Real-time Database Experiences in Network Management Application

Yoshiaki Kiriha *

NEC Corporation
C&C Research Laboratories
4-1-1 Miyazaki, Miyamae-ku,
Kawasaki, Kanagawa 216, JAPAN

Stanford University
Computer Science Department
Stanford, CA 94305
E-mail: kiriha@db.stanford.edu

August 30, 1995

Abstract This report discusses on our experiences with real-time databases in the context of a network management system, in particular a MIB (Management Information Base) implementation. We propose an active and real-time MIB (ART-MIB) architecture that utilizes a real-time database system. The ART-MIB contains a variety of modules, such as transaction manager, task manager, and resource manager. Among the functionalities provided by ART-MIB, we focus on transaction scheduling within a memory based real-time database system. For the developed ART-MIB prototype, we have evaluated two typical real-time transaction scheduling algorithms: earliest deadline first (EDF) and highest value first (HVF). The main results of our performance comparison show that EDF outperforms HVF under a low load; however, HVF outperforms EDF in an overload situation. Furthermore, the fact that the performance crossover point closely depends on the magnitude of the scheduler queue, has been validated.

*This work has been done under direction of Prof. Hector Garcia-Molina, during the author's stay as a visiting scholar in Stanford University (1994-1995).

Contents

1 Introduction

In almost all current network management systems, a relational database or an object oriented database has been utilized for implementing a MIB (Management Information Base). By using such conventional database systems, network management systems can benefit from efficient query processing and reliable transaction processing. The major trends in telecommunication network and service evolution, such as broadband capabilities based on ATM and IN based service facilities, increase the management difficulty of both networks and services due to their complex structure and various timing constraints. Therefore, the advantages of conventional database systems are not sufficient for managing evolving telecommunication networks and services.

In particular, even though there exist various kinds of management operations (e.g., call statistics query, fault occurrence notification, transmission bandwidth update), every operation cannot have its own execution priority in a conventional database system. The priority might be derived from a deadline on the completion time and the importance to the application of meeting the deadline. As a result of this, it may be difficult to execute management operations by their expected completion time. Although ITU's TMN (Telecommunication Management Network[ITU-TMN]) provides an architectural framework for managing heterogeneous networks and services, an actual design and implementation that can deal with the above mentioned problem have not been discussed in detail so far. In order to deal with this problem and to sustain timely user response and quality of network services under any circumstances, it is important for the conventional MIB to add real-time capabilities that take into account the priority of each management operation.

Generally, real-time systems can be grouped into two categories: hard real-time system and soft real-time system. For a hard real-time system, missing a deadline is equivalent to a catastrophe. In order to estimate an accurate running time, all tasks are pre-analyzed and guaranteed to complete on time. Therefore, a hard real-time system is only applicable for periodic and relatively simple tasks. For a soft real-time system, however, missing deadlines leads to degraded performance but does not entail catastrophic results. Therefore, completing a task after a deadline has expired is allowed for a soft real-time system. Actually, since a soft real-time systems includes asynchronous and complex tasks, it is very difficult to guarantee that all deadlines will be met. Hence, a soft real-time system tries to minimize the number of deadlines that are missed. A network management system is a soft real-time system, since it includes not only periodic operations but also many aperiodic operations whose response time is difficult to estimate in advance. Unfortunately, previous work on soft real-time systems and databases has been mainly focused on theoretical or simulation studies. Therefore, we believe that an evaluation of an actual implementation of soft real-time capabilities for network management systems could be very important.

For the new MIB with soft real-time capabilities, the concepts of timeliness and activeness should be introduced. In this context, timeliness is an abstraction of different requirements such as importance and response time. Activeness is an abstraction of event driven functionalities of the system. What can support these concepts is a real-time database system[Kao93]. A real-time database system (RTDBS) is a database system that has real-time capabilities. The main objective of an RTDBS is to complete transactions before their deadlines expire. Furthermore, realtime databases are closely related to active databases that provide an event driven data processing function[Ramamritham93]. Therefore, an RTDBS is believed to be useful for applications with complex structure and timing constraints. In order to intuitively see that an RTDBS may provide network management systems with significant services, we briefly show a real-time scheduling example.

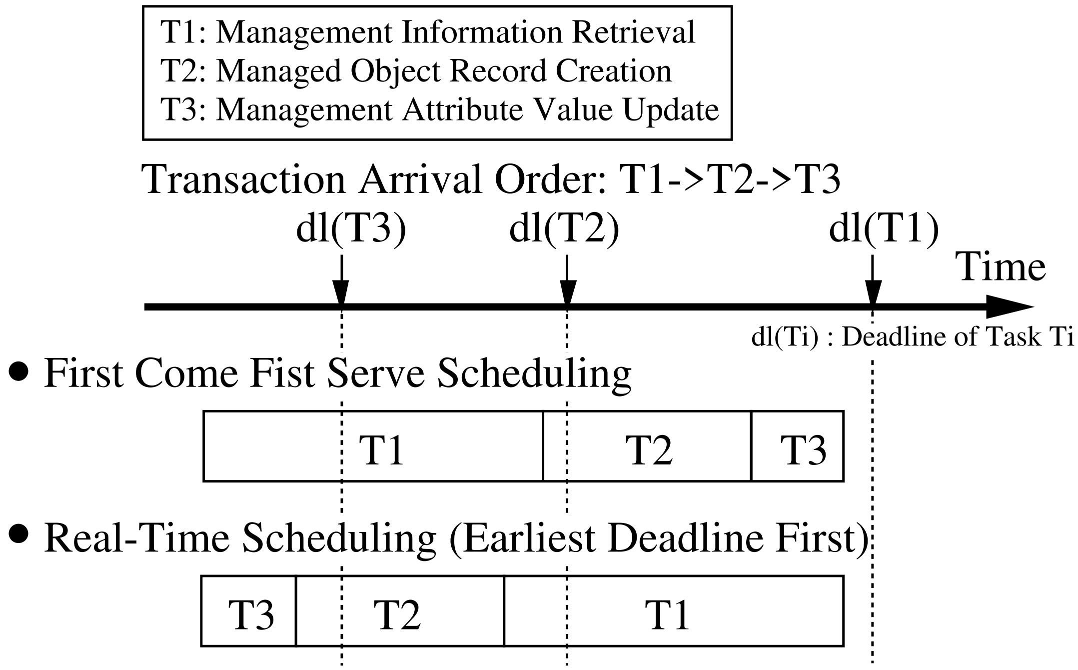

Figure 1: Scheduling Example for Management Operations

Figure 1 illustrates the difference between non real-time scheduling and real-time scheduling. A FCFS (First Come First Serve) algorithm, a non real-time scheduler, assigns a higher priority to a transaction that arrived earlier. A EDF (Earliest Deadline First) algorithm, a typical real-time scheduler, assigns a higher priority to a transaction that has an earlier deadline. In this example, three transactions arrive in the following order: T1 $\rightarrow$ T2 $\rightarrow$ T3, and the order of their deadlines is dl(T3) $\rightarrow$ dl(T2) $\rightarrow$ dl(T1). As illustrated, the T2 and T3 transactions scheduled by the non real-time scheduler cannot meet their deadlines. This causes a degradation of the user response and the quality of network services. However, all transactions that are scheduled by the real-time scheduler can meet their deadlines, even though the total processing time is the same in both scenarios. Through this scheduling example, the effect of the real-time scheduler is clearly recognized.

Taking into account the above discussion, this report proposes an active and real-time MIB architecture for a new MIB implementation by utilizing an RTDBS. The remainder of this report is organized as follows. Section 2 discusses previous work and research issues in both RTDBS and network management. Section 3 describes the new MIB architecture that provides management operations with active and real-time functionality. Section 4 describes our prototype system that has been developed based on the proposed architecture and which was used to evaluate the system behavior. Section 5 presents evaluation results showing the impact of real-time scheduling, communication delay, and concurrency control. Finally, this report concludes with some directions for further studies.

2 Previous Work

In this section we discuss previous studies in both real-time database systems and network management systems.

2.1 Real-Time Database Systems

Some of the key functional modules of a real-time database system (RTDBS) are transaction scheduling and concurrency control as illustrated in Figure 2. The transaction scheduling module stores received transactions in a queue, and schedules transaction execution by priority. The concurrency control module maintains data consistency taking into account the priority. A disk based RTDBS that stores data on disk, requires additional modules such as a buffer manager and a disk scheduler. They should also take into account the priority of transactions.

