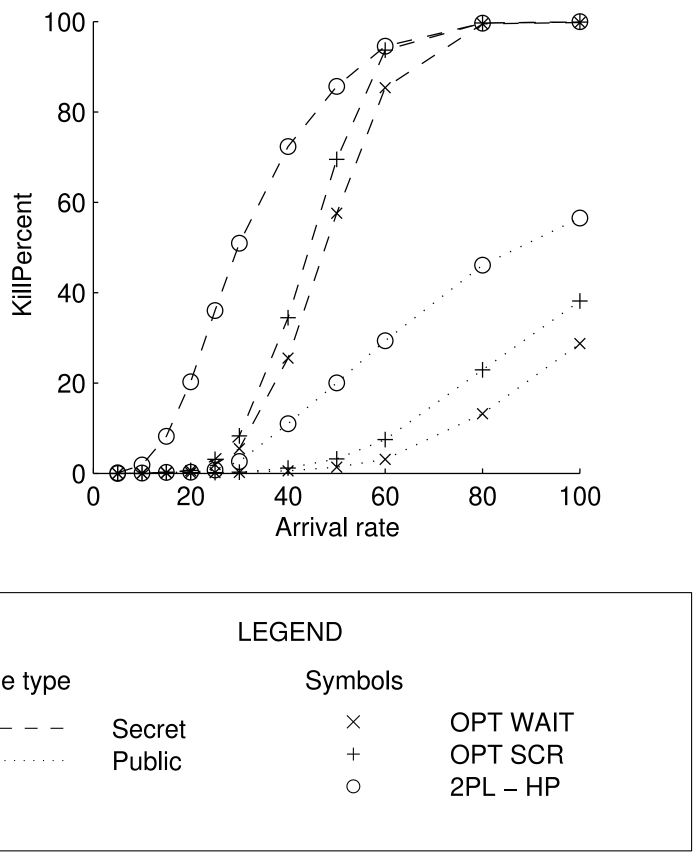


Figure 5.9: Secure Resource Scheduling

5.13 Experiment 9: Secure CC + Unsecure Resource

In this experiment, we employed the secure CC protocols described in Section 5.2 for resolving the data conflicts. Consequently, the transaction priority at the CC mechanism is given by the vector $\mathsf{P} = (\mathsf{LEVEL}, \mathsf{DEADLINE})$. However, the transaction priority at resources is assigned by $\mathsf{P} = \mathsf{DEADLINE}$. Note that in this system the physical resources are unsecure as they may be used for covert signaling. However, such a system may be justifiable in some application contexts, where at the physical resources, noise due to other transactions is likely to be generated for "free". Moreover, this experiment isolates the performance costs incurred for preventing *data* covert channels.

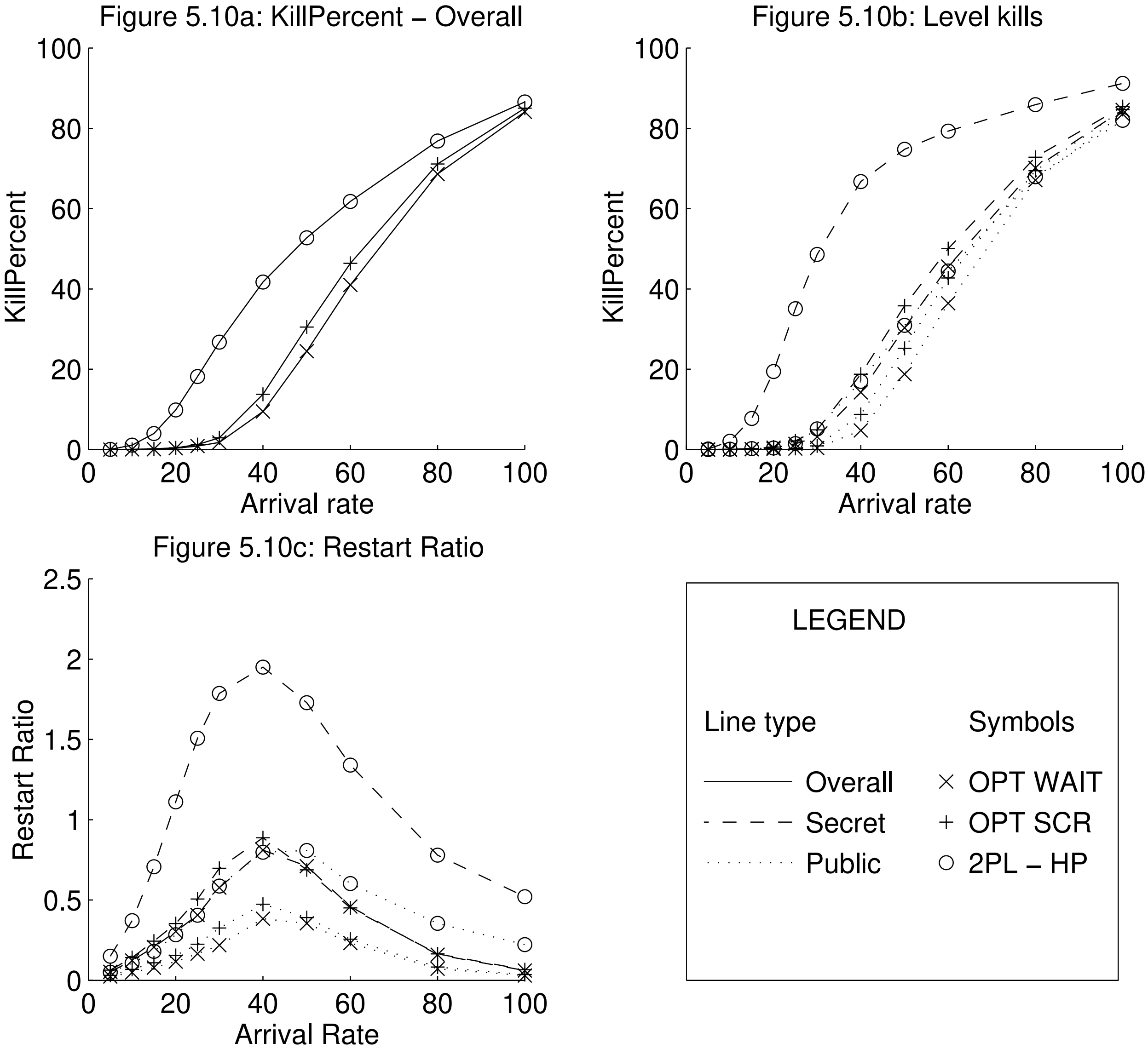


Figure 5.10: Unsecure Resource

The results of the above experiment are shown in Figures 5.10a through 5.10c. With regard to the overall kill percentage performance shown in Figure 5.10a, 2PL-HP is noticeably worse than other protocols. This is due to the fact that with 2PL-HP, most of the Secret transactions often get blocked or restarted (see Figure 5.10c) due to the conflicting Public transactions. Therefore, the Secret transactions cannot fully benefit from the unsecure priority assignment at resources. This is further evident from Figure 5.10b, where the Secret class performance in 2PL-HP is definitely worse than its Public class performance. However, in the optimistic protocols, transactions that reach the validation stage are likely to be urgent transactions since the priority given at resources is purely based on transaction deadlines. In this situation, although it is possible for a Public transaction to restart Secret transactions, it is likely that the Public transaction that enters the validation stage is most probably an urgent transaction.

One may have expected the Secret class performance in OPT-SACRIFICE to be still worse, compared to 2PL-HP, since the presence of even a single Public transaction in the conflict set of a validating Secret transaction would require its sacrifice. However, it is impossible to have a Public transaction present in the conflict set of Secret transaction because of the Bell–LaPadula conditions — no Public transaction's read can conflict with a Secret transaction's write. Therefore, the Secret class performance in OPT-SACRIFICE turns out to be much better than that of 2PL-HP.

In summary, a reasonable level of fairness is achievable for optimistic protocols, if the covert channel security at physical resources is not enforced.

5.14 Experiment 10: Non-pre-emptable Resources

The current technology disks are non-pre-emptable. Therefore, a Public transaction could experience a delay in acquiring the disk that is currently in use by a Secret transaction. This delay could be exploited for covert signaling between Secret and Public transactions. However, the covert signaling could be prevented by *artificially delaying all Public disk requests* for a (worst case) disk read time – we will hereafter refer to this system as the SECURE-DISK system. A similar approach is suggested in [36] in the context of distributed

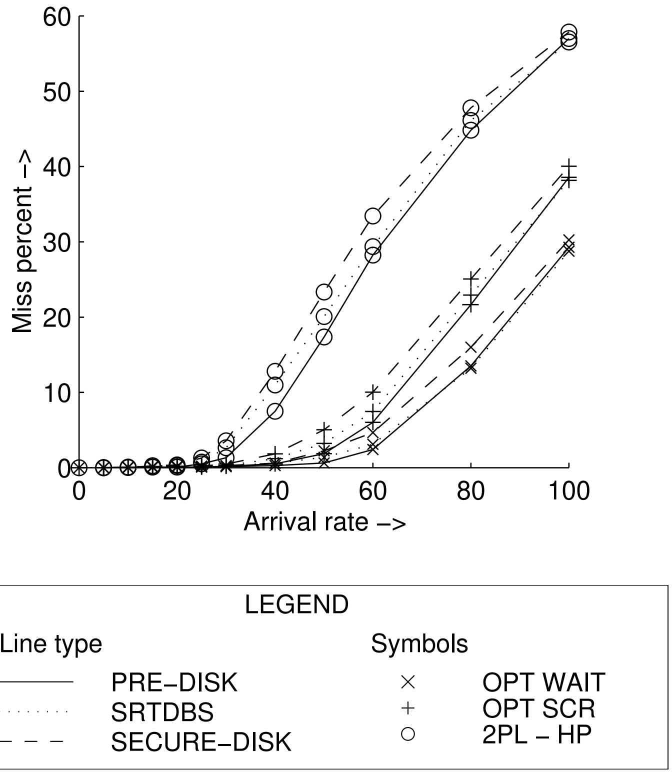


Figure 5.11: Non-premptable resources

operating systems.

In this experiment, we try to evaluate the degree of interference from Secret to Public transactions due to non-preemptable disks and quantify the performance cost to be paid for ensuring covert channel security at the disks. We also modeled a hypothetical system that is fitted with *fully pre-emptable disks*, hereafter referred to as PRE-DISK.

For this experiment, the parameter values are the same as those of Experiment 1. The **Public** class kill percentage behavior is shown in Figure 5.11 for the PRE-DISK (full lines), SRTDBS (dotted lines), and SECURE-DISK (dashed lines) systems. As can be clearly seen in this figure there is a performance degradation in the SRTDBS compared to the PRE-DISK system. This can only be due to the interference from the Secret transactions at disk due to its non-preemptive nature. Also, the Public class performance in the SECURE-DISK system is worse than the other two systems due to the artificial delay.

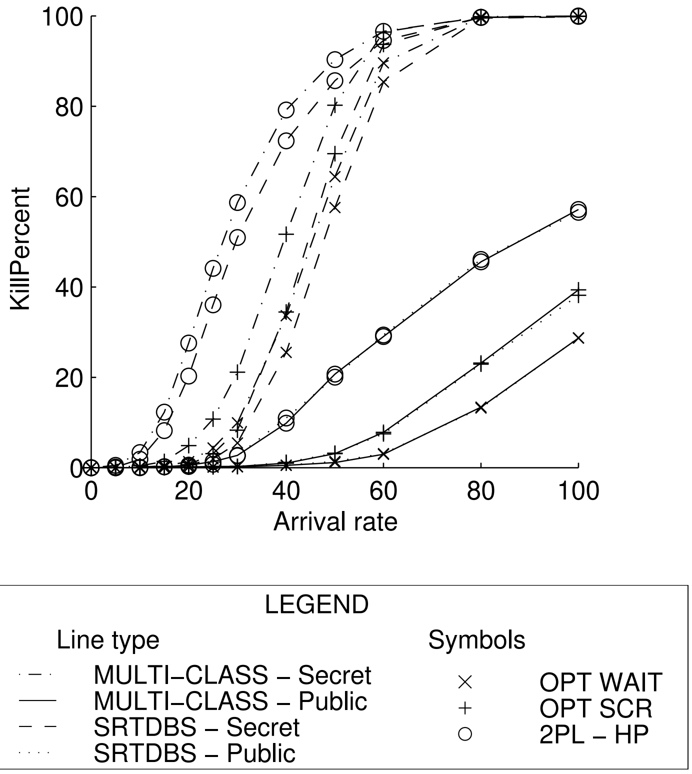


Figure 5.12: SRTDBS Vs MULTI-CLASS

In summary, immunity from covert channels arising out of the non-premptive behavior of the disk could be obtained at an additional performance cost.

5.15 Experiment 11: SRTDBS Vs MULTI-CLASS

In a MULTI-CLASS system, a transaction belongs to one of a finite number of classes and the transaction priority is assigned by the vector P=(CLASS, INTRA) where CLASS is the transaction class and INTRA is the value assigned by the priority mechanism used *within* the level. We assume that classes are numbered from zero upwards, with zero corresponding to the lowest security class. Further, priority comparisons are made in lexicographic order with lower priority values implying higher priority. In this experiment, the priority vector is P = (CLASS, DEADLINE), since we use Earliest Deadline assignment policy within the level. Note that a SRTDBS is analogous to a multi-class RTDBS where the security levels in SRTDBS correspond to the transaction classes in the multi-class environment. However, one difference in the SRTDBS is that the Bell–LaPadula conditions for data access have been enforced in an SRTDBS but not in the MULTI-CLASS system.

In the MULTI-CLASS system that we modeled, there were two classes namely Secret and Public, analogous to the Secret and Public in the SRTDBS. Public class transactions are given more priority than Secret class transactions for data and resource access. All system and workload parameters for this experiment are kept the same as that of Experiment 1.

The results of this experiment are shown in Figure 5.12, and show that the SRTDBS performance is very similar to the traditional multi class RTDBS performance.

5.16 Experiment 12: Increased Security Levels

A two-security-level system was modeled in the previous experiments. In this experiment, we investigated the performance behavior for a *five-security-level* system, where the levels are *TopSecret, Secret, Confidential, Classified* and *Public.* The remaining parameter values are the same as those used in Experiment 1. The results for this experiment are shown in Figures 5.13a through 5.13e (for graph clarity, the results for only 2PL-HP and OPT-WAIT are presented).

In Figure 5.13a, we see that there is a greater difference between the performance of the secure algorithms and their DIRECT counterparts at normal loads, as compared to the equivalent two-level experiment (Experiment 1). This is because in a five-level system, priority is much more level-based than deadline-based, thereby denying Earliest Deadline its ability to complete most transactions in a feasible set under normal loads. Under heavy loads, however, the smaller sizes of the transaction sets in each level results in Earliest Deadline performing well for the low security transactions. (This feature of Earliest Deadline was used in the Adaptive Earliest Deadline scheduling algorithm described in [39] where transactions are split up into prioritized groups with the size of the highest priority group set equal to an estimate of the maximum number of transactions that could be successfully completed by Earliest Deadline.)

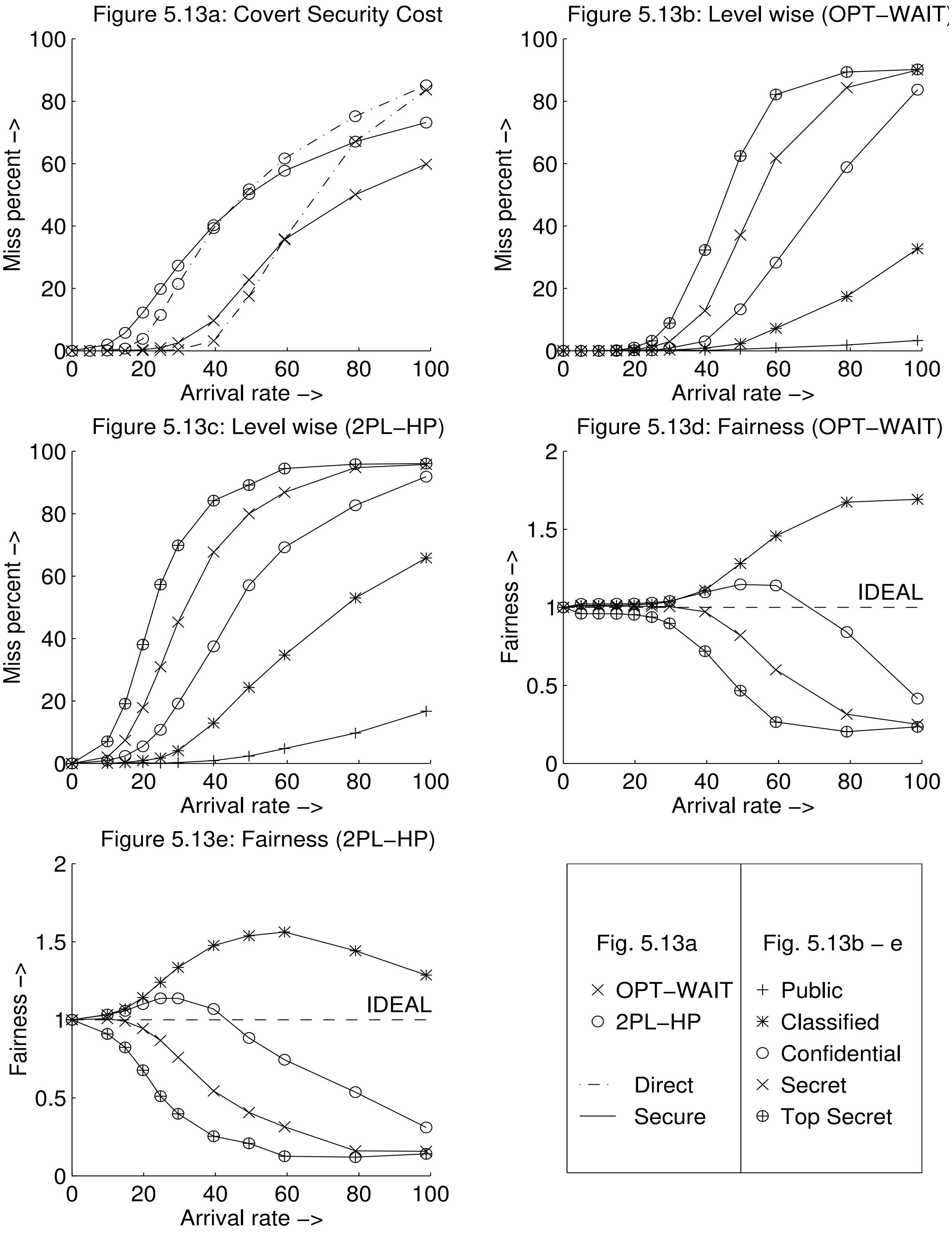


Figure 5.13: Increased security levels

In Figures 5.13b and 5.13c, we plot the miss percentage on a per-security-level basis for OPT-WAIT and 2PL-HP, respectively. These graphs clearly show the extent to which the kill percentages are skewed among the various transaction security levels, with Top Secret transactions having the most number of missed deadlines and Public transactions having the least. The graphs also show that OPT-WAIT's performance is better than that of 2PL-HP for *every* transaction security level.