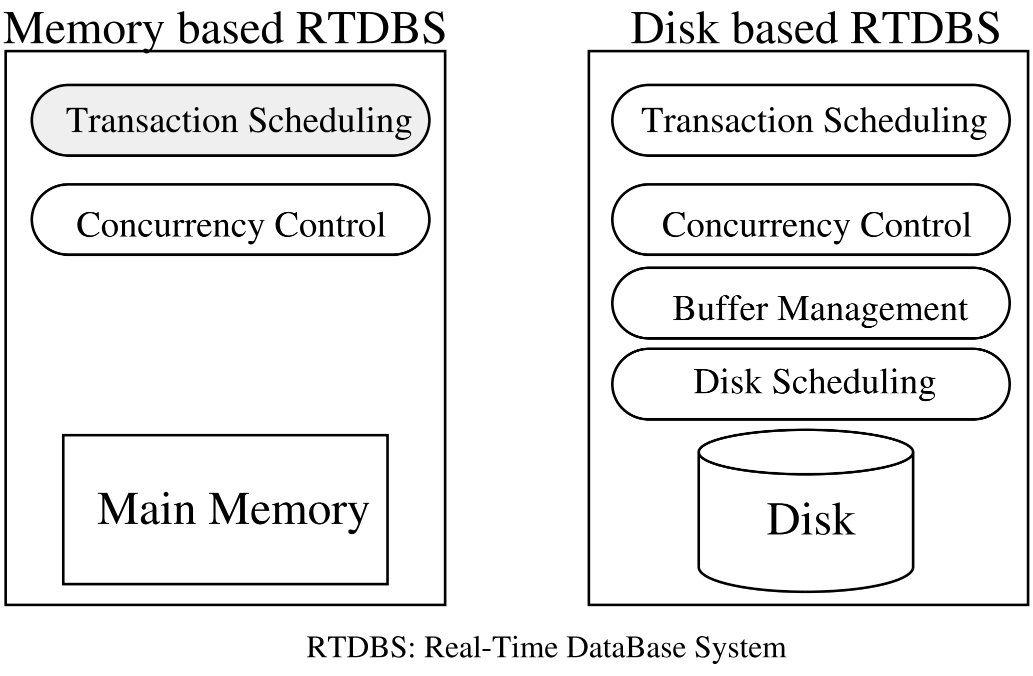

Figure 2: General RTDBS Configuration

In a network management system, the high-speed data access capability of the memory based RTDBS is more critical than the persistent data access capability of the disk based

RTDBS. Because of this, we focus on a memory based RTDBS in our study. The rest of this section, we discuss current technologies related to a memory based RTDBS as well as research issues that arise in a network management system.

2.1.1 Transaction Scheduling

When an RTDBS is used in a network management system, each management operation is executed as a transaction in order to maintain a database integrity. A transaction scheduler executes transactions based on their priority. Several scheduling algorithms based on different priority assignment policies have been proposed and evaluated[Huang90].

As an example two typical scheduling algorithms are briefly discussed. One is the EDF algorithm that has already been explained. The other one is HVF (Highest Value First) algorithm. For HVF, each transaction has a value that represents the importance to the application that the transaction completes before its deadline. HVF assigns a higher priority to a transaction that has a higher value. According to previous simulation studies, EDF minimizes the number of missed-deadline transactions in a system operating under low level of resource and data contention. However, it does not perform well when the system becomes over-loaded. This is because, under heavy loading, transactions gain high priority only when they are close to their deadlines. Gaining high priority at this stage may not leave sufficient time for completing transactions before their deadlines. In order to improve the performance of the real-time scheduling, especially under an overload situation, the HVF algorithm was designed. Since transactions under HVF gain high priority at earlier stage than under EDF, more transactions can have sufficient time to complete.

Which algorithm can provide the best performance depends on the application's properties such as the variety of transactions types, data contention, system load, and treatment of missed-deadline transactions (discarded or continued). Because of this, we should carefully choose the proper scheduling algorithm in an implementation phase. Furthermore, it is also important to design backup and recovery transactions to guarantee the database reliability.

2.1.2 Priority Assignment

Closely related to transaction scheduling, a priority assignment for each transaction is one of the most important issues in an RTDBS. Conventionally, a deadline is used as a static value of transaction priority. Priority can also be viewed as a value function[Jensen85] that is dynamically assigned considering the status of a transaction. The value function of a transaction measures how valuable it is to complete the transaction at some point in time after the transaction arrives. This function must be carefully selected to yield good performance.

Another problem is that of subtask priority assignment in case that a transaction requires distributed processing. Since network management tasks might be deployed in a distributed environment, it is a serious problem. Efficient algorithms have been studied for cases where the transaction structure is known in advance[Kao94], or for cases where a transaction is discarded if it misses its deadline[Purimetla94]. However, these assumptions may not be always applicable for management operations, so we believe better algorithms are required.

2.1.3 Concurrency Control

In case that multiple transactions are executed concurrently, a concurrency control mechanism maintains data consistency by scheduling shared data access. In conventional database products, a 2PL (two phase locking) algorithm is generally utilized. However, some extensions are needed for an RTDBS in order to deal with the priority inversion problem[Abbott92]. This is a phenomenon where a high priority lock requester waits on a low priority lock holder to finish. In order to avoid such situation, a 2PL-HP (high priority) algorithm, for example, chooses to abort the low priority lock holder and lets the high priority lock requester proceed.

Although lock-based concurrency control algorithms can provide correctness, their overhead may be significant in some real-time application environments. So several optimistic concurrency control algorithms have been studied[Haritsa90]. Optimistic concurrency control algorithms have the advantages of being non-blocking and deadlock free. According to simulation studies, the optimistic approach can outperform the lock-based approach when data contention is low and transactions that missed their deadline are discarded.

The choice of a good concurrency control algorithm is important. We note that locking-based algorithms can easily extend to a distributed version, however, optimistic algorithms should be required complex compensating procedures for aborted transactions in distributed environments.

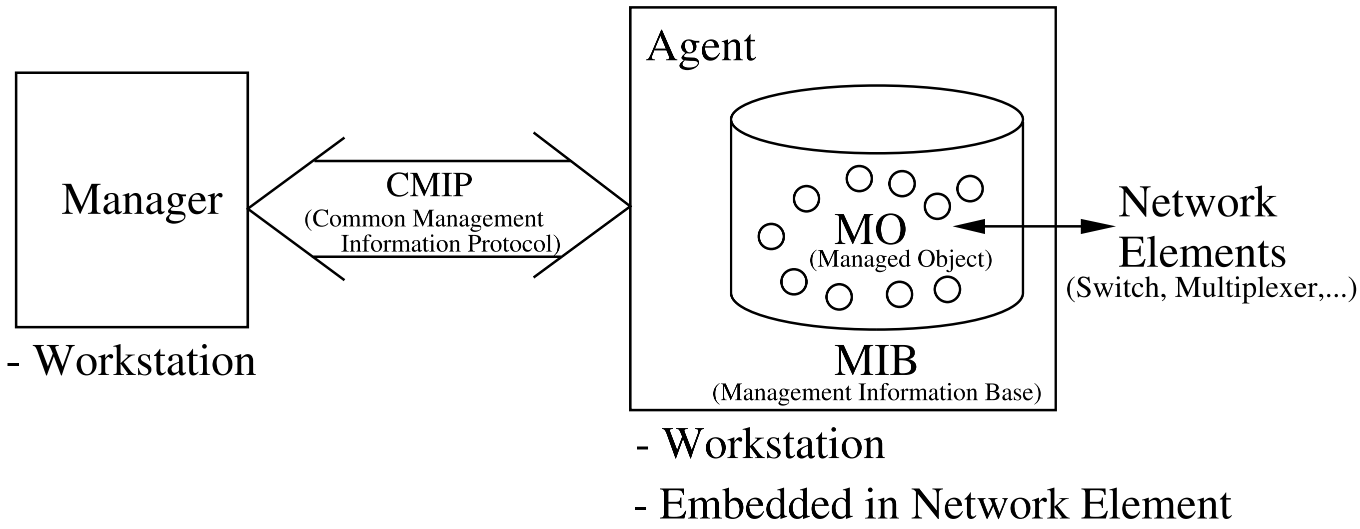

Figure 3: OSI Network Management Model

2.2 Network Management Systems

In standardized TMN architecture, a network management system consists of a manager and an agent, as depicted in Fig. 3.

The manager provides network administrators with various management functionality by cooperating with multiple agents. This cooperation is achieved by using operations that are transmitted by a standardized communication protocol called CMIP (Common Management Information Protocol). The agent controls a managed network element according to the manager's operation requests. The managed network element is modeled by using object-oriented techniques, and is represented as a group of managed objects. Each management datum corresponds to an attribute value of a managed object. The collection of managed objects is referred to a MIB. According to the OSI network management model, the MIB function is provided by an agent. However, in almost all actual management systems, static data for the managed networks and services are stored in a manager's databases for performance reason. On the other hand, dynamic data, such as the current status of each network element, is stored in an agent. When the manager requests dynamic data, the manager collects it from the corresponding agents, and temporarily stores it. Conventional database systems have been utilized as components of both managers and agents.

With regard to an MIB implementation, previous research efforts have focused on efficient utilization of commercial database systems such as relational database systems [Wasson90] [Kiriha91] and object-oriented database systems[Bapat91][Kiriha94]. The main goal of such MIB is to provide various types of queries with a fast average response time.

The standardized manager and agent model is a useful concept for increasing interoperability of management operations in heterogeneous (i.e., multi-vendor) networks. However, current MIB implementations (utilizing conventional database systems) based on this model do not always meet requirements to manage evolving networks and services. One problem is that an expected response time cannot be ensured. Even though an operation is important and urgent, the operation may have to wait until the completion of the one currently being executed. In order to deal with this problem, it is obvious that each operation has to be assigned a priority, and the execution order of such operations should be efficiently scheduled according to their priority. This is a main motivation of our study.