In Figures 5.13d and 5.13e, the fairness factors for the top four security levels (Top Secret, Secret, Confidential and Classified) are plotted on a per-security-level basis for OPT-WAIT and 2PL-HP, respectively. These figures clearly show that as the loading factor increases, progressively more and more security classes become discriminated against by the lowest security class (Public). We also find that OPT-WAIT's performance is more fair than that of 2PL-HP for *every* transaction security level.

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

5.17 Conclusions

In this chapter, we have quantitatively investigated the performance implications of maintaining covert-channel-free security of CC protocols. Unlike previous studies, which used a *tradeoff* approach between security and timeliness, we have considered security as a correctness criterion and the number of missed deadlines to be a performance issue.

We first identified that, in order to satisy the requirement of non-interference, only those real-time CC protocols that are free from priority inversion can be used in a secure RTDBS. This requirement ruled out several previously proposed real-time CC protocols, including algorithms such as 2PL Wait Promote [4] and WAIT-50 [40]. We then studied the relative performance of the secure versions of the 2PL-HP, OPT-SACRIFICE, OPT-WAIT and S2PL real-time CC protocols. Our experiments showed that, under normal loads, the overall miss percent of the secure system is worse than that of the direct system, whereas under heavy loads, it is the other way around. Within the secure system, the performance of high-security transactions was significantly worse than that of the low-security transactions. Among the secure CC protocols, OPT-WAIT performed best in minimizing the kill percentages on both an overall basis and on a per-level basis. Moreover, it exhibited the maximum degree of fairness. These results show that OPT-WAIT, which provided excellent performance in traditional (unsecure) real-time CC, continues to perform well even in the secure real-time domain.

We also conducted several other experiments for developing better insight into the SRTDBS performance behavior. We evaluated the cost of enforcing direct security and concluded that there is no significant cost to be paid for ensuring direct security. In the experiment to assess the security risk because of non-pre-emptive nature of disks, we found that higher security class transactions interfere with low clearance level transactions opening the possibility for signaling channels. We also conducted a host of experiments on various real-time transaction models and workload conditions. In all the cases that we examined, **OPT-WAIT** was found to be the best among the compared protocols in terms of both kill percentage and fairness.

We also made the important observation that, for the RTDBS to be secure, *only inter class conflicts have to be resolved in a secure manner*. This opens up the possibility of obtaining better performance by utilizing any high-performance real-time CC protocol, whether secure or not, for resolving intra-class conflicts in combination with a secure CC mechanism for resolving interclass conflicts. We explore this possibility in the next chapter.

Chapter 6

Dual Approach

6.1 Introduction

In the previous chapter we dealt with the issues pertaining to secure real-time concurrency control and we found that while some of the real-time protocols are amenable to the secure environment, it is difficult to tailor all the protocols for the real-time environment. We also found that the secure version of OPT-WAIT out-performs all other secure real-time concurrency control protocols with which we experimented in terms of both kill percentage and fairness. In this chapter, our aim is to develop secure real-time concurrency control protocols that perform even better than OPT-WAIT, and for this we present a novel approach that is based on *simultaneously* employing *multiple* CC protocols. A performance study confirms that with this approach it is possible to outperform OPT-WAIT in terms of both KillPercent and fairness performance.

6.2 Dual Approach

As observed in the previous chapter, a feature of the secure real-time database environment is that there are *two* categories of data conflicts: *inter-level* and *intra-level*, as featured in Figure 6.1. Inter-level conflicts are data conflicts between transactions belonging to different security clearance levels whereas intra-level conflicts are data con-

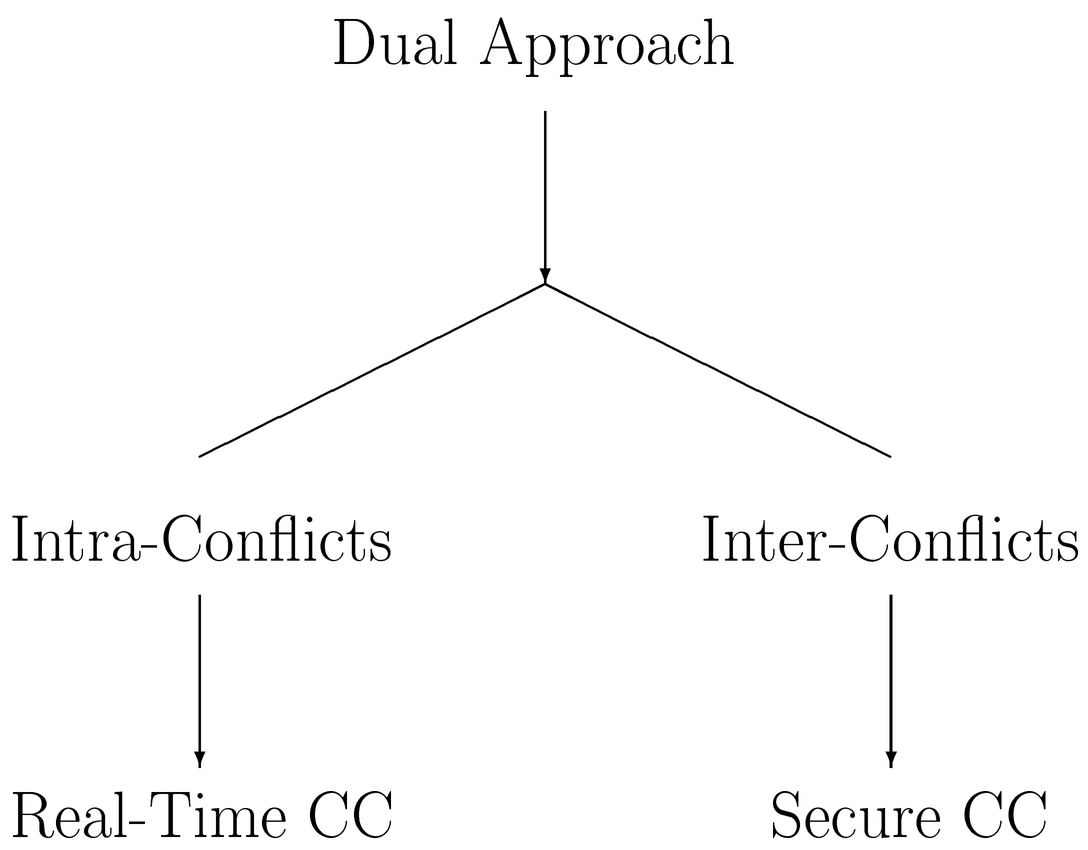


Figure 6.1: Dual approach

flicts between transactions of the same level. The important point to note here is that *only inter-level conflicts can result in security violations*, not intra-level conflicts. This opens up the possibility of using *different* concurrency control strategies to resolve the different types of conflicts. In particular, we can think of constructing mechanisms such that inter-level conflicts are resolved in a secure manner while intra-level conflicts are resolved in a timely manner. The advantage of this *dual* approach is that the RTDBS can maximize the real-time performance, by appropriate choice of intra-level CC protocol, without sacrificing security. In contrast, the tradeoff approach that has been suggested in [19] requires the application to *compromise* on security in order to achieve enhanced real-time performance. Another advantage of the dual approach is that the separation of security and timeliness concerns makes it possible to use even *non-secure* real-time CC algorithms (e.g., Wait-Promote, WAIT-50) for resolving intra-level conflicts! The dual approach therefore empowers the use, even in the secure RTDBS domain, of the rich set of real-time CC algorithms developed during the last decade.

6.3 Ensuring Serializability

At first glance, it may appear that using multiple concurrency control mechanisms in parallel could result in violation of the transaction serializability requirement. This could happen, for example, if the serial orders enforced by the individual mechanisms were to be different. A detailed study of a generalized version of this problem is presented in [99], wherein the transaction workload consists of a mix of transaction classes and the objective is to allow each transaction class to utilize its preferred concurrency control mechanism. They propose a database system architecture wherein intra-class conflicts are handled by the class's preferred concurrency control manager while inter-class conflicts are handled by a new software module called the Master Concurrency Controller (MCC) that interfaces between the transaction manager and the multiple concurrency control managers. The MCC itself implements a complete concurrency control mechanism. A single global serialization order is ensured in the entire database system by using a *Global Ordering Scheme*.

For our study, we adapt the above architecture to the secure real-time environment. In our implementation, each security level has a Local Concurrency Controller (LCC) that resolves intra-level data conflicts. The LCCs can use *any* high performance real-time concurrency protocol without risking security violations. The Master Concurrency Control mechanism, however, is chosen to be a secure protocol and is responsible for handling all inter-level data conflicts. Figure 6.2 shows an example of this implementation for a two security level system. The proof of correctness and other details of our adaptation of the MCC architecture are given in Appendix A.

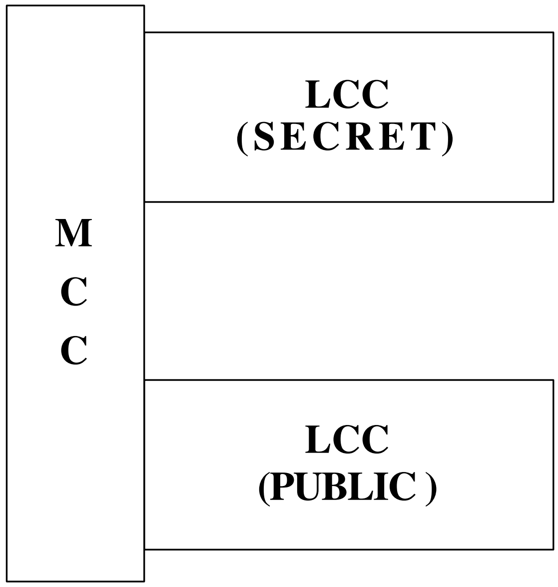


Figure 6.2: MCC architecture

Dual Algo.	Inter	Intra
WAIT-HP	OPT-WAIT	2PL-HP
HP-WAIT	2PL-HP	OPT-WAIT
S2PL-WAIT	S2PL	OPT-WAIT

Table 6.1: Dual Protocols

6.4 Dual Protocols

We developed and evaluated three dual protocols, summarized in Table 6.1, wherein inter-level conflicts are handled by one protocol while intra-level conflicts are handled by a different protocol:

1) WAIT-HP – inter-level data conflicts are resolved by OPT-WAIT and intra-level conflicts are resolved by 2PL-HP.

2) HP-WAIT – 2PL-HP is employed to handle the inter-level conflicts, whereas OPT-WAIT is used to resolve the intra-level conflicts.

3) S2PL-WAIT – once the intra-level conflicts are resolved by OPT-WAIT, the transaction enters the "virtual commit" state, and subsequently the inter-level data conflicts are resolved by S2PL. Once all conflicts are resolved, a transaction "really" commits.

Note that S2PL is a non-restart oriented algorithm, and it allows a high security level transaction to continue regardless of the virtual commit of a low security level conflicting transaction.

6.5 Performance of Dual Protocols

We conducted experiments to evaluate the performance of the abovementioned dual systems and in Figures 6.3a through 6.3c we show their performance for the same environment as that of the baseline experiment described in Chapter 5. For the sake of comparison, the performance of a pure OPT-WAIT protocol is also shown in these graphs.

Focusing our attention on the WAIT-HP and HP-WAIT dual protocols, we first observe in Figure 6.3a, which compares the overall kill percentages of the protocols, that the performance of both these approaches is considerably worse than that of the pure OPT-WAIT protocol. The reason that OPT-WAIT remains the best among them is that 2PL-HP is a wasteful algorithm, as seen in the previous experiments, and therefore "dilutes" the effect of OPT-WAIT in both the dual protocols.

We also observe in Figure 6.3a that the performance of WAIT-HP is worse than that of HP-WAIT throughout the loading range. The reason for this is that, in the secure system, the number of intra-level conflicts are significantly more than the number of inter-level conflicts. Therefore, the algorithm which is used to handle intra-level conflicts has more effect on the overall miss percentage than the protocol used to handle inter-class conflicts. In WAIT-HP, it is 2PL-HP which handles intra-level conflicts and this results in worse performance than that of HP-WAIT, which uses OPT-WAIT to handle this category of conflicts.

The kill percentages of the protocols on a per-security-level basis is provided in Figure 6.3b and the fairness factors are shown in Figure 6.3c. An interesting feature in these graphs is that at high loads, the fairness of WAIT-HP is *greater* than that of OPT-WAIT – this is the first time in all the experiments discussed so far that a real-time protocol has improved on OPT-WAIT's fairness performance. The reason that this happens is the following: In WAIT-HP, due to 2PL-HP being used for intra-level conflicts, many of the

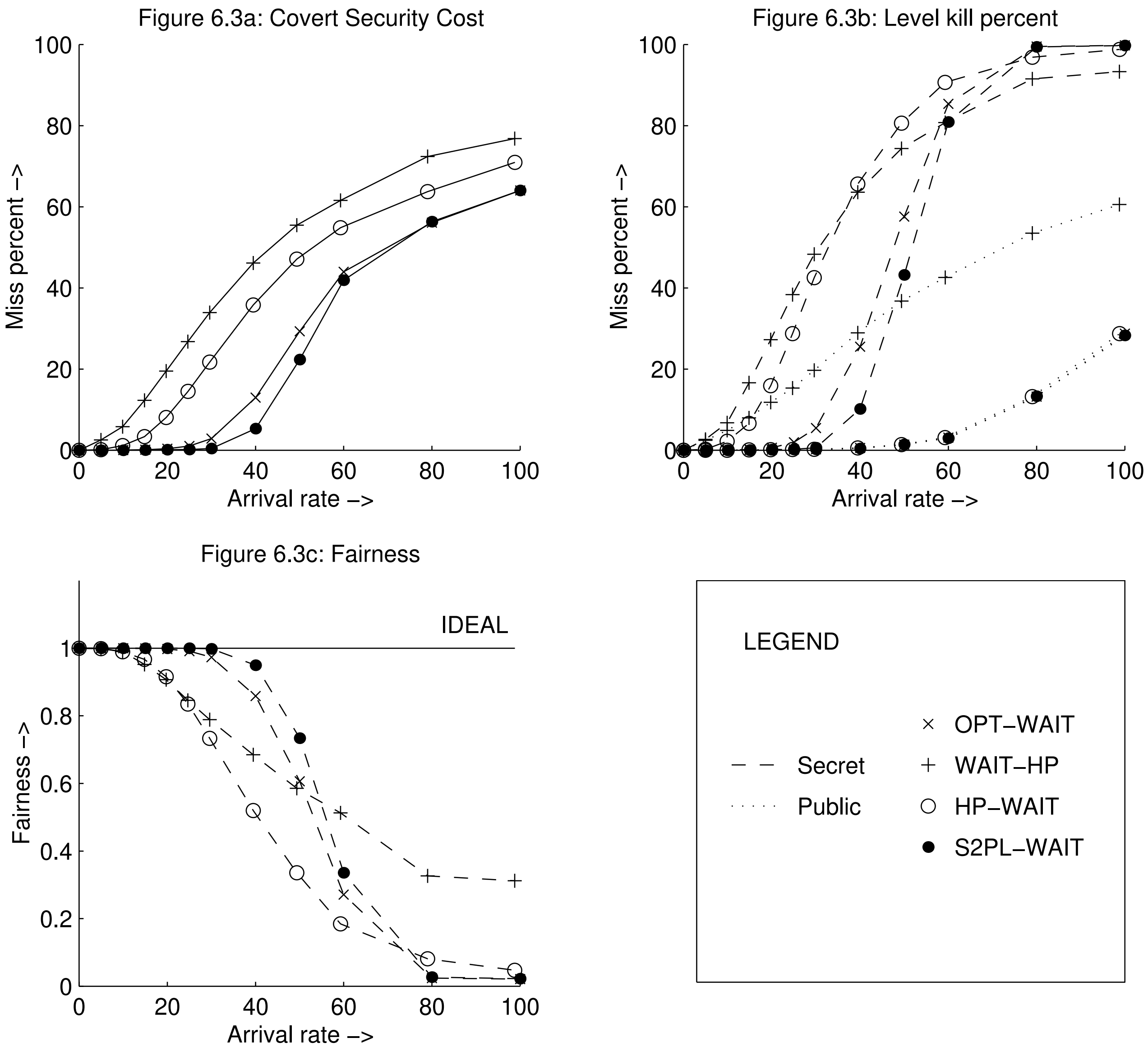


Figure 6.3: Performance of dual protocols

Public transactions are so busy "fighting" each other that they don't ever reach the end of their execution, which is when the OPT-WAIT policy of checking for inter-class conflicts comes into play. Therefore, Secret transactions suffer much less restarts from the Public transactions. This is clearly seen in Figure 6.3b where the kill percentage of the Public transactions for WAIT-HP is quite high as compared to the corresponding numbers for the other protocols.

Moving on to the S2PL-WAIT dual protocol, we find that, unlike the other two dual protocols, it performs better than OPT-WAIT, especially at lower loading levels. For example, at an arrival rate of 40 transactions per second, S2PL-WAIT more than halves the miss percentage suffered by OPT-WAIT (Figure 6.3a). The reason that this combination works well is that S2PL handles inter-class conflicts *without* resorting to restarts unlike in WAIT-HP or HP-WAIT. This means that Secret transactions suffer much less in this environment (as confirmed in Figures 6.3b and 6.3c). At the same time, using OPT-WAIT for handling intra-level conflicts helps to derive the inherent good real-time performance associated with this protocol. At high loads, S2PL-WAIT performs almost identically to OPT-WAIT because, in this region, the primary reason for transactions missing their deadlines is resource contention, rather than data contention – therefore, the virtual commit feature of S2PL rarely provides the intended benefits.

6.6 Other Experiments

In the above experiments, only secure intra-level real-time CC protocols were used. But, as mentioned earlier, the dual approach opens up the possibility of also utilizing unsecure real-time CC protocols for resolving intra-level conflicts. We conducted experiments to evaluate this possibility – in particular, we evaluated a combintation that used WAIT-50 for the intra-level CC mechanism and S2PL as the inter-level CC mechanism. The performance difference between the S2PL-WAIT combination and the S2PL-WAIT-50 combinations turned out to be very marginal and we have therefore not explicitly shown those results here. We expect, however, that by using more sophisticated real-time protocols such as OCC-TI [61], for example, in combination with S2PL, it may be possible to realize significant improvements.

6.7 Conclusions

We proposed a novel dual approach to secure CC wherein different concurrency control algorithms are used to resolve inter-level conflicts and intra-level conflicts. A global serialization order was ensured, in spite of having multiple CC algorithms operating simultaneously, by using the system architecture proposed in [99]. The dual combinations of HP-WAIT and WAIT-HP were implemented and evaluated – they both generally performed worse than pure OPT-WAIT. However, the dual combination of S2PL-WAIT performed better than OPT-WAIT, especially at lower miss percent levels. This is because S2PL is a non-restart-oriented algorithm unlike both OPT-WAIT and 2PL-HP, and therefore ensures reduction of the harm done to high-security transactions. Also, the dual approach empowers the use of the rich set of non-secure real-time CC algorithms in the secure domain. This highlights the power and flexibility that is provided by the dual approach.

In summary, our experiments show that by carefully choosing the right combination of protocols in the dual approach, we can design hybrid concurrency control algorithms that provide even better miss percent and fairness performance than OPT-WAIT.

Chapter 7

Secure Real-Time Buffer Management

7.1 Introduction

So far our focus was on the design of high-performance CC protocols for SRTDBS. We now shift our attention to an equally important transaction processing area, namely Buffer Management. In this chapter, we first identify the importance and difficulties of providing secure buffer management in the real-time environment. Then, we develop a new "dynamic" and "one-copy" buffer management policy, SABRE (Secure Algorithm for Buffering in Real-time Environments), that is demonstrably secure and designed to provide good real-time performance. Finally, we quantify the performance degradation that occurs due to having to provide security in the buffer manager.

7.1.1 Design Challenges

Buffer managers open up possibilities for covert channels, in fact, *many more than those associated with CC*. For example, the presence or absence of a delay in acquiring a free buffer slot, or the presence or absence of a specific page in the buffer set, or the allocation of a particular (physical) buffer slot, could all be used as channel mediums, whereas in CC, data access time is the primary medium.

Apart from the multitude of channel mediums described above, there are several additional problems that arise while integrating security into the buffer manager. For example, buffer managers need to support the notion of a "pin" which ensures the physical consistency of a buffer page while it is being processed by a transaction, as also the notion of a "force write" wherein the contents of a buffer page are forced to the disk on demand. Further, unlike CC, which essentially deals only with regulating data access, a buffer manager has *multiple* components – *buffer allocation*, *buffer replacement* and *pin synchronization* [35] as explained in Chapter 2.

In summary, due to all of the above reasons, making a real-time buffer manager implementation secure involves significant design complexity.

7.1.2 Buffer Categories

At any given time, the slots in the buffer pool can be grouped into the following four categories: **Pinned** – Buffer slots containing valid pages that are currently being accessed (in read or write mode) by executing transactions; **Active** – Buffer slots containing valid pages that have been earlier accessed by currently executing transactions; **Dormant** – Buffer slots containing valid pages that have not been accessed by any currently executing transaction (these pages were brought in by previously completed transactions); **Empty** – Buffer slots that are empty. Further, the first three categories (Pinned, Active, and Dormant) can be further subdivided into **clean** and **dirty** groups, which contain the set of clean and dirty (i.e. modified) pages, respectively, of that category.

7.1.3 Pin Pre-emption

In addition to the traditional buffer management components [35] of allocation, replacement and pin synchronization, an additional component that arises specifically in the secure real-time domain is *pin preemption.* Consider the case, for example, where a low clearance transaction requests buffer space but the slots currently held by high clearance transactions which would normally be candidates for replacement cannot be forced out because they are all pinned. In this case, one option is for the low clearance transaction to wait for slots to become unpinned but this would immediately result in covert channel possibilities. Therefore, there appears to be no choice but to "break the pin", that is, to permit the replacement with the proviso that the high clearance transaction holding the original pin is immediately notified that the page it had assumed to be stable had actually been replaced – the high clearance transaction can then either abort or, preferably, redo the corrupted operation at a later time.

Security considerations are not the only reason for supporting pin preemption. Consider, for example, the case where a tight-deadline transaction wishes to utilize the slot currently pinned by a slack-deadline transaction. If the urgent transaction is blocked, this would mean that high priority transactions are being blocked by low priority transactions, that is, resulting in *priority inversion* [84]. Priority inversion can cause the affected high-priority transactions to miss their deadlines and is clearly undesirable. Therefore, pin preemption is useful from *real-time* considerations also.

7.2 Buffer Management Policies

In this section, we present the set of buffer management policies evaluated in our study, which cover the spectrum from unsecure non-real-time policies to fully secure real-time policies.

7.2.1 The SABRE Policy

Our new **SABRE** policy provides complete covert-channel-free security. Unlike the policies proposed in [103], SABRE is *fully dynamic* since there is *no preallocation* of buffer slots to security levels, and *unconstrained* in that no restrictions are placed on high clearance transaction accesses. Further, only *one copy* of a data page is maintained in the buffer pool. These features ensure efficient use of buffer resources. Finally, it incorporates several optimizations both for reducing the overall number of killed transactions and for decreasing the unfairness in the distribution of killed transactions across clearance levels. We describe its design in more detail below.

1)Security Features

Merely assigning priorities in the manner described in Section 4.1.7 is necessary, but not *sufficient*, however, to make the buffer manager secure in terms of the non-interference requirement. We describe below the main additional features incorporated in SABRE to ensure covert-channel-free security.

1. The contents of a buffer slot are not "visible" to a transaction if the existence of the page in the slot is a consequence of actions performed by higher clearance transactions. Such a slot will appear as a *(pseudo)-empty* slot to the requesting transaction.

   More specifically, for a transaction of a specific clearance level, all the buffer pages of all higher clearance levels are not visible whereas the Pinned and Active buffer pages of its own level and all lower clearance levels are visible (the visibility of Dormant pages is discussed below in Feature 3). This means that, for example, a low clearance transaction is not given immediate access to a page that has been earlier brought into the buffer pool by a high clearance transaction. Instead, it is forced to wait for the same period that it would have taken to bring the page from disk had the page not been in the buffer pool (the determination of this period is discussed in Section 7.2.1).

2. High clearance transactions can replace the Dormant slots of low clearance transactions.

3. Dormant pages, of *any* level, are visible only to transactions of the *highest* clearance level – for no other transactions are they visible. That is, a low clearance transaction is not given immediate access to a Dormant page even at its own or lower clearance levels. Instead, it is forced to wait for the same period that it would have taken to bring the page from disk had the page not been in the buffer pool.

4. Priority-based pin preemption is supported at all clearance levels.[1]

[1]Supporting pin preemption at the *lowest* clearance level is not essential for security but is retained for performance reasons (see Section 7.2.1).

5. When selecting from among the set of (really) Empty slots, the slot is *randomly chosen* and not in a pre-defined order.

Feature 1 above is an obvious requirement to ensure the absence of signaling between high clearance and low clearance transactions. Feature 2, which allows high clearance transactions to "steal" the Dormant slots of low clearance transactions, is included for the following reason: If high clearance transactions could not replace *any* pages of low clearance transactions, then the buffer pool would very quickly fill up with the Active and Dormant pages of the low clearance transactions and after this the high clearance transactions would not be able to proceed further. That is, *starvation* of high clearance transactions would occur.

Note, however, that there is a price to pay for ensuring that covert channels do not result in spite of slot-stealing. This is expressed in Feature 3 wherein low clearance transactions are made to wait for access to Dormant pages at even their *own and lower* clearance levels, thereby partially losing the benefits that could be gained from inter-transaction locality. In a multi-level system, only the *highest level* transactions should be allowed to see the dormant slots. Otherwise, security violations can occur as the following example shows: Consider a three level system (Secret > Classified > Public). Here if the Classified transactions are allowed to see the Public dormant slots, a *transitive* covert channel can arise because Classified transactions see Secret transactions replacing the Public dormant slots.

Feature 4 ensures that high clearance accesses can be unrestricted without causing covert channels since borrowed slots can be returned immediately even if they are currently pinned. Finally, Feature 5 ensures that low clearance transactions cannot "guess" that they are being given a pseudo-empty slot by virtue of the fact that the slot they receive is not the first in the list of slots that they perceive to be empty.

We show below that the above design guarantees covert-channel-free security.

Proof:

In this proof, we consider the various kinds of security violations that are possible with regard to Delay Security, Value Security and Recovery Security, and show that none of these can occur with the SABRE algorithm.

Delay Security: From a transaction's viewpoint, delay security can be violated in five different ways: (1) A page is known to be absent in the buffer when it is expected to be present; (2) A page is known to be present in the buffer when it is expected to be absent; (3) A buffer slot is known to be available when the buffer is expected to be full; (4) A buffer slot is known to be unavailable when the buffer is expected to be empty; (5) A physical slot is known to be allocated when some other physical buffer slot is expected to be allocated.

The first possibility, a page is absent in the buffer when it is expected to be present, can occur only if a higher level transaction replaces the page. Our protocol disallows a high security transaction from replacing the page brought into the buffer by a low security transaction.

The second possibility, a page is present in the buffer when its expected to be absent, can happen only if a higher security level transaction brings in the page to the buffer and the low security level transaction is given access to the page. Our protocol disallows this possibility by setting a visibility level. The pages brought in by higher security level transactions are not visible to low security level transactions.

The third possibility, a buffer slot is known to be available when the buffer is expected to be full, can happen only if higher security transactions can free some buffer slots. This cannot happen since, by the visibility criterion, if the buffer pool appears to be full to a low security transaction, then all buffers must be currently held by low security transactions, and such buffers cannot be replaced by high-security transactions.

The fourth possibility, a buffer slot is known to be unavailable when the buffer is expected to be empty, can happen if the buffers slots are held by higher security level transactions. Our protocol provides the facility to avail the slots occupied by higher security level transactions by providing the pin pre-emption facility.

The fifth possibility is prevented by randomizing the slot identities.

Value Security: Value Security can be violated if and only if a high security level transaction controls the order of operations performed on a data item. According to our protocol, the only mechanism by which the above violation is possible is a high security level transaction delaying one or more operations with respect to others such that the value read atleast by one transaction is changed. For value security violations to occur, it must be possible for a transaction T_i to control the value read by at least one transaction T_j such that $SL(T_i) < SL(T_j)$, where $SL(T)$ is the security level of transaction T. This can happen only if T_i could delay T_j or another transaction T_k such that the value written by T_k is either directly or transitively **readable** by T_j. That means $SL(T_k) \leq SL(T_j)$. Since our protocol guarantees delay security this cannot happen.

Recovery Security: Our protocol does not allow a high security transaction to preempt the buffer slot from a low security transaction. It is always a high security transaction that is restarted when a low security transaction conflicts with a high security transaction. Therefore, the protocol ensures Recovery Security.

$\square$

2)Real-Time Features

SABRE incorporates the following features to enhance its *real-time* performance:

1. Transactions of a particular clearance level cannot replace the Pinned or Active pages of higher priority transactions belonging to the same level.

2. Within each clearance level, priority-based pin preemption is supported.

3. An optimized "comb" slot selection algorithm to decide the slot in which to host a new data page.

Feature 1 helps to utilize the intra-transaction and inter-transaction locality of high priority transactions, while Feature 2 ensures the absence of priority inversion. The optimized slot selection algorithm mentioned in Feature 3 is described below in Section 7.2.1.

3)Search and Slot Selection

```
if (Really Empty Slots exist) then
     slot = randomly chosen Empty Slot
else if (Dormant Slots exist) then
     /* Slot Stealing from Lower Security Levels OR
        Slot Confiscating from Higher Security Levels */
     find the lowest security level at which Dormant Slots exist
     slot = LRU (clean|dirty) among these Dormant Slots
else if (Slots of Higher Security Levels exist) then
     /* Slot Confiscating from Higher Security Levels */
     find the highest level above requester's level for
                                             which slots exist
     if (Active Slots exist) then
          find the lowest priority transaction with Active Slots
          slot = LRU (clean|dirty) among these slots
     else /* All Slots are Pinned */
          abort lowest priority transaction and release its slots
          slot = LRU (clean|dirty) among these slots
else if (Slots of Requester's Level exist) then
     if (Lower Priority Transactions with Active Slots exist) then
          find the Active Slots of lowest priority transaction
          slot = LRU (clean|dirty) among these slots
     else if (Lower Priority Transactions with Pinned Slots exist) then
          abort the lowest priority transaction and release its slots
          slot = LRU (clean|dirty) among these slots
     else /* All Slots belong to Higher Priority transactions */
          insert request in the wait queue which is
          maintained in (LEVEL, DEADLINE) priority order
else /* All Slots belong to Lower Security Level transactions */
     insert request in the wait queue which is
     maintained in (LEVEL, DEADLINE) priority order
```

Figure 7.1a: SABRE Slot Selection Algorithm

When a transaction requests a data page, the *visible* portion of the buffer pool (corresponding to the transaction's clearance level) is searched for the page and if the search is successful, SABRE returns the address of the slot in which the page is present. If the page is already pinned in a conflicting mode by a higher priority transaction, then the transaction has to wait for it to be unpinned before accessing the contents. Otherwise, it can access the page immediately after gaining a pin on the page.

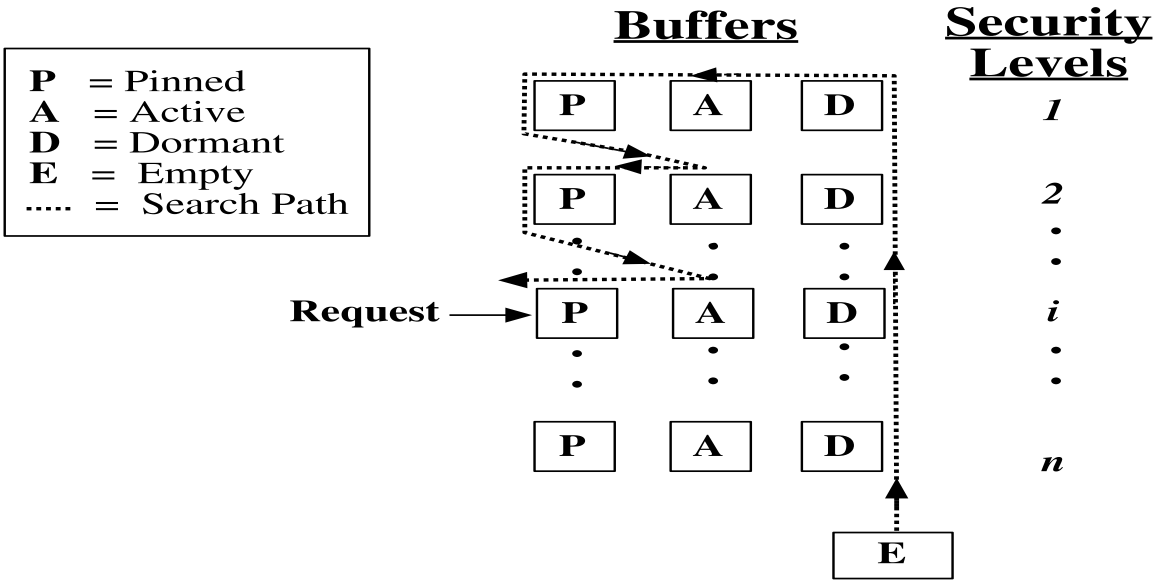


Figure 7.1b: COMB Slot Selection Route

A search could be unsuccessful for one of two reasons: (a) Because the page is really not in the buffer pool, or (b) Because it is in the non-visible portion of the buffer pool. In the first case, if really empty buffer slots are available, SABRE brings the page into one of these slots and returns the address of the slot to the requesting transaction. Otherwise, an existing buffer page is chosen for replacement according to the selection algorithm described below. The victim page, after being flushed to disk if dirty, is replaced by the requested page and the associated slot address is returned to the transaction. In the second case, the requested page is made to *appear* to have been brought from disk into the buffer slot where it currently exists by "unveiling" the slot only after waiting for the equivalent disk access time.

The slot selection algorithm used in SABRE is shown in Figure 7.1a. In this algorithm, starting from really Empty slots, the selection works its way up the security hierarchy looking for Dormant slots and then in a "comb-like" fashion works its way downwards from the top clearance level looking for Active and Pinned slots until it reaches the requester's clearance level. This route is shown pictorially in Figure 7.1b.

SABRE's comb slot selection algorithm incorporates the security and real-time features described earlier. In addition, it includes the following optimizations designed to improve the real-time performance:

1. The search for replacement buffers is ordered based on slot category – the Dormant slots are considered first and only if that is unsuccessful, are the Active and Pinned slots considered. This ordering is chosen to maximize the utilization of *intra-transaction* locality.

2. Given a set of candidate replacement slots (as determined by the slot selection algorithm), these candidate slots are grouped into clean and dirty subsets – only if the clean group is empty is the dirty group considered. Within each group the classical **LRU** (Least Recently Used) policy [27] is used to choose the replacement slot. This "LRU (clean|dirty)" ordering is based on the observation that (a) it is faster to replace a clean page than a dirty page since no disk writes are incurred, and (b) the cost of writing out a disk page is amortized over a larger number of updates to the page.

4)Fairness Features

As mentioned earlier, due to the covert-channel-free requirements, high clearance transactions in SRTDBS are discriminated against in that they usually form a disproportionately large fraction of the killed transactions. To partially address this issue, the following optimizations are incorporated in SABRE:

(a) Proxy Disk Service: The delays prescribed in Security Features 1 and 3 (Section 7.2.1) can be easily implemented by actually retrieving the desired page from disk into the buffer slot where it already currently exists. Obviously, this approach is wasteful. A more useful strategy would be to use the disk time intended for servicing this request to instead serve the pending requests, if any, of higher clearance transactions for the same disk. That is, the high clearance transaction clandestinely acts as a "proxy" for the low clearance transaction.

(b) Comb Route: In the Comb algorithm, the search for a replacement slot among the Dormant slots begins with the *lowest* clearance level. This approach maximizes the possibility of slot-stealing from lower clearance levels, resulting in increased fairness.