Another problem is that a manager's load may become very heavy. This is because, in the current management architecture, only one manager controls distributed multiple agents in a centralized manner as illustrated in Figure 4(a). To cope with this problem, distributing the manager's heavy load into multiple agents could be an effective solution. In order to distribute the system load, current passive(i.e., controlled by a manager) agents have to be delegated some parts of the Manager's role[Yemini91], and have to become active (i.e., controlled on their own). To realize such agents, communication between agents should be supported, as depicted in Figure 4(b). For example, a virtual path creation operation that requires cooperation of path-related agents can be efficiently implemented by using the communication between agents. This distrusted data management allows management systems to improve performance as well as to reduce the manager's load.

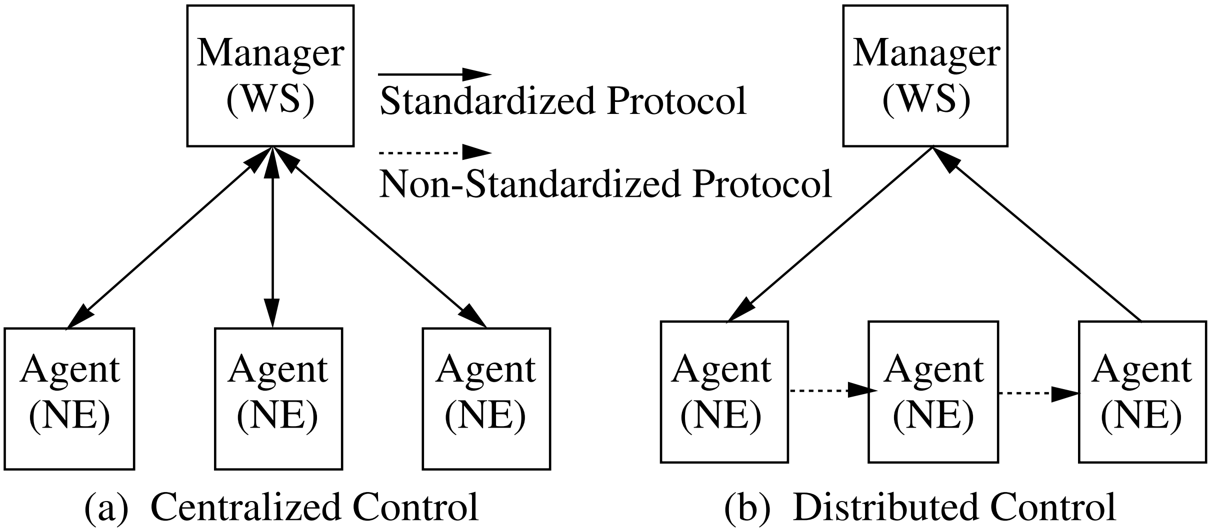

Figure 4: Management Interactions: Centralized and Distributed

3 Active and Real-time MIB Architecture

As described in Section 2, a conventional MIB should support active and real-time capabilities. As a solution, we propose to employ an RTDBS that can provide such functionality, as a new MIB module. In this section, we introduce an active and real-time MIB (ART-MIB) architecture, and discuss a mapping scheme between management operations and transactions in the new MIB.

3.1 ART-MIB Architecture

In order to provide network management systems with real-time capability, we propose a new MIB architecture that utilize functionalities of an RTDBS. Figure 5 illustrates the software components of the proposed ART-MIB architecture. In the following, we assume that the ART-MIB will be installed on every manager and agent as depicted in Figure 3.

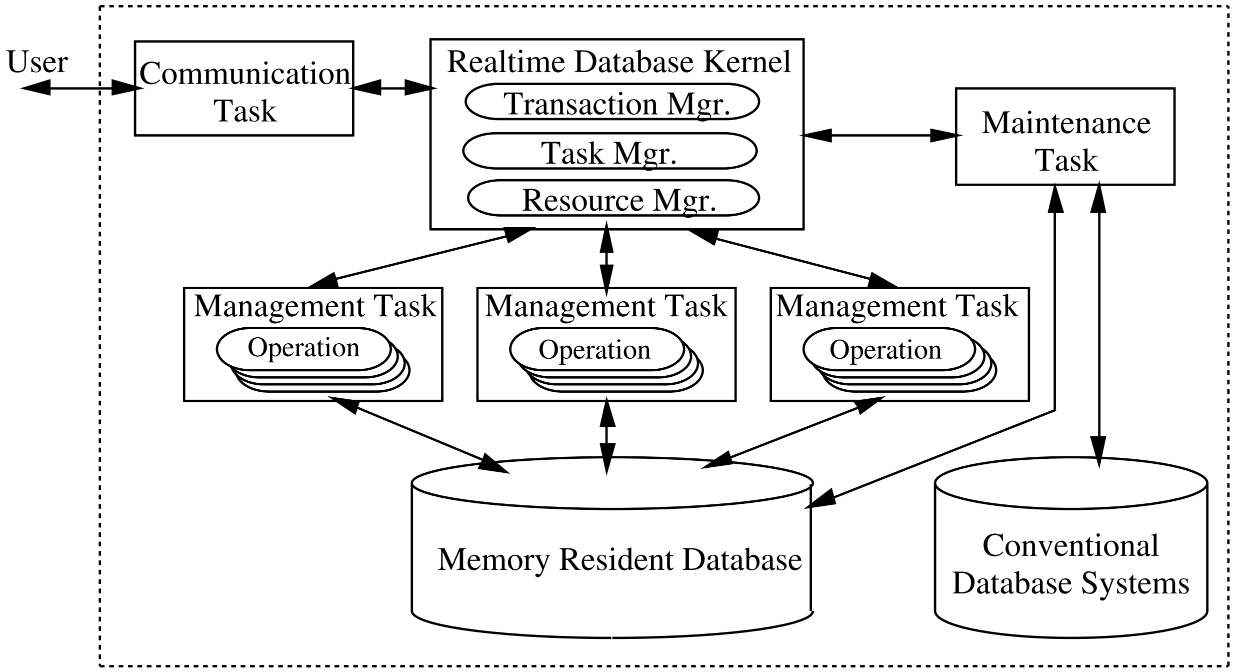

Figure 5: Active and Real-time MIB Architecture

The main component of this architecture comprises a memory resident database, management tasks, and a real-time database kernel. The memory resident database[Garcia92] stores management data in memory, and provides atomic data access methods such as data read and write. By using this, a management system based on our data management architecture can realize faster and more predictable data access functions. The management tasks provide high-level data processing functionality (i.e., management operations) such as connection set-up and bandwidth allocation. These functions are allowed to trigger other functions or to be executed periodically. Furthermore, multiple management tasks in the ART-MIB share the management data, and these are executed concurrently in order to improve performance.

The real-time database kernel is one of the most important modules of the proposed architecture. As illustrated in Figure 5, the kernel consists of three modules: a transaction manager, a task manager, and a resource manger. The three modules provide transaction scheduling, priority assignment, and concurrency control (as described in Section 2.1) respectively. The transaction manager schedules requested operations in a timely and event driven manner. The task manager tracks the status of tasks, and has responsibility for assigning deadlines or values to management operations. The resource manager provides a concurrency control mechanism for ensuring data consistency.

In addition to these components, a communication task and a maintenance task are also supported. The communication task receives an operation, passes the operation to the real-time database kernel, and provides the result of the operation. Since the real-time database kernel needs to know the real-time properties of the requested operation, the communication task provides the user application with an extended application program interface (i.e., API) that can specify some of these properties, such as timing constraints, periodicity, and triggering event conditions. The maintenance task is controlled by the real-time database kernel, and coordinates the memory resident database and conventional databases in order to provide backup and recovery functions.

As described in Section 2.1, further studies are required before deploying this architecture in an actual network management system. Because of this, we cannot determine the implementation alternatives for each functional module. However, our expectations are briefly discussed. First, for the transaction manager, we expect a dynamic scheduling algorithm that can utilize the various conventional algorithms according to the actual system status that is derived from system load, data contention, and required deadline strictness. Second, for the task manager, we also expect a dynamic deadline and value assignment algorithm. As described in Section 2.1, an end-to-end deadline guarantee is strongly desired for the network management applications that are inherently distributed. Last, for the resource manager, we expect a concurrency control algorithm that can resolve data contention in an optimistic way.

Ideally, we would study the characteristics of all functional modules. However, total system behavior might be quite complex. Hence in this report, we focus on transaction scheduling of the real-time database kernel. Furthermore, in the kernel that is a memory based RTDBS, we believe that the concurrency control overhead is not so serious. This is because transaction execution time is very small in a memory based system, and thus data-contention is generally low. In our study we focus on a memory based RTDBS, and hence do not seriously consider concurrency control alternatives.

3.2 Transaction Mapping

In order to make the ART-MIB architecture compatible with the OSI network management model, one management operation should basically correspond to one transaction shown in Figure 6(a). The previous discussion assumes this type of transaction mapping. We refer to this type of transaction as a CSV (conservative) implementation in the rest of this report. A CSV implementation can ensure strict data consistency. If a CSV transaction includes many data access operations, it tends to require a long completion time. Due to this, for a network management system that includes a large number of shared data access operations, a long CSV transaction may frequently block execution of other transactions.