7.2.2 Unsecure Buffer Policies

Since, to our knowledge, SABRE represents the *first* secure real-time buffer management policy, we have compared its performance with that of representative *unsecure* buffer management policies – in particular, **CONV** and **RT**, policies for conventional DBMS and for RTDBS, respectively. We do not compare our protocols with the protocols described in [103], because their static protocols require resource reservations while their dynamic protocols are vulnerable to covert channels.

```
if (Empty Slots exist) then
     slot = any Empty Slot
else if (Dormant Slots exist) then
     slot = LRU (clean|dirty) among these Dormant Slots
else if (Active Slots exist) then
     slot = LRU (clean|dirty) among these Active Slots
else  /* all slots are pinned */
     insert request in the wait queue which is
     maintained in FCFS order
```

Figure 7.2: The CONV Slot Selection Algorithm

For the CONV and RT policies, the *entire* buffer pool is always visible since they are not security-cognizant. In CONV, for a successful search, the requesting transaction can access the page immediately unless it has already been pinned in a conflicting mode – in this case the transaction has to wait for the page to be unpinned before accessing its contents. RT also follows a similar policy for successful searches except that in the case of conflicting pins, if the pin holder is of lower priority than the requesting transaction, the pin is preempted by aborting the lower priority holder.

For unsuccessful searches, CONV and RT follow the slot selection algorithms shown in Figures 7.2 and 7.3, respectively. CONV essentially implements the classical LRU

```
if (Empty Slots exist) then
     slot = any empty slot
else if (Dormant Slots exist) then
     slot = LRU (clean|dirty) among the Dormant Slots
else if (Lower Priority Transactions with Active Slots exist)
     find the active slots of the lowest priority transaction
     slot = LRU (clean|dirty) among these slots
else if (Lower Priority Transactions exist)
     abort the lowest priority transaction and release its slots
     slot = LRU (clean|dirty) among these slots
else  /* all slots are with higher priority transactions */
     insert request in the wait queue which is
     maintained in DEADLINE priority order
```

Figure 7.3: The RT Slot Selection Algorithm

approach, including also the (clean|dirty) and category-based search optimizations of SABRE, while RT implements a prioritized version of the same approach.

7.2.3 Baseline Policies

To isolate and quantify the effects of buffer management on the system performance, we also evaluate two artificial baseline policies, **ALLHIT** and **ALLMISS**, in the simulations. ALLHIT models an ideal system where every page access results in a buffer pool "hit" , while ALLMISS models the other extreme – a system where every page access results in a buffer pool "miss".

7.2.4 Concurrency Control

For CC, we use the 2PL–HighPriority [4] protocol. Note that in our study of CC protocols described in the previous chapters, we had found S2PL-WAIT to provide the best real-time performance. However, since complex open problems remain with respect to integrating validation schemes with buffer management [38], we have opted to use 2PL-HP instead of choosing some unrealistic and artificial integration scheme which would greatly reduce the utility of our experimental results.

7.3 Experiments and Results

Using the simulation model discussed in Chapter 4, we evaluated the performance of the various buffer managers for a variety of security-classified transaction workloads and system configurations. The settings of the workload and system parameters for these experiments are listed in Table 4.3. These settings were chosen to ensure significant intra-transaction and inter-transaction locality in the reference patterns and also significant buffer contention, thus helping to bring out the performance differences between the various buffer management policies.

7.4 Overview of Experiments

In our first experiment, we profile the performance of the BM policies where both intra transaction locality and inter transaction locality are present in the workload. In the following set of experiments, we assess the performance behavior when there is no explicit locality, there is only inter-transaction locality, and there is only intra-transaction locality, respectively. While these experiments are reported for a two-security-level system, in our final experiment, we evaluate the performance behavior in the context of a five-security-level system.

7.5 Experiment 1: Real-Time Security Experiment

In our first experiment, we evaluate the cost of providing covert-channel-free security in the real-time environment for a system with two security levels: *Secret* and *Public*. For this configuration, Figure 7.4a shows the overall KillPercent behavior as a function of the (per-second) transaction arrival rate for the SABRE, CONV, RT, ALLHIT and ALLMISS buffer policies. We observe here that, as expected, the ALLHIT and ALLMISS baseline policies exhibit the best and worst performance, respectively. What is more interesting is the *vast gap* between the performance of ALLHIT and that of ALLMISS indicating the extent to which intelligent buffer management can play a role in improving system

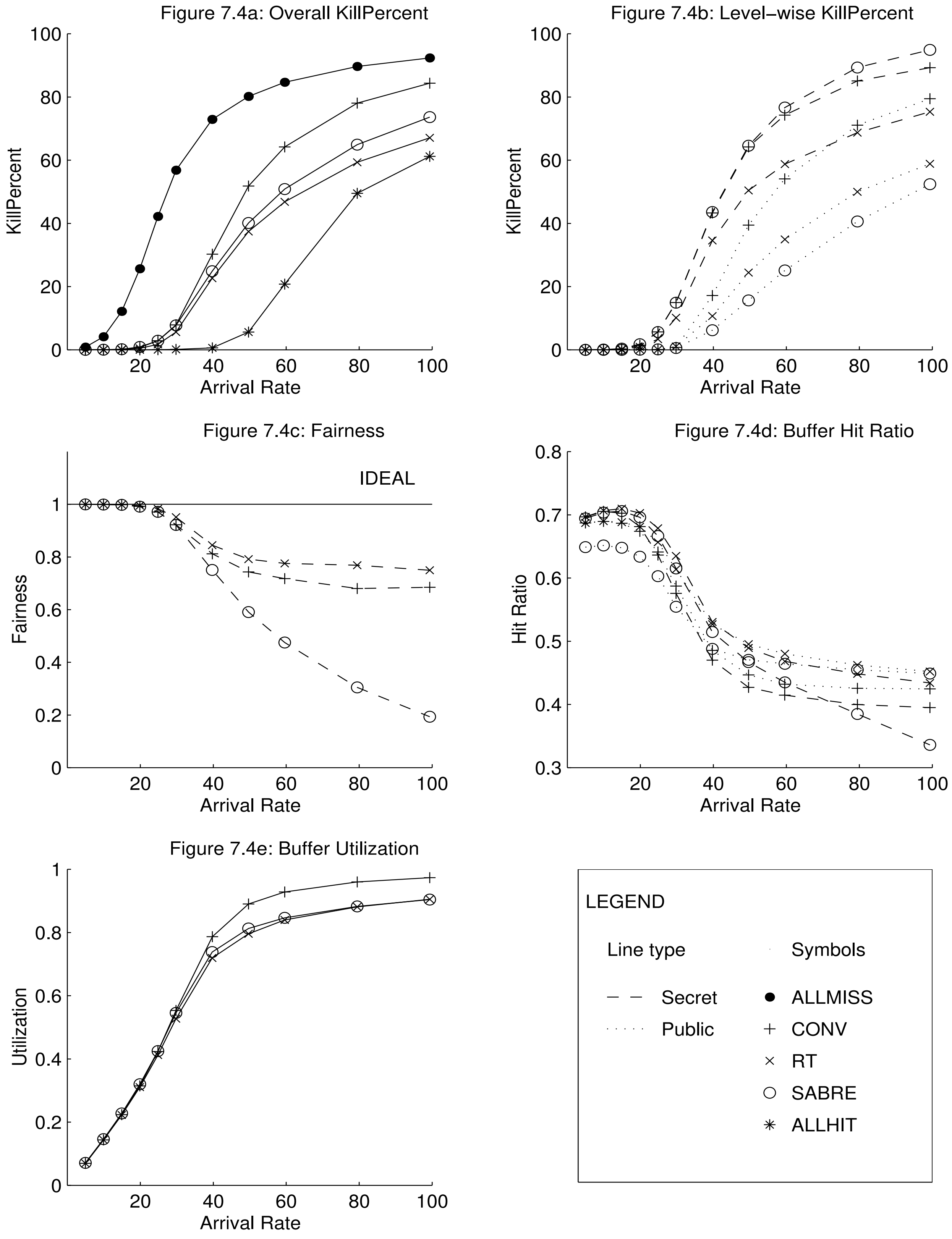


Figure 7.4: Real-Time security experiment

performance and highlighting the need for well-designed buffer policies.

Among the practical protocols, we observe that the real-time performance of SABRE and RT is similar, with RT holding the edge, whereas CONV is significantly worse, especially under heavy loads. RT shows better performance than CONV because it gives preferential treatment to urgent transactions in the following ways: (1) They derive more *intra-transaction* and *restart* locality since their Active pages cannot be replaced by low priority transactions; and (2) They get "first pick" of the available slots due to the prioritized queuing for buffer slots.

The reason that SABRE performs slightly worse than RT is that since priorities are decided primarily based on clearance levels, it is still possible for a slack-deadline Public transaction to restart a tight-deadline Secret transaction, resulting in the Secret transaction missing its deadline. This effect is highlighted in Figure 7.4b, which shows the *level-wise* kill percentages – with SABRE, Secret transactions (dashed lines) form a much larger proportion of the killed transactions. This is further quantified in Figure 7.4c, which captures the ClassFairness metric, and clearly demonstrates that SABRE is extremely unfair to Secret transactions, especially under heavy loads.

It may appear surprising in Figures 7.4b and 7.4c that RT and CONV, although not security-cognizant, are still somewhat unfair to Secret transactions. This behavior is not due to the buffer policies themselves, but is a *side-effect* of the rest of the system being secure and therefore discriminating against Secret transactions, resulting in more Public page requests being received at the buffer pool.

In Figure 7.4d, we show the *buffer hit ratio*, that is the average probability of finding a requested page in the visible portion of the buffer pool, for the various protocols on a level-wise basis. The first point to note is that in all the protocols there is a *crossover* between the Secret and Public hit ratios – at low loads Secret is better whereas at high loads Public is better. This is explained as follows: In CONV and in RT, the low load hit ratios should have been similar since they are not security cognizant, but again there is a side-effect arising out of the rest of the system being secure – more Secret transactions are restarted for *CC* reasons than Public transactions and therefore there is more *restart locality* derived for Secret transactions. Under heavy loads, however, this effect is overtaken by the Public transactions hogging most of the buffer slots and thereby making the Secret transactions hot spot mostly absent from the buffer pool.

In SABRE, the Secret transactions do better at low loads, not only because of the restart locality mentioned above, but also due to two additional factors: (1) the whole buffer pool is visible to Secret transactions; and (2) the "slot stealing" feature. Under heavy loads, however, *even more* of the buffer slots are occupied by Public transactions as compared to CONV and RT since Secret transactions cannot replace the Active pages of Public transactions, and therefore the Secret hit ratio is markedly worse. Another interesting observation with respect to SABRE is that although its design prevented the use of much of the Dormant page locality (as described in Section 7.2), yet its overall hit ratio is not materially worse than that of CONV or RT. This is because, except under very light loads, most of the pages in the buffer pool will be Active since they represent the hot spot and therefore *Active page locality predominates.*

Finally, Figure 7.4e shows the *buffer utilization*, that is, the overall average number of Pinned buffer slots, and it is clear here that CONV utilizes the buffers more than RT and SABRE. However, this does not result in better real-time performance because CONV is deadline-indifferent whereas RT and SABRE selectively give the Pinned slots to tight-deadline transactions.

In summary, the results of this experiment highlight the benefits of using deadline-cognizant buffer management policies. Further, it shows that SABRE provides security with only a modest drop in real-time performance. The non-interference requirement, however, causes SABRE to be rather unfair to high clearance transactions, especially under heavy loads.

7.6 Experiment 2: No Locality

Having profiled the real-time performance of buffer managers under conditions of significant locality and significant buffer contention, we turn our attention to their performance characteristics in the situation when *no* reference locality is present in the data access

Table 7.1: No Locality

InterLoc	0.0
IntraLoc	0.0

pattern. The settings of the workload and system parameters for the experiment are listed in Table 4.3. Note that all parameters remain the same as that of Experiment 1 except the *InterLoc* and *IntraLoc* parameters which are both set to 0 in this experiment to indicate the absence of both inter-transaction locality and intra-transaction locality (see Table 7.1).

Figure 7.5a shows the overall KillPercent behavior as a function of the transaction arrival rate. At low loads there is no noticeable difference in performance between the policies. At high loads, when there is buffer contention, RT performs marginally better than CONV, because of the priority cognizant buffer allocation policy followed in RT. SABRE provides performance that is in between CONV and RT.

In Figure 7.5b, the level-wise break-up of miss percentages is given. As expected, the secret class performance in SABRE is worse than its Public class performance. This is further evident from the fairness characteristics plotted in Figure 7.5c. This fairness measure quantifies the extent to which Secret class transactions are discriminated against in the system with respect to an ideal fairness factor of 1.0. We observe that at low loads when virtually all transactions make their deadlines, all the buffer management policies are fair. However, as the load increases, all the policies become increasingly unfair. RT exhibits the best fairness and SABRE the worst. The poor fairness in SABRE can be attributed to its enforcement of non-interference conditions for security. As described in Experiment 1, the other two protocols (CONV and RT) are also unfair due to the side effects caused from the rest of the system.

Turning our attention to Figure 7.5d, where the buffer hit ratios are shown, all the buffer managers show almost identical hit ratio characteristics, close to the theoretically-expected hit ratio of 0.05 (since there are 1000 pages in the database and 50 buffers in the system). Finally, Figure 7.5e shows the buffer utilization. Similar to Experiment 1, we

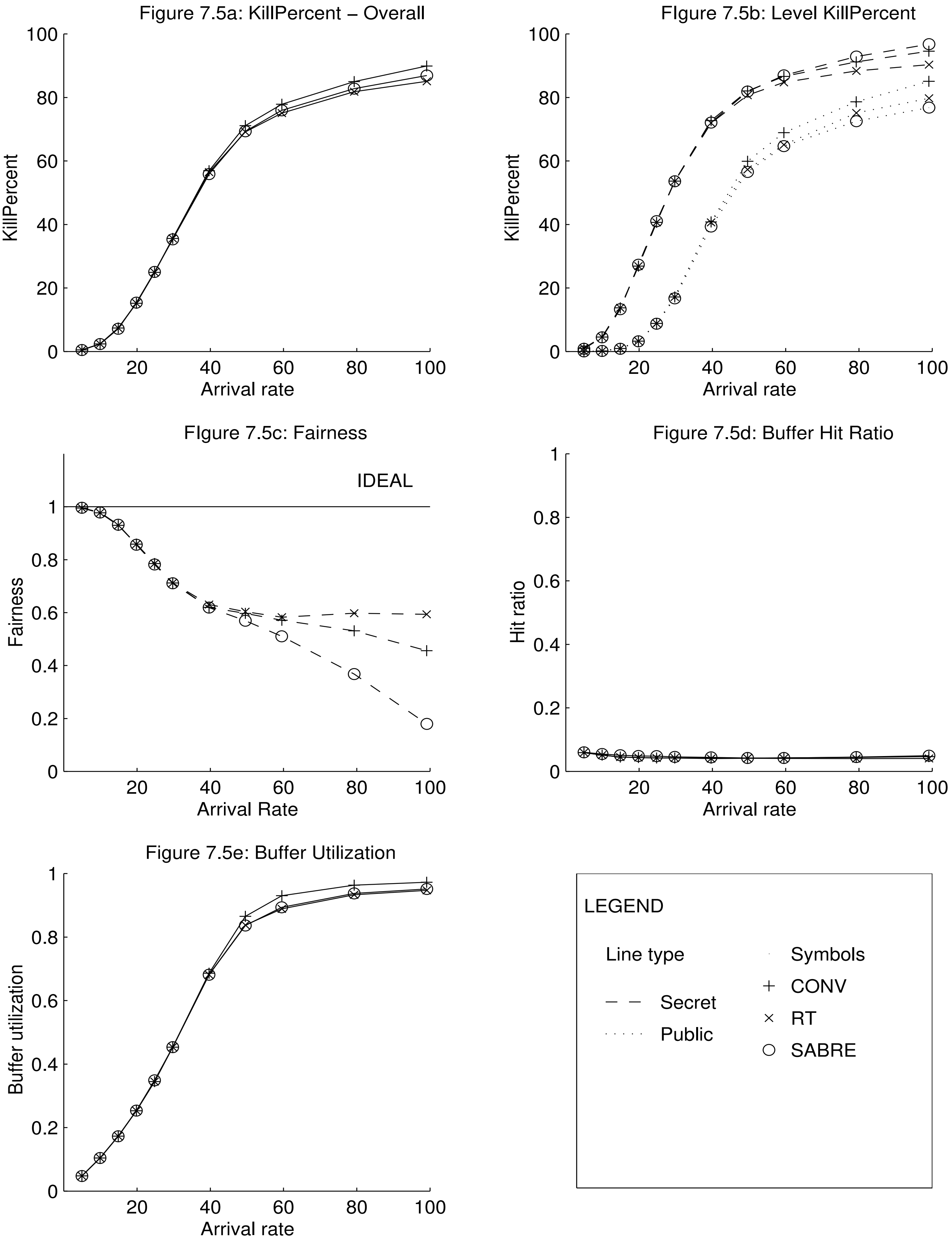


Figure 7.5: No Locality

Table 7.2: Inter-Transaction Locality

InterLoc	0.14
IntraLoc	0.0

see that although CONV utilizes more buffers it does not result in real-time performance improvement.

In summary, the SABRE buffer management policy ensures security with a very small drop in real-time performance in the no locality case. However, SABRE is increasingly unfair at higher loads due to having to satisfy non-interference constraints.

7.7 Experiment 3: Inter-Transaction Locality

The goal of this experiment is to study the impact of *inter* transaction locality on the system performance. All parameters remain the same as the parameters chosen in the No Locality Experiment, except that the the parameter *InterLoc* is set to 0.14 to introduce inter transaction locality in the workload (see Table 7.2). The performance characteristics obtained are shown in Figures 7.6a–e.

In Figure 7.6a, the overall kill percentages of all the buffer managers are given. We see here that at normal loads all the BM protocols perform almost identically but at higher loads, CONV performs the worst whereas RT provides the best performance and SABRE performs almost as well as RT. Note that, surprisingly, the miss percentage values in this experiment are *higher*, for all policies, than their respective kill percentage values in the No Locality experiment despite the fact that the hitratio is much higher here (see Figure 7.6d). The reason for this behavior are the hot spots resulting from the global skew in the data accesses. These hot spots *increase* data conflicts and, consequently, increase the number of deadline misses.