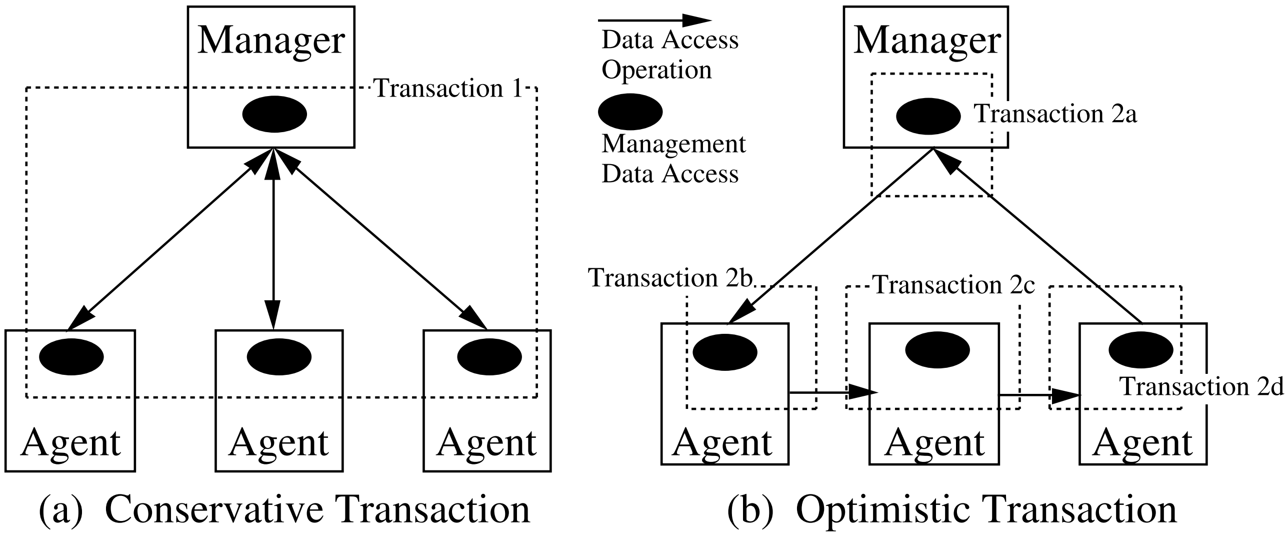

Figure 6: Two types of Operation Implementation: Conservative and Optimistic

On the other hand, Figure 6(b) illustrates the other alternative for management operations mapping. In this figure, one management operation corresponds to four transactions (Transaction 2a - 2d). We refer to this type of transaction as an OPT (optimistic) implementation in the rest of this report. Compared to a CSV implementation, an OPT implementation does not guarantee strict data consistency, and requires a compensating procedures in case of a transaction abortion. However, since OPT can execute each transaction in short time, data contention becomes less. This results in an improved degree of concurrency for transaction execution.

In this report we will study which implementation (CSV or OPT) is suitable for network managements, especially for our proposed ART-MIB. In particular, we study the impact of communication delay and transaction triggering overhead in both implementations. As observed in Figure 6, the communication delay for CSV is more serious than that for OPT. And the transaction triggering overhead for OPT is more serious than that for CSV. Furthermore, we note that if an OPT implementation is utilized to realize our ART-MIB, then the non-standardized communication protocol described in Section 2.2 needs to be developed.

4 Implementation of Evaluation System

In order to deploy the active and real-time MIB (ART-MIB) architecture in an actual network management system, it is important to develop a prototype system and to evaluate it. In this section, implementation issues such the prototype system configuration, managed network, and management operation specification are discussed.

4.1 System Configuration

In order to understand which scheduling algorithm (i.e., EDF or HVF) is better for network management systems, we have developed and evaluated a prototype system. Our evaluation system consists of a generator of management operations and a prototype MIB, as illustrated in Figure 7. The management operation generator sends randomly selected operations to the prototype MIB and periodically collects statistical information such as missed deadline percentage, execution time, queuing time, CPU utilization, and queue length. This configuration is compatible with the OSI network management model described in Section 2.2.

The prototype MIB consists of a memory based RTDB engine and management tasks. As the RTDB engine, we have utilized the STRIP (STanford Realtime Information Processor) Engine[Adelberg94] that is a memory based RTDBS, running on an HP workstation (HP Apollo 9000/735, HP-UX 9.03). The RTDB engine supports two scheduling algorithms: EDF and HVF (also, FCFS for comparison). However, one algorithm has to be selected before the engine starts. Furthermore, functions as transaction triggering and concurrency control (two phase lock with highest priority) are supported. In our evaluation, we assumed that neither deadlock nor preemption occur during transaction execution. We believe that this assumption is reasonable because the effect of deadlock and preemption are very small in a memory based RTDBS.

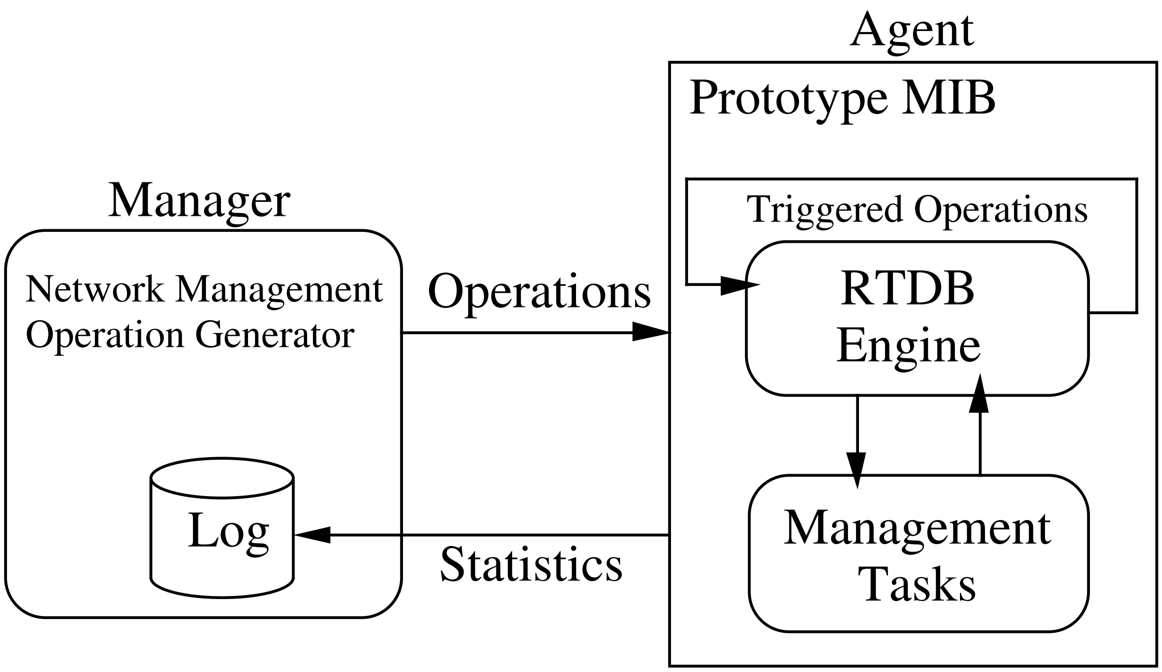

Figure 7: Evaluation System Configuration

The management tasks support not only basic network management operations (e.g., link status monitor and alarm report) but also provide management operations for ATM transmission networks (e.g., create, delete trace, and bandwidth update of virtual paths). Section 4.3 discusses the supported operations in more detail.

4.2 Managed Network Configuration

The evaluation system is assumed to manage an ATM transmission network. Before describing the management information, we briefly introduce several ATM network term. Figure 8 illustrates an example of an ATM transmission network. A virtual path connection (VP connection) is a multiplexed logical link between two network elements (e.g., ATM cross connect equipment) connected by a physical link. The VP connection is terminated in both network elements, and these terminations are uniquely identified as connection termination points (CTP). A virtual path trail (VP trail) is a logical link for a certain network service and consists of several virtual path connections. In Figure 8,

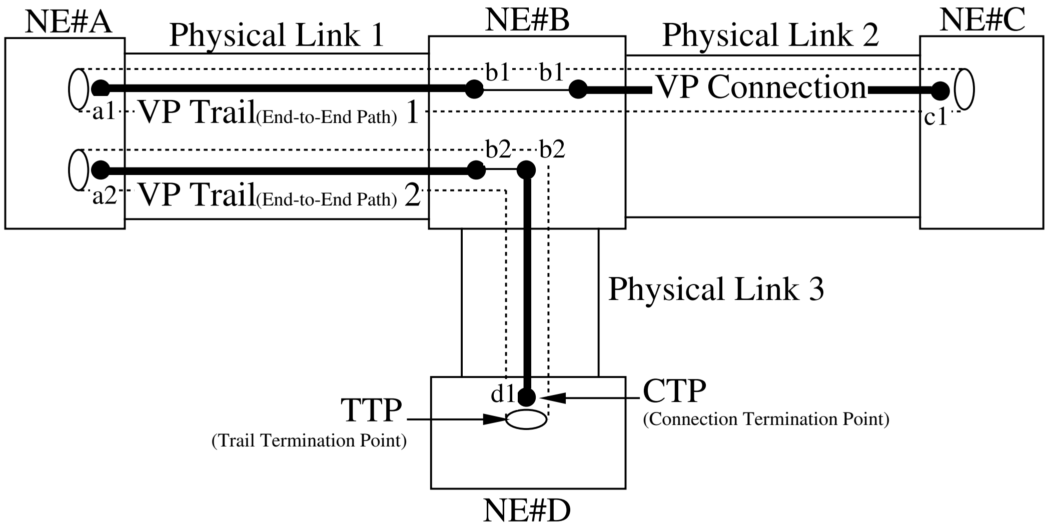

Figure 8: ATM Transmission Network

both VP trail 1 and 2 consist of two VP connections. The VP trail is also terminated in service-end network elements, and these terminations are identified as trail termination points (TTP).

We have designed a relational database schema that represents such an ATM transmission network, because the STRIP engine supports a relational data model. Figure 9 shows the database schema. Information concerning VP trails and physical links is stored in a TRAIL table and a PLINK table respectively. The TRAIL table includes such data elements as unique identifier, both TTPs, allocated bandwidth, and routing information. The PLINK table includes such data as unique identifier, available bandwidth, and connecting network element pair.

We note that VP connections are not explicitly represented. VP connections are managed through corresponding CTPs whose information is stored in each NE table. For example, the management information concerning the two VP connections in VP trail 1 appears in the first row (CTP='a1') of the NE#A table, the first row (CTP='b1') of the NE#B table, and the first row (CTP='c1') of the NE#C table.

TRAIL

Trailid	S-TTP	D-TTP	BW	ROUTE
1	a1	c1	100M	#A#B#C
2	a2	d1	100M	#A#B#D

PLINK

PLINKID	Available-BW	NE#
1	500M	#A#B
2	300M	#B#C
3	200M	#B#D

NE#A

CTP	D-CTP	PLINKID	TRAILID
a1	b1	1	1
a2	b2	1	2

NE#B

CTP	D-CTP	PLINKID	TRAILID
b1	c1	2	1
b2	d1	3	2

NE#C

CTP	D-CTP	PLINKID	TRAILID
c1	--	-- (termination)	1

NE#D

CTP	D-CTP	PLINKID	TRAILID
d1	--	-- (termination)	d

Figure 9: Relational Data Schema in Evaluation System

We assume that the TRAIL and PLINK tables are stored in a manager system, and each NE table is stored in an agent corresponding to each network element.

Although the managed network has already been specified by using a relational data model, we briefly discuss the mapping between the relational data schema and the OSI management information structure. In order to provide compatibility with the OSI environment, our data schema can be visualized as illustrated in Figure 10. In this figure, each row data in each table corresponds to a managed object that has several attributes to be managed. This figure also shows a containment relationship (e.g., Connection objects belong to a Link object) and a pointer relationship (e.g., a Trail object has a pointer attribute that indicates constituent Connection objects) among all managed objects. Considering data location, Link objects and Connection objects are assumed to be stored in an agent that corresponds to a NE. Other management data is assumed to be stored in a manager.

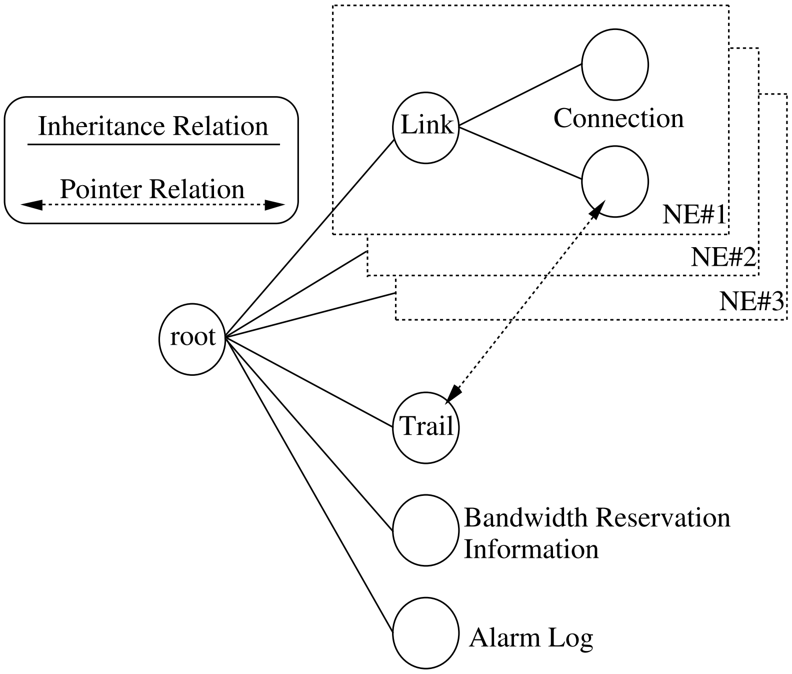

Figure 10: Management Information Tree (MIT) in Evaluation System

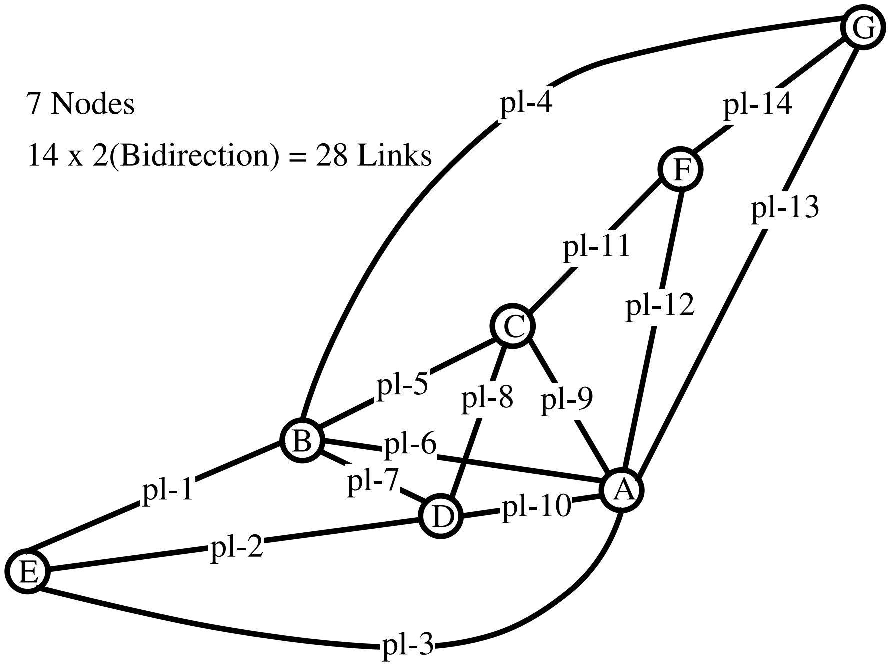

Figure 11: Evaluated Network Configuration

In our evaluation, we assume the network configuration depicted in Figure 11. As shown, we consider an ATM transmission network that consists of 7 network elements, and 28 physical links (i.e., each link is bidirectional). In our evaluation system, we have randomly generated management data for VP trails and VP connections that uses these network elements and physical links.

4.3 Management Operations

By using the management data described in the previous section, the evaluation system supports the following management operations:

- Create, Delete, and Trace for a Trail and constituent Connections
  (i.e., Create Trail, Delete Trail, Trace Trail, Create Connection, Delete Connection, and Trace Connection,)
- Bandwidth Allocation for a trail and constituent connections
  (i.e., Trail Bandwidth Update and Connection Bandwidth Update)
- Reservation based Bandwidth Control
  (i.e., Bandwidth Update Reservation and Reservation Execution)
- Status Monitor, Alarm Report, and Scanning Alarm for a Physical Link
  (i.e., Link Monitor, Alarm Report, and Scanning Alarm)
- Reroute of a VP Trail
  (i.e., Reroute)

A detailed specification of these operations appears in Attachment A of this report. We only note that such operations as reservation execution, link monitor, and scanning alarm are periodically executed. The rest of the operations are executed in an aperiodic fashion. In the following, we discuss such implementation issues of management operations such as utilizing a trigger mechanism, assigning deadlines and values, and the