The level wise split-up of kill percentage values is given in Figure 7.6b. The Secret kill percentage for RT is less than that of CONV due to its priority-cognizant, but security indifferent, characteristic. Turning our attention to the Public miss percentages, CONV shows the worst kill percentage at high loads whereas the RT Public kill percentage is lower due to its priority-cognizance. Public transactions in SABRE have significantly better performance than their Secret counterparts. This is because the Public transactions do not "feel" the secret transactions in the buffer, due to the non-interference feature of SABRE.

The fairness characteristics shown in Figure 7.6c indicate that SABRE is highly unfair at high loads. This happens because most of the buffer slots will be made either Active or Pinned by the Public transactions, thereby leaving little room for Secret transactions in the buffer.

The buffer hit ratio characteristics are given in Figure 7.6d. At low loads, the hit ratios are rather high for all buffer managers. As the load increases, however, the hit ratio curves descend and reach an almost constant value. The fall is due to the partial loss of hot spots from the buffer due to more and more transactions bringing non-hot-spot pages into the buffer. Also, at low loads more restart transaction locality is derived for transactions and this contributes to the hit ratio.

Though CONV utilizes the buffers more than RT and SABRE (Figure 7.6e), this does not result in better real-time performance because RT and SABRE selectively give the pinned slots to tight-deadline transactions whereas CONV is deadline-indifferent.

To conclude, the inter transaction locality in the workload gives rise to increased restarts and thereby reduced real-time performance. Though the real-time features of RT and SABRE make them marginally better than CONV in terms of kill percentage performance, the improvement is relatively minor.

7.8 Experiment 4: Intra-Transaction Locality

This experiment was conducted to isolate the effect of *intra* transaction locality on the system performance. The parameter *IntraLoc* is set to 0.8 (see Table 7.3). The remaining parameter values are the same as those used in the No Locality experiment (Experiment 2).

The results of this experiment are shown in Figures 7.7a-e. We first note that the

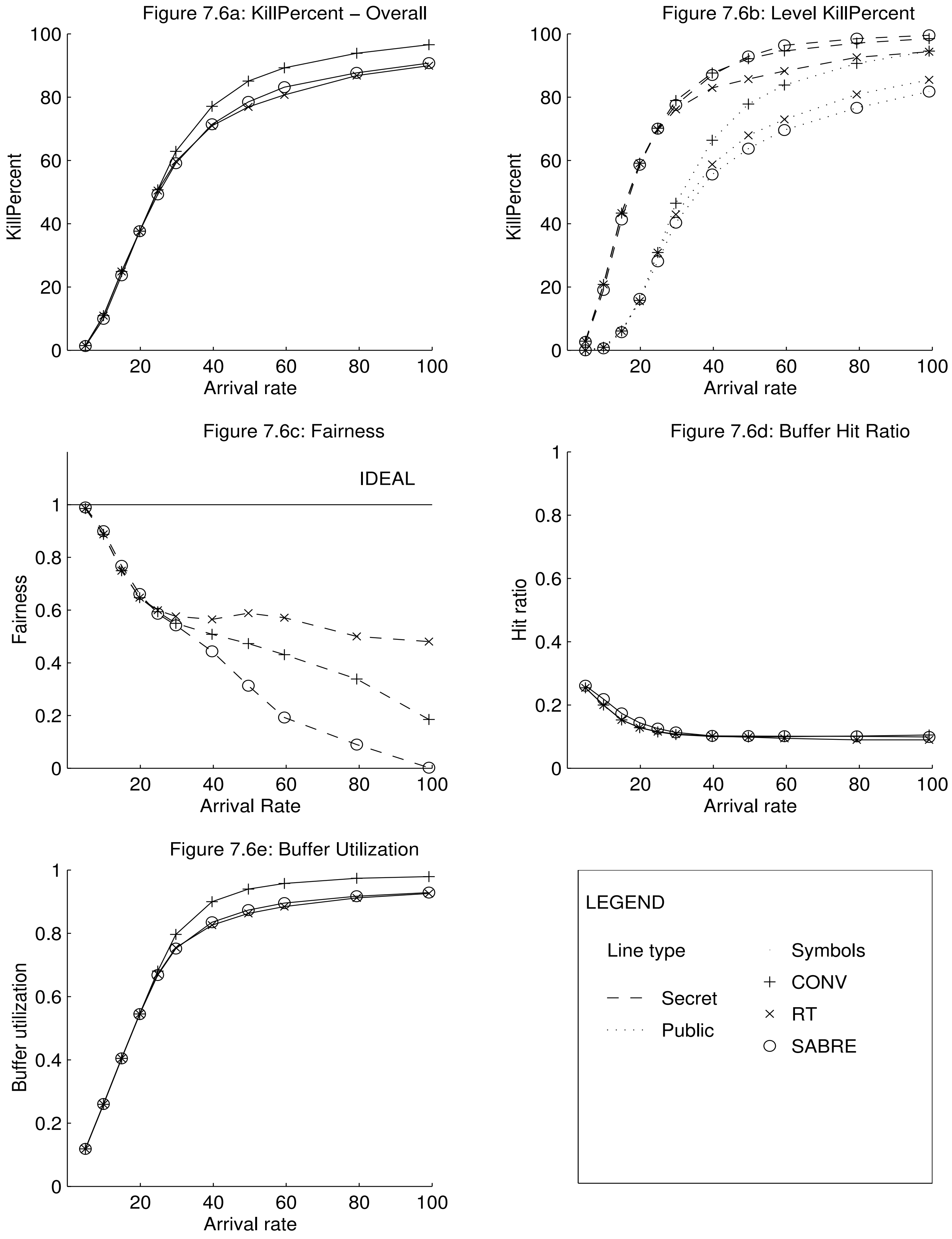


Figure 7.6: Inter-Transaction Locality

Table 7.3: Intra-Transaction Locality

InterLoc	0.0
IntraLoc	0.8

overall kill percentages (Figure 7.7a) are considerably *reduced* for all the policies, compared to their values in the No Locality experiment, because the intra-locality increases the hitratio *without* attracting the penalty of data conflicts.

Another interesting point to note here is that CONV performs much worse than RT and SABRE unlike in the previous experiments where there was only a marginal difference in real-time performance. This is due to the fact that the deadline cognizance of SABRE and RT is more effective under intra transaction locality. In this case, a transaction repeatedly refers some of the pages that it accessed earlier. The priority-cognizant buffer management policies keep the active/pinned pages of the urgent transactions in the buffer and therefore they will be available when re-referred. This helps the urgent transactions to complete before their deadlines, thereby resulting in performance improvement. At high loads, however, SABRE performance becomes worse than RT, and that is because the security-cognizance of SABRE results in Public transactions hogging most of the buffer slots and thereby causing the Secret hot spots to be absent from the buffer pool.

Similar to earlier experiments, in SABRE, the Secret transactions lose heavily to the Public transactions at high loads as is evident from the level-wise kill percentages shown in Figure 7.7b and from the fairness characteristics shown in Figure 7.7c. Again, this is due to SABRE's enforcement of covert channel security constraints.

Figure 7.7d shows the buffer hit ratio characteristics. The hitratios are high for all the protocols due to the introduction of intra transaction locality, and SABRE has the best hitratio amongst them. Finally, Figure 7.7e shows the buffer utilization. In SABRE, the Secret cannot replace the active Public slots, and therefore the overall buffer utilization is less. The CONV utilization is the highest but this does not result in better KillPercent performance due to the priority indifferent utilization of buffer slots.

In summary, with intra-transaction locality, the real-time features of RT and SABRE

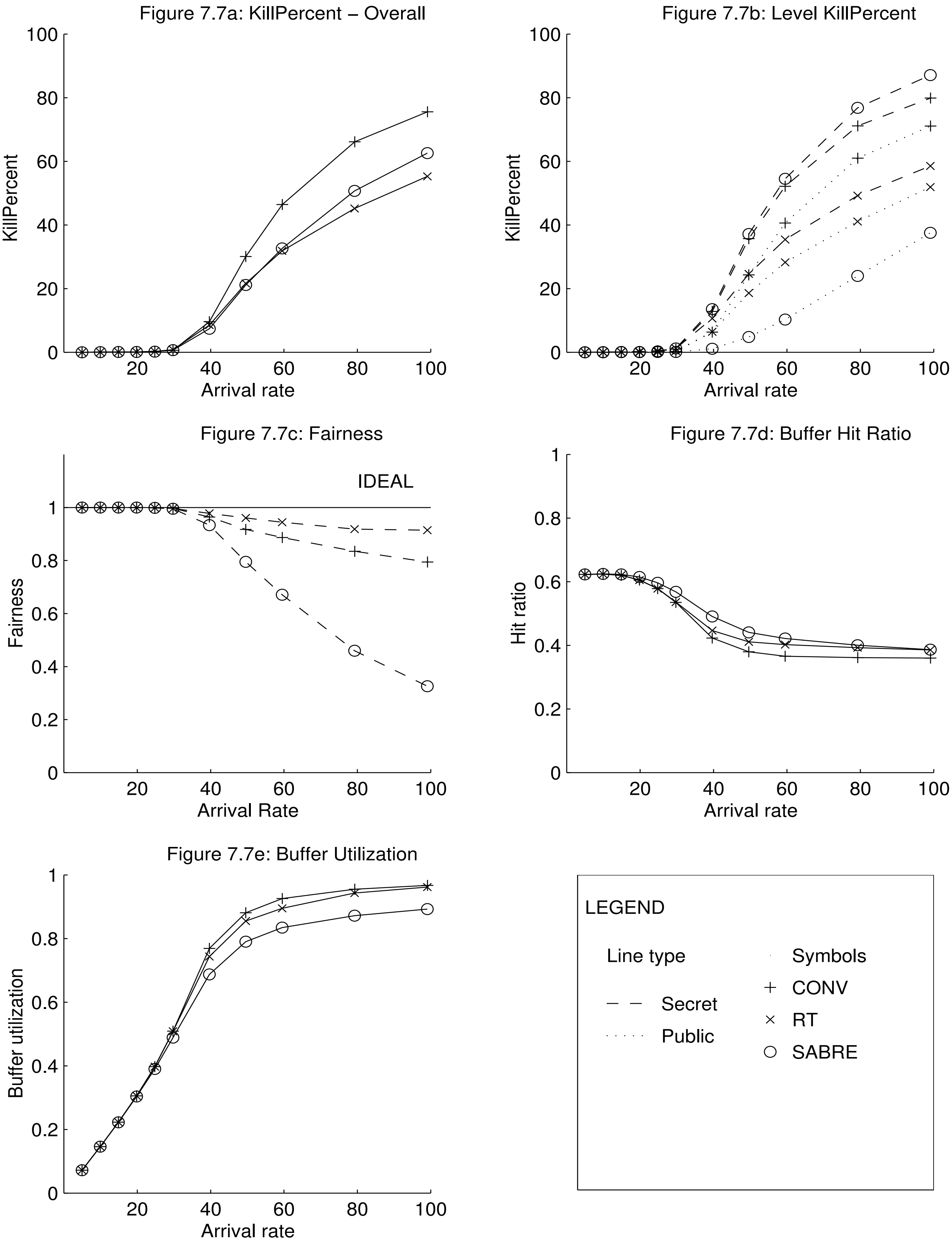


Figure 7.7: Intra-transaction locality

Table 7.4: Increased Security Levels

ClassLevels	5
ClearLevels	5
InterLoc	0.14
IntraLoc	0.8

deliver significant performance benefits. SABRE, the secure real-time buffer management policy, continues to be unfair as in previous experiments.

7.9 Experiment 5: Increased Security Levels

In this experiment, we kept all the system and workload parameters the same as those in Experiment 1 except that there were *five* security levels (*Top Secret, Secret, Confidential, Classified, Public*) for this experiment (see Table 7.4).

The results of this experiment are shown in Figures 7.8a–e. There are some notable differences with regard to the overall KillPercent performance in this five-level environment (Figure 7.8a) as compared to the equivalent 2-level experiment described (Experiment 1) : First, at low loads, SABRE performs slightly worse than both CONV and RT. Second, the overall killpercent for SABRE is noticeably more than that of RT at high loads (although still less than CONV). The reason for this behavior is that in a five-level system, priority is much more *level-based* than *deadline-based.* Therefore, the real-time features of SABRE have much less effect resulting in degraded real-time performance.

With respect to the two artificial baseline policies, ALLHIT and ALLMISS, we note again that there is a vast gap in their performance. This highlights the fact that there is a good potential for an intelligent buffer management policy to improve the real-time performance.

In Figure 7.8b, we plot the kill percentage on a per-security-level basis for SABRE. These graphs clearly show the extent to which the kill percentages are skewed among the various transaction security levels, with Top Secret transactions having the most number of missed deadlines and Public transactions having the least.

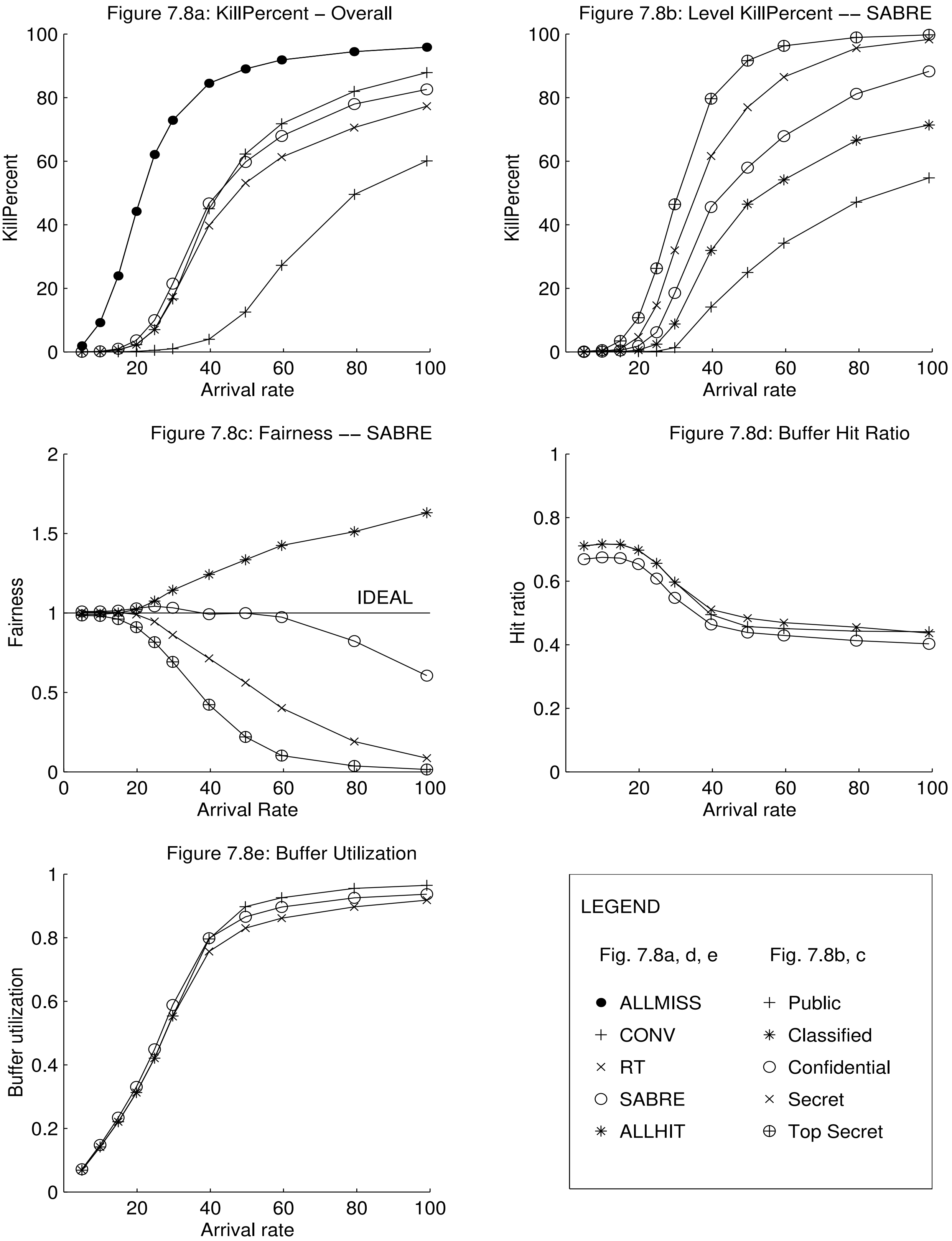


Figure 7.8: Increased security levels

In Figure 7.8c, the fairness factors for the top four security levels (Top Secret, Secret, Confidential and Classified) are plotted on a per-security-level basis. These figures clearly show that as the loading factor increases, progressively more and more security classes become discriminated against by the lowest security class (Public).

In Figure 7.8d, we show the buffer hit ratio for the various protocols on an overall basis. We see here that the SABRE buffer hitratio is only slightly less than that of CONV and RT.

Finally, Figure 7.8e shows the buffer utilization, that is the average number of pinned buffer slots, for the protocols and it is clear here that CONV utilizes the buffers slightly more than RT and SABRE. However, this does not result in better real-time performance because RT and SABRE selectively give the pinned slots to tight-deadline transactions whereas CONV is deadline-indifferent.

In summary, in a five level system SABRE performs slightly worse than both CONV and RT at low loads. At high loads, however, SABRE performs slightly better than CONV. Another observation is that the security requirements make SABRE unfair to the higher security classes, especially at high loads.

7.10 Conclusions

In this chapter, we have quantitatively investigated the performance implications of the choice of buffer management protocol in full-secure firm-deadline RTDBS. We first highlighted the several challenges associated with providing secure buffer managers including the multiplicity of covert-channel mediums, the multiplicity of buffer components, etc. We then presented SABRE, a new real-time buffer manager protocol that provides complete covert-channel-free security. SABRE is a dynamic one-copy protocol that ensures efficient usage of buffer resources and incorporates several optimizations for reducing the overall number of missed deadlines and for decreasing the unfairness in the distribution of missed deadlines across security levels.

Second, the real-time performance of SABRE was evaluated against the CONV and RT unsecure conventional and real-time buffer management algorithms for a variety of security-classified transaction workloads and system configurations. Our experiments showed that (a) it is essential to include deadline-cognizance in the buffer manager for good real-time performance. and (b) as per our experimental results, especially for applications with few security levels, SABRE efficiently provides both security and good real-time performance.

To the best of our knowledge, the above results represent the first work in the area of secure real-time buffer management.

Chapter 8

Fairness

8.1 Introduction

Our CC protocols and BM policies described in the earlier chapters improve the real-time performance and simultaneously ensure full security. However, these protocols are unfair to the high security classes due to the non-interference requirement that they have to satisfy. In this chapter our objective is to develop mechanisms for improving fairness across security levels of a secure real-time transaction processing system.