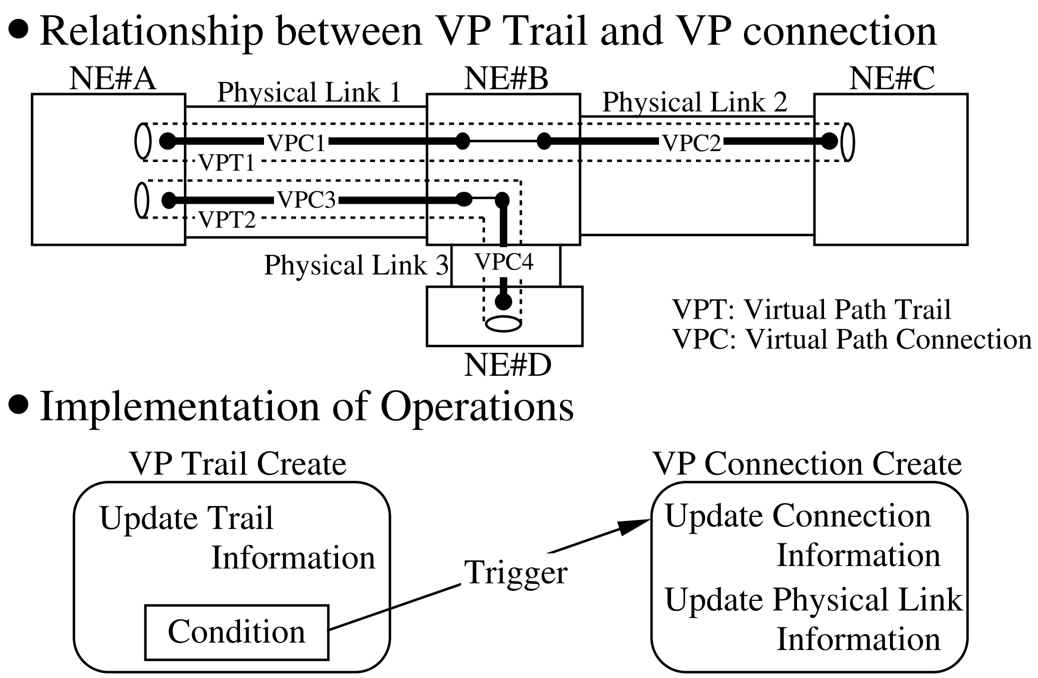

Figure 12: Management Operation Example

relationship among operations.

As an example, we discuss the create operation for an ATM virtual trail as depicted in Figure 12. This figure shows that the VP trail VPT1 consists of two virtual path connections: VPC1 and VPC2. Therefore, a create operation for a VP trail should invoke one or more create operations for the constituent VP connections (e.g., VPT1 creation requires VPC1 and VPC2 creation). Furthermore, the create operation must check whether the corresponding physical links have sufficient bandwidth for the corresponding VP connections before invoking their creation. We have implemented such a check by using the trigger mechanism. As a consequence, the VP trail create operation adds a new entry to the TRAIL table in Figure 9, and triggers the VP connection create operation as a new transaction if the above condition is true. The triggered create operations add some entries to corresponding NE table in Figure 9, and update the available bandwidth data in the PLINK table.

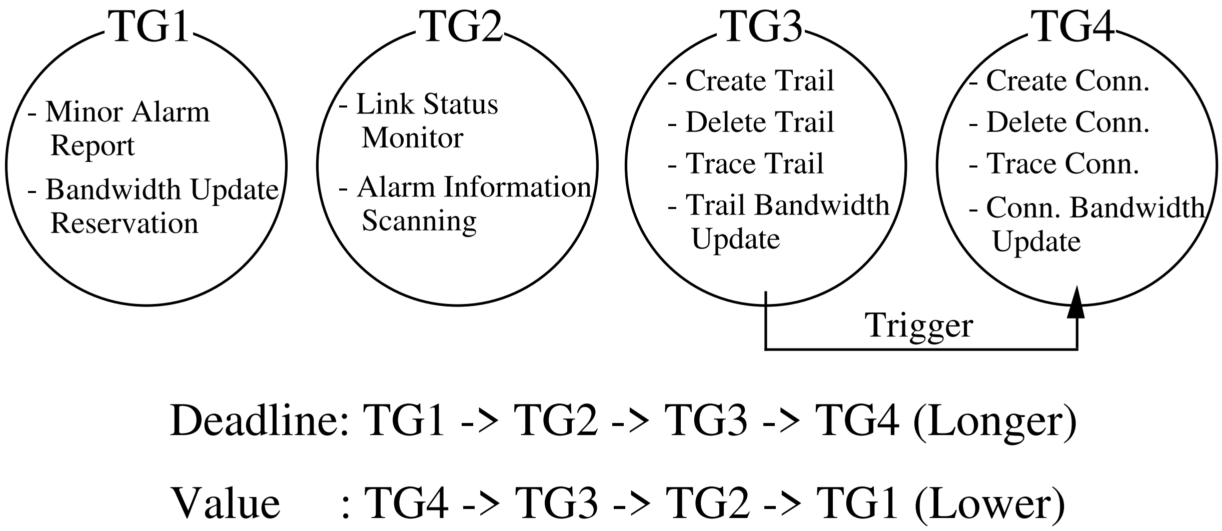

Figure 13: Transaction Groups: Their Deadline and Value

In order to assign a deadline and a value to each operation, we grouped supported operations into four transaction groups: TG1, TG2, TG3, and TG4, as illustrated in Figure 13.

As illustrated in the figure, TG1 includes light weight operations. Both examples in the figure require only adding one record to a table. TG2 includes such periodic operations as a link status monitor and scanning alarm information. Since TG2 operations require reading all information from a table, they take longer to complete than TG1 operations. TG3 includes such aperiodic operations as create, delete, trace and bandwidth update for virtual path trails. Compared to TG2 operations, TG3 operations require longer completion time, and it is more important to meet their deadlines. TG4 includes operations for virtual path connections that are triggered from TG3 operations. Due to this relationship, TG4 operations have to be given the highest value to complete before their deadline expires. Furthermore, TG4 operations are more complex than TG3 operations, since TG4 operations have to update information concerning one or more connections.

Taking this into account, we have assigned each task group a deadline and a value. For deadlines, operations in TG1 are assigned relatively short deadlines, TG2 operations are given deadlines further in the future, and so on. For the value, TG4 operations have the highest value, TG3 the next highest, and so on. Table 1 shows the actual value for their deadlines and values.

Table 1: Deadlines and Values for each Transaction Group

Task Group	Slack Time (msec)	Value
TG1	10	3
TG2	50	5
TG3	80	8
TG4	100	10

As described in the operation example, each operation is related to other operations. Figure 14 shows the relationships between all supported operations.

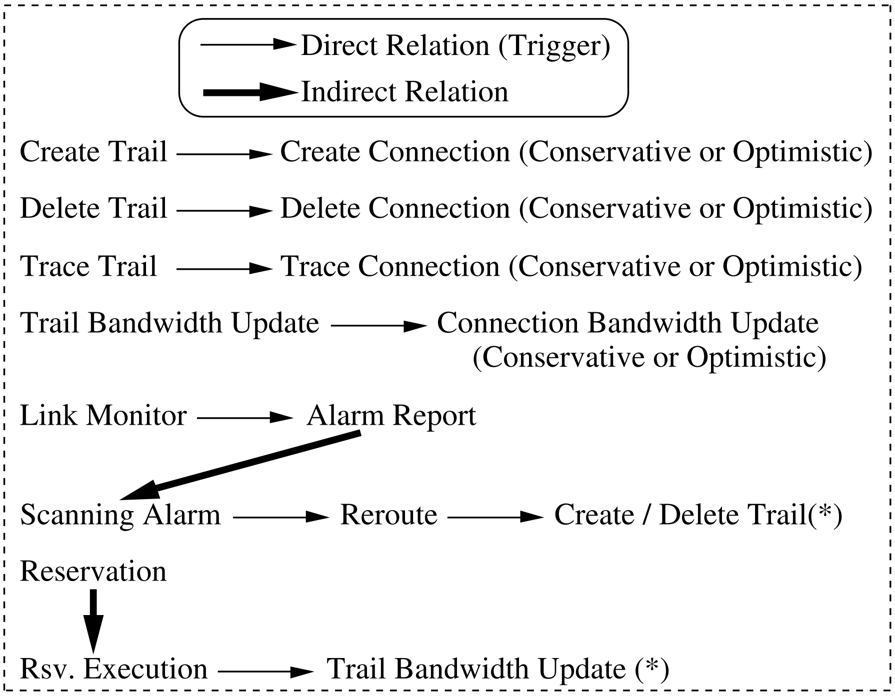

Figure 14: Supported Management Operations and their Relationships

In this figure, the direct relation means that one operation triggers another operation. And the indirect relation means that the number of previously executed operations affects the processing amount of another operation. For example, a reservation execution operation has to deal with the number of reservations registered by reservation operations executed before its execution.

In closing this section, we note that we have developed both CSV and OPT implementations (described in Section 3.2) for TG4 operations. In our evaluation study, the CSV implementation simulates a centrally controlled management system, and the OPT implementation simulates one controlled in a distributed manner.

5 Evaluation Results

We will discuss our evaluation results in this section. An overview of real-time scheduling characteristics is shown first, then the characteristics of two scheduling algorithms are discussed in more detail. In addition, we will show the impact of communication delay and concurrency control. Finally, we will study the crossover points where HVF outperforms EDF and vice versa.

5.1 Result Overview

First, we discuss three figures that describe RTDBS performance characteristics in a network management environment. In this evaluation, management information includes data for 100 trails, and the statistics shown have been collected after executing 300 operations (excluding the number of triggered operations).

Figure 15 shows the average execution time of three scheduling algorithms: FCFS (non real-time scheduler), EDF, and HVF. We note that the average execution time is almost all same even if the scheduling algorithm is different. The reason why the average execution time increases as the mean inter-arrival time decreases is that the queuing delay increases under the overload situation.

Figure 16 shows the missed deadline percentage (MDP) for the three scheduling algorithms. We note that both real-time scheduling algorithms outperform FCFS in any case. EDF outperforms HVF when the mean inter-arrival is longer than 0.5 msec. On the other hand, when the mean inter-arrival time becomes shorter than 0.5 msec, HVF outperforms EDF. The following describes why this occurred in the overload situation. For EDF scheduling, since TG4 operations are scheduled later than the other operations, TG4 operations take longer time to complete under an overload situation than in a low load situation. This phenomenon seriously affects the completion time of the other operations whose deadlines are shorter than TG4 operations. On the other hand, for HVF scheduling, since TG4 operations are scheduled earlier, the impact of the completion time of TG4 operations on the other operations is smaller than that for EDF, even if the system becomes over-loaded.

We believe that these evaluation results validate that an RTDBS, especially with real-time scheduling, is useful for a network management system. In order to deploy a new MIB utilizing the RTDBS we should understand the precise conditions where HVF outperforms EDF.

In order to more precisely know when HVF outperforms the EDF scheduler, we have studied the maximum queue length. We believe that the maximum queue length (MQL) precisely represents the system load. This is because, since the evaluated system includes triggered transactions, the mean inter-arrival time is not always representative of the system load.

In Figure 17, two curves, MDP-EDF and MDP-HVF, show the missed deadline percentage versus the number of transactions to be processed during a constant time (10 sec). The other two dotted curves, MQL-EDF and MQL-HVF, show the maximum queue length. As this figure shows, the trend of the MDP curve is as same as the one for the MQL curve, for both EDF and HVF. From this figure, we can observe the precise conditions where HVF outperforms EDF. We note the following evaluation results.

- The number of transaction when HVF begins to outperform EDF is about 540.
- If the queue length under EDF becomes larger than 40, the system would better off switching the scheduling algorithm from EDF to HVF.
- If the queue length under HVF becomes less than 20, the system would better off switching the scheduling algorithm from HVF to EDF.

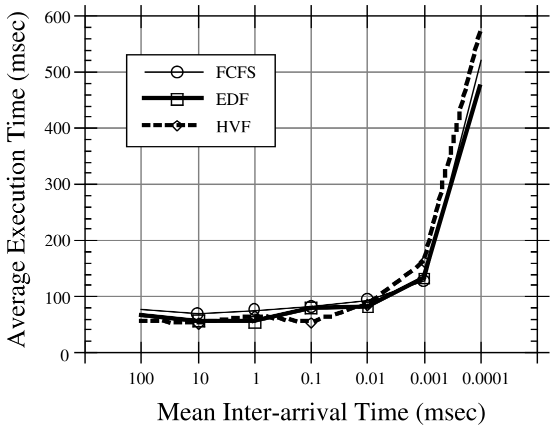

Figure 15: Average Execution Time Comparison

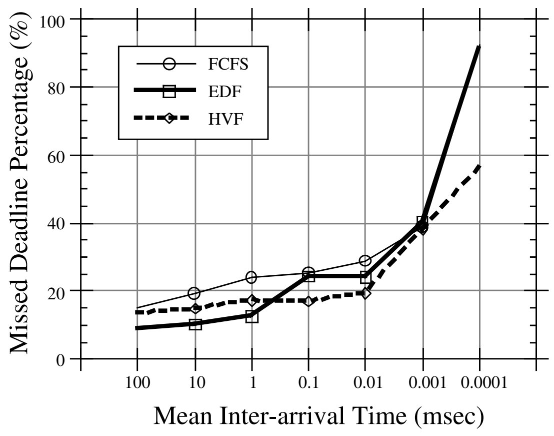

Figure 16: Missed Deadline Percentage Comparison

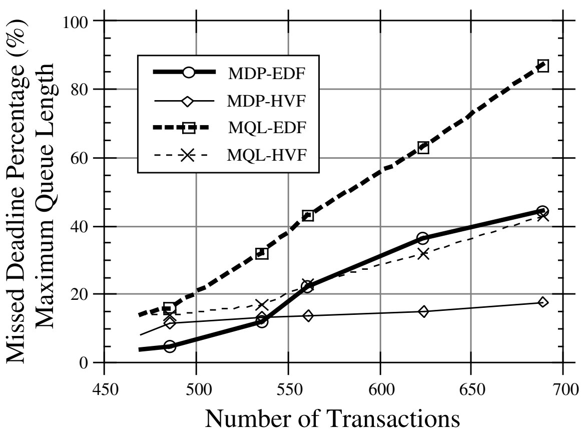

Figure 17: Characteristics of Maximum Queue Length

5.2 Characteristics of Real-Time Scheduling

As described in Section 4, the evaluation system has implemented TG4 operations by using two transaction types: CSV (conservative) and OPT (optimistic). In this section, we discuss the characteristics of four implementations: EDF (CSV), EDF (OPT), HVF (CSV), and HVF (OPT). In this evaluation, management information includes data for 100 trails.

First of all, Figures 18 - 20 show the total execution time of the four implementations. The three figures show results after execution of 100 operations, 200 operations, and 300 operations respectively. We note that there is no difference among the four curves in all of three figures. Because of this fact, following two assumptions will be verified that they are true. First, we can determine which scheduling algorithm is better, from the MDP criteria that will be shown in the following figures (i.e., Figures 21 - 27). In the following of this section, we will present MDP results. Second, the overhead time of triggering transactions in OPT implementations is quite small compared to the total execution time due.

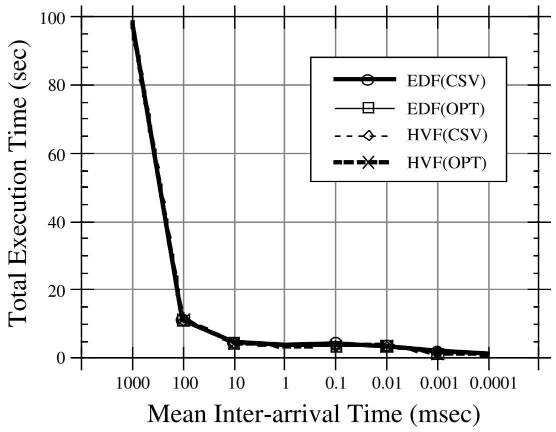

Figure 18: Total Execution Time after 100 Operations Execution

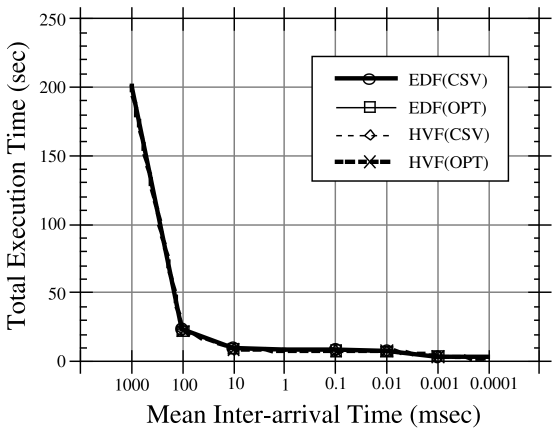

Figure 19: Total Execution Time after 200 Operations Execution

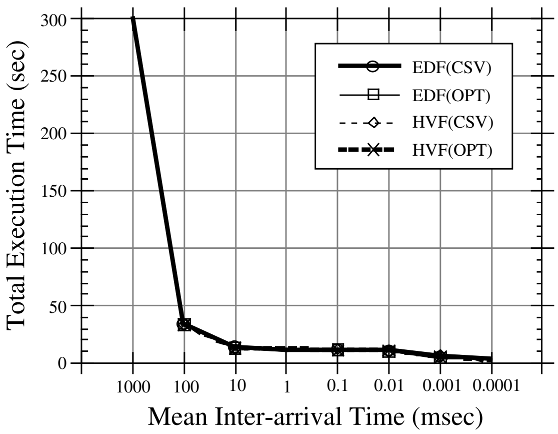

Figure 20: Total Execution Time after 300 Operations Execution

Figures 21 - 23 show the MDP of four implementations: EDF (CSV), EDF (OPT), HVF (CSV), and HVF (OPT). The three figures show the results after execution of 100 operations, 200 operations, and 300 operations respectively.

As observed from the three figures, MDP increases when the number of executed operations increases for all implementations. This is because, increasing the number of triggered transactions makes the system (i.e., the prototype MIB) be over-loaded. In our evaluation scenario, alarm report operations on a certain physical link stores their results in a database. And if the number of alarm report records goes over a threshold value (detected by a scanning alarm operation), then the system triggered delete and create operations for several VP trails that uses the physical link (i.e.,a reroute operations). As the number of executed operations increases over time, such triggered operations increase due to the stored alarm report records. Actually for EDF, the performance is better than HVF in almost all cases in Figure 21. However, EDF performance degrades in Figure 22, and it is outperformed by HVF (OPT) in all cases, in Figure 23.