The policies for ensuring fairness can be categorized into static and dynamic policies. Static mechanisms for ensuring fairness such as resource reservation can provide fairness while still maintaining full security. However, they may considerably degrade the *real-time* performance arising out of resource wastage due to their inability to readjust themselves to varying workload conditions.

Dynamic mechanisms, on the other hand, have the ability to adapt to workload variations. However, providing full-security fairness with dynamic mechanisms appears to be *fundamentally impossible* since the fairness-inducing mechanism can itself become the medium of information compromise. For example, consider the following situation: A conspiring Secret process submits a workload such that the Secret class performance degrades. The fairness mechanism subsequently tries to improve the Secret class performance by reserving more resources for the Secret class. A collaborating Public transaction could now

```
Initial condition: Admit[Public] = 1.0, KillPercent[Public] = 0.0,
                                        KillPercent[Secret] = 0.0
LOOP:  if KillPercent[Secret] > 0.05 then
           FairFactor = KillPercent[Public] / KillPercent[Secret]
           if (FairFactor < 0.95) then
                Admit[Public] = Admit[Public] * 0.95
           else if (FairFactor > 1.05) then
                Admit[Public] = Admit[Public] * 1.05
       else
           Admit[Public] = 1.0
       sleep T seconds
       measure KillPercent[Public] and KillPercent[Secret] over
               the last T second interval
       goto statement LOOP
```

Figure 8.1: The GUARD Admission Control Policy

feel the reduction in the availability of system resources and sense the presence of the Secret process in the system. Therefore, this mechanism could be exploited for covert signaling.

In light of the above discussion, we consider in this chapter the more tractable problem of whether it is possible to dynamically provide fairness while guaranteeing *Orange Security* (i.e. covert-channel bandwidth of less than one bit per second).

8.2 The GUARD Admission Control Policy

Admission control has been used earlier for improving the performance in firm deadline RTDBS [73, 39]. However, we are not aware of any prior research work that suggests its use as a fairness mechanism in the secure transaction processing domain. In this section, we present **GUARD**, an adaptive admission-control protocol that attempts to provide such fairness while guaranteeing orange security.

For ease of exposition, we will assume that there are only two security levels, *Secret* and *Public*. For this environment, the GUARD admission control policy is shown in Figure 8.1. The basic idea in the policy is that based on the imbalance in the deadline miss percentages of the Secret and Public classes, the admission of Public transactions into the system is *periodically* controlled every T seconds (the setting of T is discussed below). The $FairFactor$ variable, which is the ratio of the the deadline kill percentage for the Public and Secret classes, captures the degree of unfairness during the last observation period. Ideally the FairFactor should be 1.0 and so, if there is a significant imbalance ($FairFactor < 0.95$ or $FairFactor > 1.05$), what is done is to decrease or increase the admit probability of the Public transactions accordingly.[1] The increase or decrease margin has been set to a nominal 5 percent. The hope is that this mechanism will eventually result in the multiprogramming level of the Public transactions reaching a value that ensures that the Secret transactions are not unduly harmed. Finally, to ensure that the admission control becomes operative only when the Secret transaction class is experiencing sustained missed deadlines, a threshold kill percentage of 5 percent is included for the admission control.

One danger associated with admission control policies is that, although providing fairness, they may cause unnecessary increases in the kill percentages of *both* the Public and Secret transactions as compared to the corresponding values *without* admission control. Such "virtual fairness" had also been observed with some of the CC protocols evaluated in our study (see Chapter 5). However, as will be borne out in the experimental results described in this chapter, the GUARD policy does not suffer from this shortcoming – it only evenly redistributes the "pain" but does not really increase its overall magnitude.

8.2.1 Bandwidth Measurement

In the GUARD policy, covert channels can arise due to the admission control mechanism. For example, a corrupt Secret user can, by modulating the outcomes (increase, decrease, or constant) of the Admit[Public] computation, signal information to collaborating Public users. Corresponding to each computation, $log_2 3 = 1.6$ bits of information can be transmitted (based on information-theoretic considerations [85]) and since the computation is made once every T seconds, the total channel bandwidth is $1.6/T$ bits per second.

[1]No restrictions are placed on the admission of Secret transactions.

By setting this equal to 1 bit per second, the condition for being orange-secure, we get $T_{min} = 1.6$ seconds to be the minimum re-computation period.

8.2.2 Design Rationale

The GUARD policy, as described above, implements a system-wide admission control policy by controlling entry of Public transactions into the database system. At first sight, it may be thought that it might have been simpler to apply similar controls at the individual resources instead. There are several problems with this approach, however, the main one being that transactions *already hold locks* on data items when they request a resource – delaying or not admitting requests of Public transactions will result in "cascading blocking" since the Secret transactions will queue up behind the data locks held by the Public transactions (the Secret cannot preempt the Public due to security reasons). This would only result in performance becoming degraded for *both classes* of transactions without benefits to either. Second, it is not clear how to measure unfairness at the resource. Third, even if it were possible to measure unfairness, should the control be applied on a per-access basis - in this case, a transaction that had not been hindered before may now find itself being forced to wait for the resource, thereby negating priority effects. Fourth, the control would work only in the case where the resource is the primary bottleneck, whereas GUARD's approach is more general. Fifth, it is easy to switch to full security if applications demand it by switching off the admission control – this may be more difficult to do at the individual resource level. Finally, GUARD's approach is based on output metrics, not internal system parameters.

8.2.3 Increased Number of Security Levels

The GUARD protocol described above assumed a two security-level system. The protocol can be extended to the generic N security level scenario, as given below in Figure 8.2. Note that the channel capacity in this general case is $log_2 3^{N-1} = 1.6(N-1)$ bits and therefore $T_{min} = (N-1) * 1.6$ seconds.

```
Initial condition: Admit[1:n] = 1.0, KillPercent[1:n] = 0.0
LOOP:  for i = 1 to n-1
         do
         if OverallKillPercent > 0.05 then
              do
              FairFactor =  KillPercent[i] / OverallKillPercent
              if (FairFactor < 0.95) then
                  Admit[i] = Admit[i] * 0.95
              else if (FairFactor > 1.05) then
                  Admit[i] = Admit[i] * 1.05
              done
          else
              Admit[i] = 1.0
         done
       sleep T seconds
       measure KillPercent[1:n] over the last T second interval
       goto statement LOOP
```

Figure 8.2: General GUARD Admission Control Policy

8.3 Admission Control Model

Our admission control model comprises of a sensory mechanism and a regulatory mechanism. The sensory mechanism calculates the average kill percentage of each security level over a period of time and makes it available to the regulatory mechanism. Depending on the data collected by the sensory mechanism over a period of time, the regulatory mechanism uses a *geometrically weighted* combination of the sensed values of the sensed values over a time span and determines the multi-programming level for each of the security levels. The sensory mechanism operates once every S seconds and the regulatory mechanism is scheduled once every T seconds. The T is set in accordance with the degree of security required. For an orange secure two level system, the T parameter is set to 1.6s. The S parameter is set to 0.25s in our experiments.

8.4 Comparative Systems

In the following experiments, we compare two systems: (1) **FULL**: A full secure SRTDBS equipped with secure 2PL-HP CC manager and SABRE buffer manager; (2) **GUARD**: An orange secure SRTDBS that is equipped with secure 2PL-HP CC manager, SABRE buffer manager, and the GUARD admission controller. Note that the FULL system was already evaluated in Chapter 7 – we reproduce its results here for the purpose of comparison.

8.5 Experiment 1: Two Security Levels

In our first experiment, we evaluated the performance of FULL and GUARD for a two security-level system. In this experiment, the GUARD parameters, S and T were set to $100ms$ and $1.6s$, respectively. The results of this experiment are shown in Figures 8.3a–c, which capture the overall kill percent, the level-wise kill percent, and the class fairness, respectively. In these figures we see that GUARD achieves *close to ideal fairness*. Further, at low loads, GUARD causes only *a small increase of the overall kill percentage* with respect to FULL, whereas at heavy loads, GUARD *actually does slightly better*. At low loads, due to the inherent lag involved in the feedback process, GUARD sometimes tends to be over-conservative, preventing Public transactions from entering even when strictly not necessary, and thereby increasing the Public kill percentage. In contrast, under heavy loads, being conservative does less harm than being too liberal , and therefore its performance shows minor improvement over that of FULL.

In summary, the results of this experiment show that GUARD achieves its fairness *very efficiently* and that the admission control component functions as intended in its design.

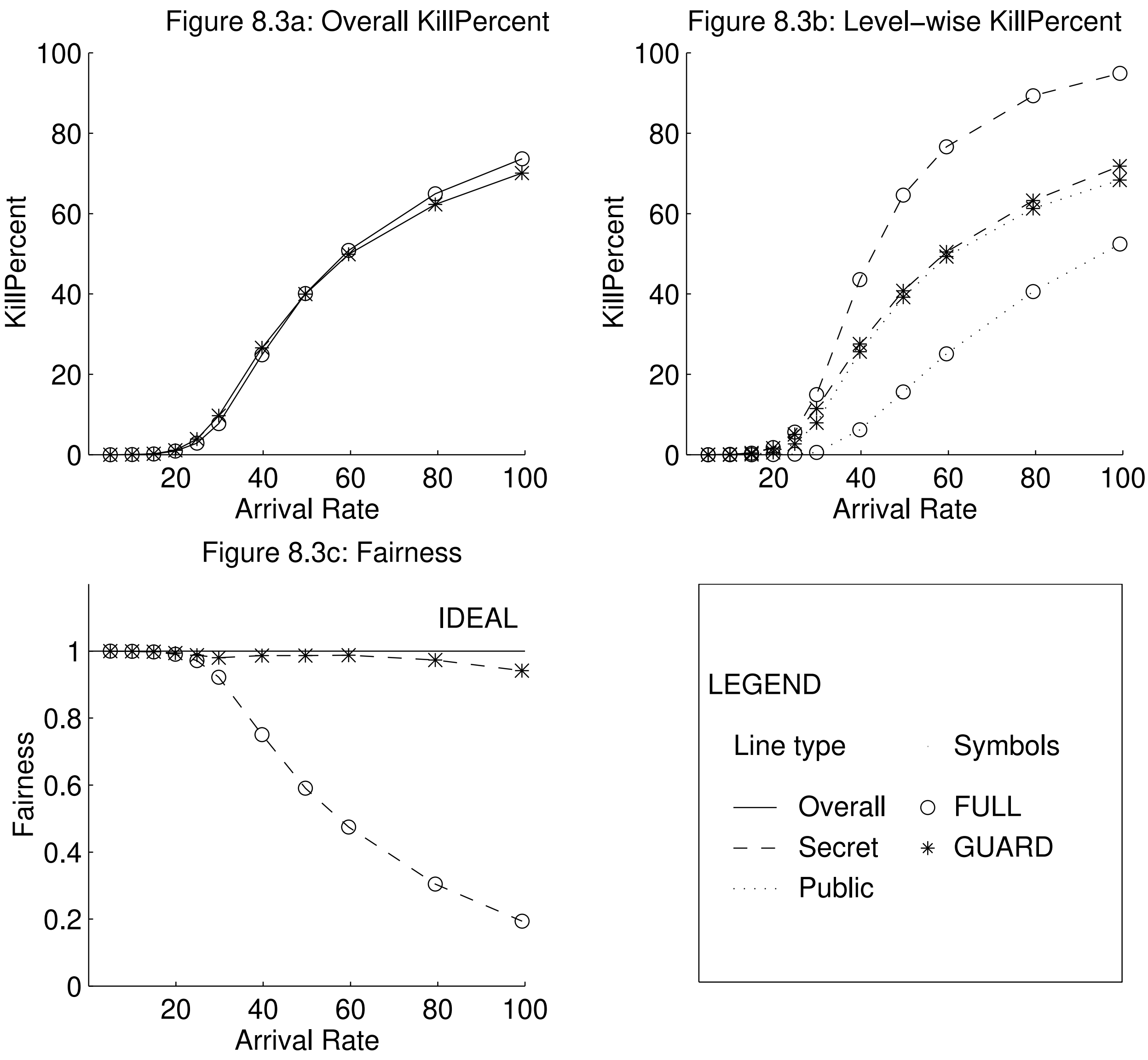


Figure 8.3: Fairness Experiment

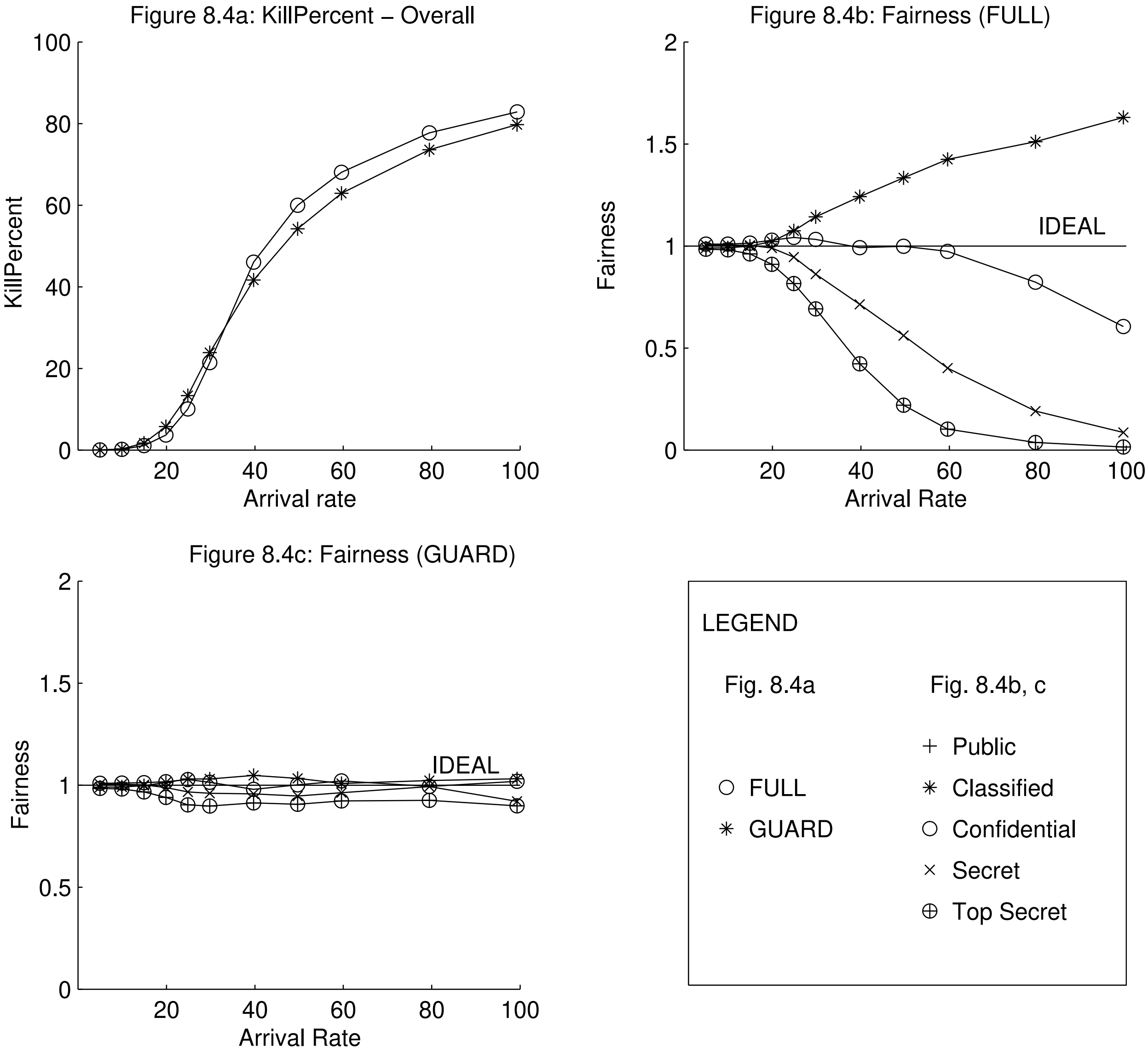


Figure 8.4: Increased security levels

8.6 Experiment 2: Increased Levels

In this experiment, we compare the FULL and GUARD systems for a five security level environment. The results of this experiment are shown in Figures 8.4a-c.

In Figure 8.4a which shows the overall KillPercent characteristics, we see that as in the two level experiments, GUARD performs marginally worse than FULL at low loads, due to the occasional over-conservative behavior of GUARD protocol. In contrast, under heavy loads, being conservative does less harm than being too liberal (since the system is essentially in a "thrashing" region), and therefore its performance shows minor improvement over that of FULL.

In Figures 8.4b and 8.4c, the fairness characteristics of the FULL and GUARD system are shown. As is evident from Figure 8.4c, GUARD exhibits close to the ideal fairness characteristics throughout the loading range even in the five level system.

8.7 Other Experiments

As mentioned earlier, we chose to use the 2PL-HP CC protocol in these experiments due to the difficulties involved in integrating optimistic CC with buffer management. However, we also conducted experiments wherein the S2PL-WAIT and OPT-WAIT CC protocols were modeled in combination with GUARD in the absence of a buffer pool. In such a scenario, the GUARD protocol achieved near ideal fairness while simultaneously fulfilling the orange security requirements. We also conducted several experiments for different workload conditions and security levels. Our experiments consistently showed that under all circumstances that we evaluated, GUARD achieved close to ideal fairness.

8.8 Conclusions

To summarize, static mechanisms for ensuring fairness are generally unsuitable for real-time transaction processing environments. Our experience shows that a system-wide dynamic fairness enforcement policy is a plausible substitute, and we present the **GUARD** feedback-based admission control policy which ensures fairness without violating the information leakage bandwidth restrictions specified in military standards. Performance evaluation of GUARD shows that it provides close to ideal fairness without incurring a significant degradation of the overall real-time performance.

Chapter 9

Conclusions and Future Research

9.1 Summary of Contributions

In this thesis, we have quantitatively investigated the design of high performance transaction processing mechanisms for maintaining covert-channel-free security in real-time database systems. Such systems find application in safety-critical installations and military units where enforcing security is crucial to the success of the enterprise. Despite the application importance, the issue of providing security in real-time database systems has been largely ignored by the research community. We address this lacuna in our work by designing secure concurrency control and buffer management protocols for firm deadline applications, for whom completing a transaction after its deadline has expired is of no utility and may even be harmful.

Apart from ensuring the data integrity constraints, a secure real-time database system has to simultaneously satisfy two requirements, namely, provide security and minimize the number of missed deadlines. Unfortunately, the mechanisms for achieving the individual goals often work at cross-purposes. Therefore, the design of secure transaction processing protocols involves considerable design complexity. Unlike previous studies, which used a *tradeoff* approach between security and timeliness, we have considered security for two kinds of applications: (1) Full-secure applications for whom security is as an "all-or-nothing" issue; (2) Orange-secure applications for whom it is acceptable to have covert-channels under the guarantee that the bandwidth of these channels is less than 1 bit per second. In comparison to these *correctness* criterion, the number of missed deadlines is a *performance* issue. Therefore, our study investigates the problem of how to minimize, as far as possible, the number of missed transaction deadlines *without* compromising security. To the best of our knowledge, this is the first detailed study of database security in the real-time database context.

9.1.1 Full Security

For full secure applications, we first identified that in order to satisfy the requirement of non-interference, only those real-time CC protocols that are free from priority inversion can be used in an SRTDBS. This requirement ruled out several previously proposed real-time CC protocols, including algorithms such as 2PL Wait Promote and WAIT-50. Then, using a detailed simulation model of a firm-deadline RTDBS, we studied the performance of the secure versions of the 2PL-HP, OPT-SACRIFICE and OPT-WAIT real-time CC algorithms; a non-real-time secure CC algorithm, Secure 2PL, was also included in the suite of algorithms. We also compared the performance of these algorithms with respect to a system where only direct unauthorized access, but not covert channels, is prevented.

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

In order to improve the performance further, we proposed a novel dual approach to secure CC wherein different CC algorithms are used to resolve inter-level conflicts and intra-level conflicts. A global serialization order was ensured, in spite of having multiple CC algorithms operating simultaneously, by using the system architecture proposed in [99]. The dual combinations of HP-WAIT and WAIT-HP were implemented and evaluated – they both generally performed worse than pure OPT-WAIT. However, the dual combination of S2PL-WAIT performed better than OPT-WAIT, especially at lower miss percent levels. This is because S2PL is a non-restart-oriented algorithm unlike both OPT-WAIT and 2PL-HP, and therefore ensures reduction of the harm done to high-security transactions.

Having addressed the issues pertaining to secure CC, we shifted our attention to full-secure real-time buffer management. We first highlighted the several challenges associated with providing secure buffer managers including multiplicity of covert-channel mediums, the multiplicity of buffer components, etc. We then presented SABRE (Secure Algorithm for Buffering in Real-time Environments), a new real-time buffer manager protocol that provides complete covert-channel-free security. SABRE is a dynamic one-copy protocol that ensures efficient usage of buffer resources and incorporates several optimizations for reducing the overall number of missed deadlines and for decreasing the unfairness in the distribution of missed deadlines across security levels.

Using a detailed simulation model, the real-time performance of SABRE was evaluated against the CONV and RT unsecure conventional and real-time buffer management algorithms for a variety of security-classified transaction workloads and system configurations. Our experiments showed that (a) it is essential to include deadline-cognizance in the buffer manager for good real-time performance, and (b) SABRE efficiently provides security in that its real-time performance is not significantly worse than that of RT. To the best of our knowledge, these results represent the very first work in the area of secure real-time buffer management.

9.1.2 Orange Security

The new protocols that we described above are meant for the full secure environment. A basic problem with these schemes is that, although they provide good overall real-time performance, they have poor *fairness* – transactions belonging to higher security classes usually form a disproportionately large fraction of the transactions that miss their deadlines. Dynamically providing fairness without incurring covert channels, however, appears to be *fundamentally impossible* since the fairness-inducing mechanism itself can become the medium of information compromise. Therefore, for applications that require complete security, unfairness can only be mitigated to an extent, but not eliminated.

For orange secure applications, however, we were able to satisfactorily address the problem. In particular, we presented the **GUARD** protocol, which is a feedback-based admission control policy. By appropriately setting the feedback period it was ensured that GUARD provided orange security. The GUARD protocol measures the imbalance in the miss ratios across various security levels and accordingly controls the admission probabilities of transactions at each security level. A performance evaluation of GUARD showed that it ensures close to ideal fairness without really affecting the overall real-time performance. That is, the GUARD policy evenly redistributes the "pain" but does not really increase its magnitude.

9.2 Future Research

The research that we have presented in this thesis can be extended in a variety of ways:

- For certain real-time applications, it is possible that different transactions may have *different values* and the goal of the RTDBS is to maximize the total value of the in-time transactions. This problem has been extensively covered in the conventional real-time transaction processing context [39]. Our dual protocols for CC could be easily extended to such environments by using the Hierarchical Earliest Deadline priority assignment [39] policy for the INTRA component of the priority vector.

- Since our focus was primarily on centralized SRTDBS, we did not address the issue of designing secure real-time recovery solutions. This is imperative, however, for distributed SRTDBS, since it is difficult to satisfy the non-interference requirement in distributed transaction commit protocols.

- Our GUARD admission-control protocol employs a simple feedback-based mechanism to ensure fairness. Its performance could perhaps be further improved by enhancing its design with regard to improving its convergence characteristics.

- Our study was focused on transaction processing systems where transactions have "firm" deadlines. Some of our protocols may be extended to "soft" deadline systems, where transactions are allowed to continue even after the expiry of their deadlines.

- A basic assumption in our study is that the SRTDBS has no a-priori knowledge of transaction processing requirements or of transaction data accesses. It would be worthwhile to investigate how, and to what extent, performance could be improved if the SRTDBS were to be apriori provided with more information about transaction characteristics (e.g., transaction size estimates).

- Another assumption that we have made is that timing constraints are associated only with transactions. There are several applications, however, wherein the *data* itself may be subjected to time constraints [39]. For example, in applications involving object tracking, the data describing object locations is time-constrained in that it is valid for a limited time only. Transactions operating with such data have to ensure that they use temporally consistent values. This may call for the use of multiple versions of each data item to provide a history of data values. We therefore need to look into the design and evaluation of secure real-time multi-version CC algorithms.

9.3 Closing Note

In summary, we recommend the **S2PL-WAIT** concurrency control protocol, the **SABRE** buffer management policy, and the **GUARD** admission control mechanism to designers of SRTDBS as good choices for achieving high-performance secure real-time transaction processing. While we have not explicitly evaluated the combination of S2PL-WAIT and SABRE due to the open integration problems, we hope that the results presented here will motivate further research into this issue.

To the best of our knowledge, the work described in this thesis represents the first detailed study of security solutions in the real-time database context.

Appendix A

Master Concurrency Control

We include here the *Global Ordering Scheme* [99] used to ensure a single global serialization order in CC protocols based on the dual approach described in Chapter 6. In this scheme, the Local CC managers (LCCs) are required to keep track of the local serialization order of transactions and to communicate this to the master concurrency controller (MCC) whenever requested. The LCCs communicate the local serialization order using the notion of serialization numbers which is defined shortly. The MCC then ensures that a single global order is maintained.

Definition 1 *A* **serialization number sequence***, with respect to a schedule S, for a set of transactions is a sequence of numbers such that for any pair of transactions T_i and T_j, if m and n indicate the serialization numbers of T_i and T_j respectively, then the following holds: $m < n \Rightarrow \exists$ a serial schedule conflict equivalent to S, in which T_i finishes before T_j.*

The local serialization order of transactions at a LCC is communicated to the MCC by augmenting the interface presented by the LCCs to the MCC in the following fashion: Each LCC provides a function called Prepare_to_Commit. This function, when invoked with a transaction identifier, responds with an indication of whether or not the transaction can be committed. If it can be committed, the *serialization number* of the transaction with respect to the global schedule restricted to transactions of that class alone is also returned by the LCC of that class. The serialization numbers returned by a LCC are distinct. In other words, serialization numbers are never reused by a LCC, irrespective of the outcome of a transaction.

The MCC invokes the function Prepare_to_Commit when it gets the Commit_Trans call from the Transaction Manager. The returned serialization number is then checked for validity (explained shortly) and the transaction is committed or aborted based on the outcome of this test. We will need a few definitions before outlining the validity test.

Definition 2 *Let S be a schedule of a set of transactions T produced by a DBMS employing precisely one CC policy, say* M. *A* **serialization function, ser,** *for* M *is a function that maps every transaction in T to one of its operations such that the following holds: S is conflict equivalent to S', where S' is the serial schedule in which T_i occurs before T_j if and only if $ser(T_i)$ occurs before $ser(T_j)$ in S.*

All CC algorithms either inherently have serialization functions, or they can be introduced externally [29]. In particular, for locking protocols and optimistic algorithms, the function that maps every transaction to its commit operation is a serialization function, whereas for timestamp algorithms, the function is associated with the begin operations of transactions.

Definition 3 *The* **serialization time stamp** *(STS) of T_i is the time at which $ser(T_i)$ for the CC policy at MCC is received by the MCC.*

The MCC uses the STS to define the serialization order at the MCC while the LCCs communicate their local order through serialization numbers returned by the Prepare_to_Commit call. The MCC now ensures that the serialization orders at the MCC and the LCCs agree by maintaining the following data structures: The MCC maintains a *serialization time stamp* (STS) for each transaction. In addition to STS, the MCC also maintains for each transaction a *low water mark* (LWM) and a *high water mark* (HWM). The LWM and HWM of T_i determines the range in which its serialization number, as returned by the LCC, can lie. When a transaction is initiated, its LWM and HWM are initialized to 0 and ∞ respectively. They are updated as follows whenever a transaction T_i commits:

- For all T_j such that T_j belongs to the same class as T_i and T_j is after (in MCC's serial order) T_i (i. e., $STS(T_i) < STS(T_j)$) and T_j conflicts with T_i, set $LWM(T_j)$ = max ($LWM(T_j)$, serialization number of T_i).

- For all T_j such that T_j belongs to the same class as T_i and T_j is prior (in MCC's serial order) to T_i, (i. e., $STS(T_i) > STS(T_j)$) set $HWM(T_j)$ = min ($HWM(T_j)$, serialization number of T_i).

We now describe the test used for checking the validity of the serialization number returned in response to a Prepare_to_Commit call. Based on the result of the test, the transaction is committed or aborted.

Commit Test : The MCC allows a transaction to be committed only if its serialization number, as returned by the LCC, lies between its LWM and HWM.

We will now prove that the commit test ensures that a single global order is maintained.

Theorem 1 *If there exists an edge $T_i \rightarrow T_j$ in the precedence graph corresponding to a schedule produced by any combination of CC algorithms at MCC and the slave LCCs, then $STS(T_i) < STS(T_j)$.*

Proof: We will assume $STS(T_i) > STS(T_j)$ and derive a contradiction.

Case 1 : T_i and T_j belong to the same class.

Case 1.1 : T_i commits before T_j

Since $STS(T_i) > STS(T_j)$, T_j is prior (in MCC's serial order) T_i and hence $HWM(T_j)$ is set to some value which is smaller than or equal to the serialization number of T_i when T_i commits. Since $T_i \rightarrow T_j$, the serialization number of T_j returned by the LCC will be greater than that of T_i. So it exceeds $HWM(T_j)$ and hence the MCC will not allow it to commit.

Case 1.2 : T_j commits before T_i

We know that $LWM(T_i)$ will be set to some value which is greater than or equal to the serialization number of T_j when T_j commits because by our assumption T_i is after (in MCC's serial order) T_j and T_i conflicts with T_j. But $T_i \rightarrow T_j$ and hence the serialization number of T_i returned by the LCC is smaller than that of T_j. It is less than $LWM(T_i)$ and MCC will not allow it to commit.

Case 2 : T_i and T_j belong to different classes.

By the definition of STS, if $T_i \rightarrow T_j$, then $STS(T_i) < STS(T_j)$ because inter-class conflicts are regulated by the CC algorithm at the MCC, which will produce serializable schedules only.

$\square$

Appendix B

Secure Two Phase Locking

B.1 The S2PL Protocol

We describe here the Secure 2PL (S2PL) algorithm proposed in [89, 19]. We have reproduced their protocol exactly as it was originally proposed except that, for clarity, we have modified the description at some places. The following notation is used in the description: $queue[x]$ is a queue of operations, each of which could be holding a virtual lock $VpLock$, dependent virtual lock $DVpLock$, a real lock $pLock$ or waiting to set a lock $pWait$. An operation p could be a read r, or write w and an action is represented in the form $p_i[x]$. The $queue$ is not a strict FIFO queue, since elements can be inserted in the middle, but it can be considered a priority queue ordered by the $before$ relation between transactions. $SL(T_i)$ is the security access class of the transaction T_i.

When a request for lock p is submitted to the S2PL CC protocol, action is taken as per the following algorithm:

$Case1$ $(p = read) \wedge \exists(wLock(T_i, x) \vee VwLock(T_i, x) \vee DVwLock(T_i, x))$

$\quad$ $Read\ value\ of\ x\ written\ by\ w_i[x];$

$Case2$ $((p = write) \wedge \exists rLock(T_i, x))$

$\quad\quad$ $Upgrade\ rLock(T_i, x)\ to\ wLock(T_i, x);$

$Case3$

$\quad$ $\exists j((qLock(T_j, x) \vee VqLock(T_j, x) \vee DVqLock(T_j, x)$

$\vee\, Wait(T_j, x)) \wedge (j \neq i) \wedge ((p = write) \vee (q = write)))$

$pos := length(queue[x]);$

$while\ (pos > 0)$

$\quad T_j \longleftarrow transaction\ at\ queue[x].pos;$

$\quad if\ (operation\ (queue[x].pos)\ conflicts\ with\ p)$

$\quad\quad if\ (T_i \in before(T_j))$

$\quad\quad\quad \forall T_3((T_3 \in before(T_j)) \wedge (T_3 \in after(T_i))$

$\quad\quad\quad\quad\quad \wedge\, (SL(T_3) > SL(T_i)) \wedge (SL(T_3) > SL(T_j)))$

$\quad\quad\quad\quad abort\ T_3;\ /*\ value\ security\ violation\ */$

$\quad\quad\quad if\ (lock(queue[x].pos)\ is\ a\ real\ lock)\ /*\ deadlock\ */$

$\quad\quad\quad\quad call\ victim\ selection\ routine;$

$\quad\quad\quad\quad if\ (victim = T_i)$

$\quad\quad\quad\quad\quad exit;\ /*\ exit\ from\ routine\ */$

$\quad\quad\quad\quad else$

$\quad\quad\quad\quad\quad continue;\ /*\ to\ start\ of\ while\ loop\ */$

$\quad\quad\quad\quad endif$

$\quad\quad else\ /*\ if\ (T_i \notin before(T_j))\ */$

$\quad\quad\quad if\ (SL(T_i) = SL(T_j))$

$\quad\quad\quad\quad if\ (T_j\ has\ not\ committed)$

$\quad\quad\quad\quad\quad Insert(queue, pWait(T_i, x), pos);$

$\quad\quad\quad\quad else$

$\quad\quad\quad\quad\quad if\ (p = read)\ read\ w_j[x];$

$\quad\quad\quad\quad\quad Insert(queue[x], pLock(T_i, x), pos);$

$\quad\quad\quad\quad endif$

$\quad\quad\quad elseif\ ((p = write) \wedge (\exists z, VwLock(T_i, z)))$

$\quad\quad\quad\quad Insert(queue, DVwLock(T_i, x), pos);$

$\quad\quad\quad\quad\quad\quad\quad /*\ Insert\ after\ pos\ */$

$\quad\quad\quad\quad dependent(w_i[z]) := w_i[x];$

$\quad\quad\quad else\ if\ (SL(T_i) < SL(T_j))$

$Insert(queue, VwLock(T_i, x), pos);$

$else$

$Insert(queue, pWait(T_i, x), pos);$

$endif$

$before(T_i) := before(T_i) \cup T_j \cup before(T_j);$

$after(T_j) := after(T_j) \cup T_i \cup after(T_i);$

$exit;$ /* $exit\ from\ routine$ */

$endif$

$else$

$pos := pos - 1;$

$endif$

$endwhile$ /* $end\ of\ while\ loop$ */

$if\ (pos = 0)$

$Insert(queue[x], pLock(T_i, x), pos);$

B.2 Security Loopholes in S2PL

In the remainder of this chapter, we review the S2PL protocol and point out the circumstances under which it *fails* to ensure complete security. For ease of understanding, we use transaction execution schedules. A schedule is a log of the (start of) operations performed by the system for a transaction. On the left-hand side of the schedule, we have transactions represented in the form $T_i(CL)$; this represents a transaction with identifier i and security clearance level CL. Corresponding to every transaction appearing in the schedule, on the right hand side, there is a list of actions of the form $p_i[x]$ where "p" is an action the transaction T_i performs on the data item x. An action p can appear in the schedule in two forms, r or w, which stands for read and write, respectively. For example, the read performed on a data item x by transaction T_i is represented as $r_i[x]$. Similarly, a write performed on the data item x by transaction T_i is represented as $w_i[x]$. The actions in the schedule occur in left to right order. The left most action occurs first and the right most action occurs the last. If some actions appear in the same vertical column, it means they can occur simultaneously (i.e., there is no ordering defined among those operations).

B.2.1 Recovery Security Loophole

A possible way in which the value security in S2PL could be violated and the method to prevent such violation is explained in [19].

Transactions	Activity					
$T_1(TOPSECRET)$	$r_1[y]$	$r_1[x]$	....			c1
$T_2(SECRET)$					$r_2[z]$	c2
$T_3(CLASSIFIED)$	$w_3[x]$			$w_3[z]$	c3	
$T_4(UNCLASSIFIED)$		$w_4[y]$	$w_4[z]$		c4	

In the above schedule, the value security is violated as follows. The TOPSECRET transaction T_1 secures a read lock on data item y. Subsequently, the UNCLASSIFIED transaction T_4 submits a write request on "y". Since the T_1 already holds a lock on "y", T_1 is set *before* T_4 in the serialization order. Similarly T_3 is set *before* T_1 in the serialization order, because T_3 is already holding a write lock on data item "x" when T_1 requests for "x". When T_3 requests a write lock on data item "z", T_4 already has a dependent virtual lock on it. Since T_1 is *before* T_4 and T_3 is *before* T_1, the transaction T_3 is allowed to overtake T_4 in $queue[z]$. The value of "z" read by the SECRET transaction T_2 will be the value written by T_4. Had T_1 not been present in the system, T_2 would have read the value written by T_3. Since it is possible for a higher security level transaction to alter the value read by a lower security level transaction, if the above schedule is allowed to execute, value security may be violated.

The way in which S2PL deals with the above kind of value security violations is to abort **all** transactions *before* T_4 and *after* T_3 whose security level is higher than T_3 and T_4. So in this case T_1 is aborted and the value security is maintained. However, this scheme of aborts can itself lead to *recovery security* violations, as described below.

Transactions	Activity								
$T_0(ULTRA\ SECRET)$	$r_0[a]$	$r_0[c]$	..						c0
$T_1(TOP\ SECRET)$		$r_1[b]$		$w_1[a]$				$r_2[d]$	c2
$T_2(SECRET)$								$r_2[d]$	c2
$T_3(CLASSIFIED)$	$w_3[c]$						$w_3[d]$	c3	
$T_4(UNCLASSIFIED)$				$w_4[b]$	$w_4[d]$	c4			

Consider the above schedule. Here when T_4 requests for a write lock on "b", T_1 already holds a read lock on "b". Therefore, T_1 is set to *before* T_4. When T_1 requests a write lock on data item "a", T_0 is already holding a read lock on it, so T_0 is set to *before* T_1 in the serialization order. When T_0 requests for a lock on data item "c", T_3 is already holding a lock on it and therefore, T_3 is set *before* T_0 in the serialization order. Later T_4 performs a dependent virtual write on "d" and commits. Now when T_3 requests a write lock on "d", T_3 is allowed to supersede T_4 since the former comes earlier in the serialization order. This would not have happened if either T_0 were absent or T_1 were absent. According to the S2PL protocol, T_0 and T_1 are both aborted to ensure value security. But then this opens another covert channel as follows:

Imagine the case where T_0 wants to send some information to T_1, it could collaborate with T_1, T_2, T_3, and T_4 and create the above schedule. Faced with this scheudle, S2PL promptly detects value security violation and aborts *both* T_1 and T_0. Since T_1 is also aborted along with T_0, it is a security violation since had T_0 been absent, T_1 would not have aborted.

B.2.2 Delay Security Loophole

Their delay security proof states that transactions are not delayed because of high security transactions. Examine the following schedule:

Transactions	Activity				
$T_1(TOP\ SECRET)$	$r_1[a]$				$r_1[c]$
$T_2(SECRET)$		$w_2[a]$		$w_2[b]$	
$T_3(CLASSIFIED)$			$w_3[b]$		

Consider the above schedule, Here, T_1 gets a read lock on data item "a" and after a while T_2 requests for a write on "a". Since the TOP SECRET transaction T_1 already holds a read lock on data item "a", a virtual lock (non-blocking) is granted for T_2. Now the CLASSIFIED transaction T_3 obtains a write lock on data item "b". Later, when T_2 requests a lock on data item "b", since it already has secured a virtual lock (for data item "a"), it is granted a dependent virtual lock (again, non-blocking). Suppose T_1 was not present in the system, $w_2[b]$ of transaction T_2 would have blocked until T_3 releases its lock on data item "b". Because of the presence of T_1, T_2 performs $w_2[b]$ quicker than otherwise. This is a delay security violation since T_2 completes *earlier* due to the presence of a higher security transaction.

Transactions	Activity								
$T_1(SECRET)$	$r_1[a]$								$r_1[c]$
$T_2(UNCLASSIFIED)$		$w_2[a]$	$w_2[b]$	c2					
$T_3(UNCLASSIFIED)$					$r_3[b]$	..	$w_3[c]$	c3	

In the above schedule, after T_2 completes its operations, it enters virtual commit since it has a virtual lock on the data item "a". After the virtual commit of T_2, if another UNCLASSIFIED transaction T_3 requests the data item "b", it should be allowed to read from the virtual copy to ensure correctness and delay security. Suppose T_3 reads "b" and writes to "c" and commits. After T_3 commits, the write lock on "c" is released. Then if the SECRET level transaction T_1 reads "c" it will violate serializability.

B.3 Non-serial Schedules

Apart from the above security-related problems, S2PL may also cause *non-serial* schedules, as explained below.

Transactions	Activity							
$T_1(TOP\ SECRET)$					$r_1[c]$			
$T_2(SECRET)$			$r_2[b]$			$w_2[c]$		$w_2[d]$
$T_3(CLASSIFIED)$	$r_3[a]$			$w_3[b]$				
$T_4(UNCLASSIFIED)$		$w_4[a]$					$w_4[d]$	

Consider the above schedule. Here, when T_4 requests a write lock on the data item “a”, T_3 has already obtained a read lock on “a”, so T_3 is set *before* T_4 in the serialization order and T_3 is added to the *beforelist* of T_4. Also, T_4 is added to the *afterlist* of T_3. For similar reasons, T_3 is added to the *afterlist* of T_2 and T_2 is added to the *beforelist* of T_3. Then T_2 obtains a dependent virtual lock on data item “d” after which T_4 requests for a write lock on “d”. Now, when T_4 makes the request for “d”, T_2 is **not** in the *beforelist* of T_4 and therefore it is set after T_2. This is incorrect because, in fact T_2 is before T_4 in the serialization order (in fact, the *afterlist* of T_2 contains T_4). Therefore, there could be a serializability violation in the S2PL protocol.

References

[1] T. Anderson, Y. Breitbart, H. Korth and A. Wool, "Replication, Consistency, and Practicality: Are These Mutually Exclusive?" *Proc. of the ACM-SIGMOD Conf.*, 1998.

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

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

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

[5] M. Abrams, S. Jajodia and H. Podell, *Information Security*, IEEE Computer Society Press, 1995.

[6] P. Amman, S. Jajodia and I. Ray, "Ensuring atomicity of multilevel transactions", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, CA, 1996.

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

[8] V. Atluri, E. Bertino and S. Jajodia, "Degrees of Isolation, Concurrency Control Protocols and Commit Protocols", *Database Security, VIII: Status and Prospects*, 1994.

[9] V. Atluri, S. Jajodia and E. Bertino, "Transaction Processing in Multilevel Secure Databases using Kernelized Architecture: Challenges and Solutions", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, No. 5, 1997.

[10] V. Atluri, S. Jajodia, T. F. Keefe, C. MaCollum and R. Mukkamala, "Multilevel Secure Transaction Processing: Status and Prospects", *Database Security, X: Status and Prospects*, Chapman & Hall 1997, eds. Pierangela Samarati and Ravi Sandhu.

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

[12] E. Bertino, P. Samarati and S. Jajodia, "High Assurance Discretionary Access Control in Object Bases", *Proc. of First ACM Conf. on Computer Communications and Security*, Fairfax, VA, November 1993.

[13] K. J. Biba, "Integrity constraints for secure computer systems", *EST TR-76-372*, ESD/AFSC, Hanscom AFB, Bedford, MA, April 1977.

[14] B. T. Blaustein, S. Jajodia, C. D. McCollum and L. Notargiacomo, "A Model of atomicity for multilevel transactions", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, California, 1993.

[15] M. J. Carey, "Priority in DBMS Resource Scheduling", *Proc. of VLDB Conf.*, August 1989.

[16] S. Castano, M. Fugini, G. Martella and P. Samarati, *Database Security*, Addison-Wesley, 1995.

[17] O. Costich, "Transaction processing using an untrusted scheduler in a multilevel database with replicated architecture", *Data Security, V: Status and Prospects*, North-Holland, Amsterdam, 1992.

[18] H. Chou and D. DeWitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems", *Proc. of VLDB Conf.*, August 1985.

[19] R. David, "Secure Concurrency Control", *Master's Thesis*, Computer Science Department, University of Virginia, May 1993.

[20] R. David and S. Son, "A Secure Two Phase Locking Protocol", *Proc. of Symp. on Reliable Distributed Systems*, October 1993.

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

[22] S. Davidson, I. Lee and V. Wolfe, "A Protocol for Timed Atomic Commitment", *Proc. of 9th Intl. Conf. on Distributed Computing Systems*, 1989.

[23] D. E. Denning, "Secure information flow in Computer Systems", *Ph.D. Thesis*, University of Purdue, 1975.

[24] D. E. Denning, *et al.* "Views for multilevel database security", *IEEE Trans. on Software Engineering*, 13 (2).

[25] "DOD Trusted Computer System Evaluation Criteria", *Department of Defense Standard, DoD 5200.28-STD*, December 1985.

[26] L. C. Dion, "A complete protection model", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, CA, April 1981.

[27] W. Effelsberg and T. Haerder, "Principles of Database Buffer Management", *ACM Trans. on Database Systems*, Dec. 1984.

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

[29] D. Georgakopoulous, M. Rusinkiewicz and A. Sheth, "On serializability of multidatabase transactions through forced local conflicts", *Proc. of 7th IEEE Intl. Conf. on Data Engineering*, 1991.

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

[31] R. Graubart, "On the need of a third form of access control", *Proc. of 12th National Computer Society Conf.*, Gaithersburg, MD, October 1989.

[32] R. Gupta, J. Haritsa, K. Ramamritham, and S. Seshadri, "Commit Processing in Distributed Real-Time Database Systems", *Proc. of 17th IEEE Real-Time Systems Symp.*, Washington, DC, USA, Dec 1996.

[33] J. Gray, "Notes on Database Operating Systems", *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, 60, Springer-Verlag, 1978.

[34] J. Gray, P. R. McJones, and M. Blasgen, "The Recovery Manager of the System R Database Manager", *ACM Computing Surveys*, Vol. 13, No. 2.

Transaction Processing: Concepts and Techniques, Morgan-Kaufmann, 1993.

[35] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan-Kaufmann, 1993.

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

[37] J. Gray, "Notes on Database Operating Systems", *Operating Systems: An Advanced Course, Lecture notes in Computer Science*, 60, Springer-Verlag, 1978.

[38] T. Haerder, "Observations on Optimistic Concurrency Control Schemes", *Information Systems*, 9(2), 1984.

[39] J. Haritsa, "Transaction Processing in Firm Real-Time Database Systems", *Ph.D. Thesis*, Computer Sciences Department, University of Wisconsin, Madison, 1991.

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

[41] D. Hong, T. Johnson and S. Chakravarthy, "Real-Time Transaction Scheduling: A Cost Conscious Approach", *Proc. of ACM SIGMOD Conf.*, May 1993.

[42] J. Huang, "Real-Time Transaction Processing: Design, implementation, and performance evaluation", *Ph.D. Thesis*, University of Massachusetts, May 1991.

[43] J. Huang, J. Stankovic and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes", *Proc. of 17th VLDB Conf.*, 1991.

[44] S. Jajodia and V. Atluri, "Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, California, 1992.

[45] S. Jajodia and B. Kogan, "Integrating object-oriented data model with multilevel data security", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, California, 1990.

[46] S. Jajodia and R. Sandhu, "Toward a multilevel relational data model", *Proc. of ACM SIGMOD Conf.*, Denver, CO, May 1991.

[47] R. Jauhari, "Priority Scheduling in Database Systems", *Ph.D. Thesis*, Computer Sciences Department, University of Wisconsin, Madison, August 1990.

[48] E. Jensen, C. Locke and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proc. of 6th IEEE Real-Time Systems Symp.*, December 1985.

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

[50] I. E. Kang and T. F. Keefe, "On transaction processing for multilevel replicated databases", *Proc. of the European Symp. on Research in Computer Security*, 1992.

[51] H.T. Kuang and J. T. Robinson, "On Optimistic Methods for Concurrency Control", *ACM Trans. on Database Systems*, June 1981.

[52] T. F. Keefe and W. T. Tsai, "Multiversion concurrency control for multilevel secure database systems", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, California, 1990.

[53] T. Keefe, W. Tsai and J. Srivastava, "Multilevel Secure Database Concurrency Control", *Proc. of IEEE Intl. Conf. on Data Engineering*, February 1990.

[54] B. W. Kernighan and D. M. Ritchie, *C Programming Langauge* Prentice-Hall, 1988.

[55] W. Kim and J. Srivastava, "Buffer Management and Disk Scheduling for Real-Time Relational Database Systems", *Proc. of 3rd Intl. Conf. on Management of Data*, Bombay, India, December 1991.

[56] L. Kleinrock, *Queueing Systems*, Vol. II, John Wiley & Sons, 1976.

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

[58] B. Lampson and D. Lomet, "A New Persumed Commit Optimization for Two Phase Commit", *Proc. of VLDB Conf.*, Aug 1993.

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

[60] A. M. Law and C. S. Larney, *Introduction to Simulation Using SIMSCRIPT II.5*, CACI Products company, La Jolla, Calif., 1984.

[61] J. Lee and S. H. Son, "Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems", *Proc. of 11th IEEE Real-Time Systems Symp.*, December 1993.

[62] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, 20(1), 1973.

[63] R. A. Lorie, "Physical Integrity in a Large Segmented Database", *ACM Trans. on Database Systems*, Vol. 2, No. 1, Pages 91–104.

[64] B. Lampson and H. Sturgis, "Crash Recovery in Distributed Data Storage System", *Tech. Rep.*, Xerox, Palo Atlo Research Center, California, 1976.

[65] W. T. Maimone and I. B. Greenberg, "Single-level multiversion schedulers for multilevel secure database systems", *Proc. of 6th Annual computer security applications conf.*, Tuscon, Arizona, 1990.

[66] C. J. McCollumn, J. R. Messing and L. Notargiacomo, "Beyond the pale of MAC and DAC – defining new forms of access control", *Proc. of IEEE Symp. on Security and Privacy*, Oakland, CA, May 1990.

[67] J. McDermott, S. Jajodia and R. Sandhu "A single level scheduler for replicated architecture for multilevel secure databases", *Proc. of 7th Annual Computer Society Applications Conf.*, San Antonio, Texas, 1991.

[68] Y. Lin and S. Son, "Concurrency Control in Real-Time Database Systems by Dynamic Adjustment of Serialization Order", *Proc. of 11th IEEE Real-Time Systems Symp.*, December 1990.

[69] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, Jan. 1973.

[70] D. Menasce and T. Nakanishi, "Optimistic versus Pessimistic concurrency control mechanisms in Database Management Systems", *Information Systems*, 7(1), 1982.

[71] C. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R* Distributed Database Management System", *ACM Trans. on Database Systems*, 11(4), 1986.

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

[73] H. Pang, "Query Processing in Firm Real-Time Database Systems", *Ph.D. Thesis*, Computer Sciences Department, University of Wisconsin, Madison, 1994.

[74] H. Pang, M. Carey and M. Livny, "Transaction Scheduling in Multiclass Real-Time Database Systems", *Proc. of IEEE Real-Time Systems Symp.*, December 1992.

[75] H. Pang, M. Carey and M. Livny, " Managing Memory for Real-Time Queries", *Proc. of ACM SIGMOD Conf.*, May 1994.

[76] J. Peterson and A. Silberschatz, *Operating Systems Concepts*, Addison-Wesley, 1986.

[77] "Polyhedra, Technical Overview", `http://www.polyhedra.com`, 1998.

[78] K. Ramamritham and J. Stankovic, "Overview of the SPRING Project", *IEEE Real-Time Systems Newsletter* , Winter 1989.

[79] I. Ray, E. Bertino, S. Jajodia and L. Mancini, "An Advanced Commit Protocol for MLS Distributed Database Systems", *Proc. of 3rd Intl. Conf. on Computer and Communication Security*, ND, India, March 1996.

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

[81] D. Rosenkrantz, R. Stearns and P. Lewis, "System Level Concurrency Control for Distributed Database Systems", *ACM Trans. on Database Systems*, Vol. 3, No. 2, June 1978.

[82] D. Russel and G. T. Gangemi, *Computer Security Basics*, O'Reilly and Associates, Inc., 1991.

[83] E. C. Russel, *Building Simulation Models with SIMSCRIPT II.5*, CACI Products company, La Jolla, Calif., 1983.

[84] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *Tech. Rep. CMU-CS-87-181*, Departments of CS, ECE, and Statistics, Carnegie Mellon University, 1987.

[85] C. E. Shannon, “A Mathematical Theory of Communications”, *Bell Syst. Tech. J.*, October, 1948.

[86] D. Skeen, “Nonblocking Commit Protocols”, *Proc. of ACM SIGMOD Conf.*, May 1981.

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

[88] K. Smith, “Managing rules in active databases”, *Ph.D. Thesis*, Computer Science Department, University of Illinois at Urbana-Champaign, 1992.

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

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

[91] S. Son, R. David and B. Thuraisingham, “Improving Timeliness in Real-Time Secure Database Systems”, *ACM SIGMOD Record*, Special Issue on Real-Time Database Systems, March 1996.

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

[93] N. Soparkar, E. Levy, H. Korth and A. Silberchatz, “Adaptive Commitment for Real-Time Distributed Transactions”, *TR-92-15*, Computer Science Department, University of Texas-Austin, 1992.

[94] K. Smith and M. Winslett, “Entity modeling in the MLS relational model”, *Proc. of the VLDB Conf.*, Vancouver, British Columbia, Canada, 1992.

[95] Special Issue on Real-Time Database Systems, *Real-Time Systems Journal*, September 1992.

[96] J. Stankovic and W. Zhao, "On Real-Time Transactions", *ACM SIGMOD Record*, March 1988.

[97] M. Stonebraker, "Operating System support for Database Management", *Comm. of ACM*, Vol. 24, No. 7, 1981.

[98] A. Stoughton, "Access flow: a protection model which integrates access control and information flow". *Proc. of IEEE Symp. on Security and Privacy*, Oakland, CA, May 1981.

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

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

[101] O. Ulusoy, "Research Issues in Real-Time Database Systems", *Tech. Rep. BU-CEIS-94-32*, Department of Computer Engineering. and Information Science, Bilkent University, Turkey, 1994.

[102] K. G. Walter et al, "Primitive Models for Computer Security", *Tech. Rep. ESD-TR-4-117*, Case Western Reserve University, Cleveland, OH, January 1974.

[103] A. Warner and T. Keefe, "Version Pool Management in a Multilevel Secure Multiversion Transaction Manager", *Proc. of the IEEE Symp. on Security and Privacy*, 1995.

[104] A. Warner, Q. Li, T. Keefe and S. Pal, "The Impact of Multilevel Security on Database Buffer Management", *Proc. of the European Symp. on Research in Computer Security*, 1996.

[105] H. Wedekind and G. Zoerntlein, "Prefetching in Real-Time Database Applications", *Proc. of ACM SIGMOD Conf.*, June 1986.

[106] Y. Yoon, "Transaction Scheduling and Commit Processing for Real-Time Distributed Database Systems", *Ph.D. Thesis*, Korea Advanced Institute of Science and Technology, May 1994.