We note that EDF performance degrades not only with short mean inter-arrival times but also with long ones. In Figure 22, both EDF implementations increase their MDP when the mean inter-arrival time is 1000 msec (1 sec). This is also because that increasing the number of triggered transactions makes the system be over-loaded temporarily. In our operation scenario, a link status monitor operation triggers an alarm report operation, and a scanning alarm operation triggers a reroute operation. As described in the previous paragraph, the scanning alarm operation determines whether a reroute operation should be triggered or not, from the number of alarm report records. Since a scanning alarm operation has a longer slack time than that of an alarm report operation, a larger number of triggered alarm report operations might be executed before executing a scanning alarm operation, in case that the mean inter-arrival time is long. As a result of this, a scanning alarm operation needs to read many alarm report records, and triggers many reroute operations.

As previous simulation studies, this result shows that the real-time scheduling performance depends on the system (i.e., management system) load. Of course, if the mean inter-arrival time becomes shorter, then the system becomes over-loaded. Furthermore, the number of triggered transactions also affect the system load. For example, in our operation scenario, such operations as alarm report, bandwidth update reservation, bandwidth update execution , and reroute affect the number of triggered transactions. If such operations increase, the performance of real-time scheduling is similar to, or becomes worse than that in Figure 23. On the other hand, if such operations are not frequent, the performance of real-time scheduling is similar to that in Figure 21.

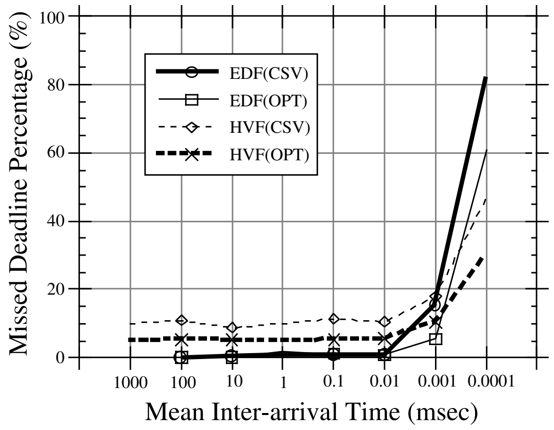

Figure 21: Missed Deadline Percentage after 100 Operations Execution

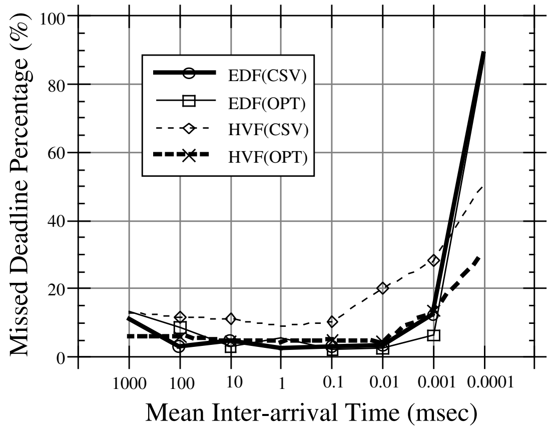

Figure 22: Missed Deadline Percentage after 200 Operations Execution

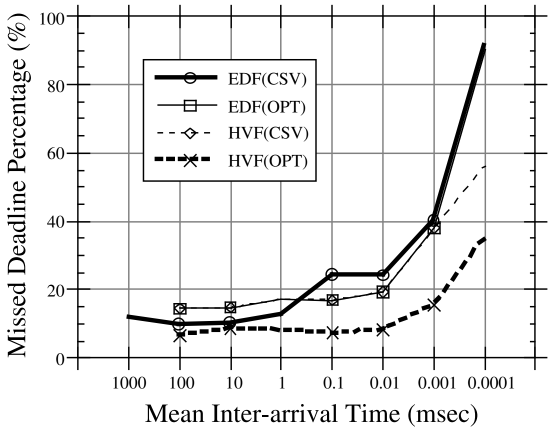

Figure 23: Missed Deadline Percentage after 300 Operations Execution

In order to better understand the differences between the scheduling algorithms, the next four figures (i.e., Figures 24 - 27) show the MDP in a different way.

Figure 24 and 25 show the MDP of three snapshots (i.e., after 100, 200, 300 operations execution) for EDF (CSV) and EDF (OPT) respectively. For EDF, TG3 and T4 operations that account for a large number of total executed operations, begin to miss their deadlines in an overload situation. Because of this, when the system load increases, a rate of MDP degradation becomes large, as observed in the two figures. The trends for CSV and OPT curves are similar. However, we note that EDF (CSV) shows a little better MDP under a low load (e.g., 100 msec - 0.01 msec) in two cases after executing 100 and 200 operations.

Figures 26 and 27 show the MDP of three snapshots for HVF (CSV) and HVF (OPT) respectively. For HVF, TG1 and T2 operations that account only for a small number of total executed operations, begin to miss their deadline in an overload situation. Because of this, even though the system load increases, a rate of MDP degradation is not so large as in the earlier figures. Furthermore, we note that the performance degradation of HVF (OPT) is smaller than that for HVF (CSV). Compared to CSV, a larger number of TG4 operations is triggered in OPT. Therefore, for HVF (OPT), the ratio of the number transactions that can meet their deadlines, to the total number of executed operations, becomes larger than that for HVF (CSV). Because of this, the MDP of HVF (OPT) becomes relatively smaller than that of HVF (CSV).

From the results in this section, we conclude that a MIB would better off utilizing EDF (CSV) in a low load scenario, and would better off utilizing HVF (OPT) in an overload situation.

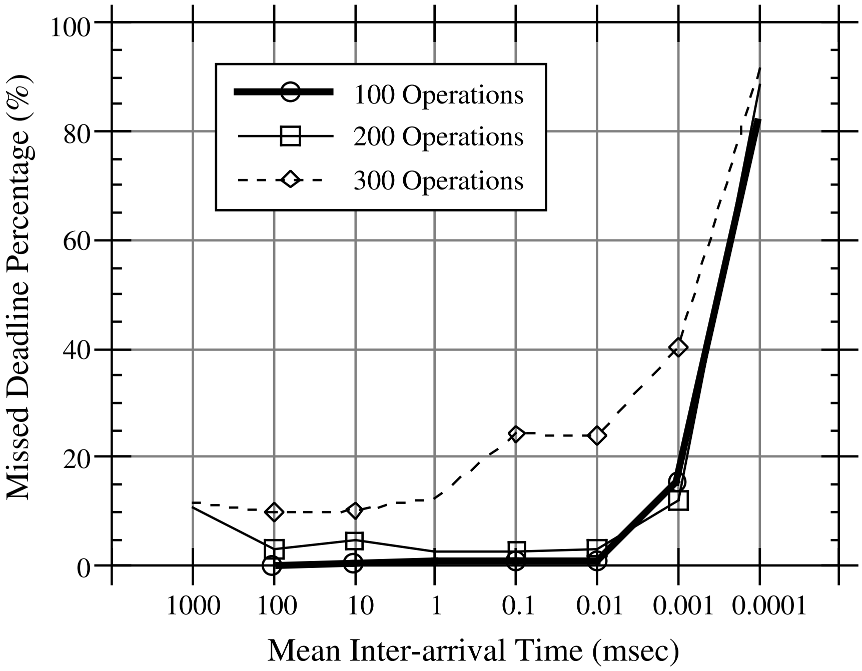

Figure 24: MDP Characteristics of EDF (Conservative)

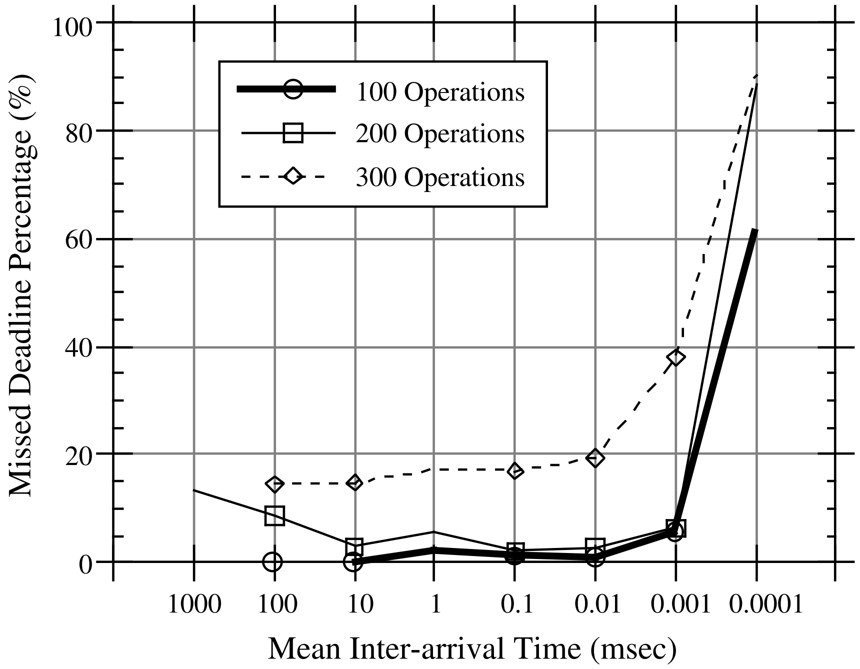

Figure 25: MDP Characteristics of EDF (Optimistic)

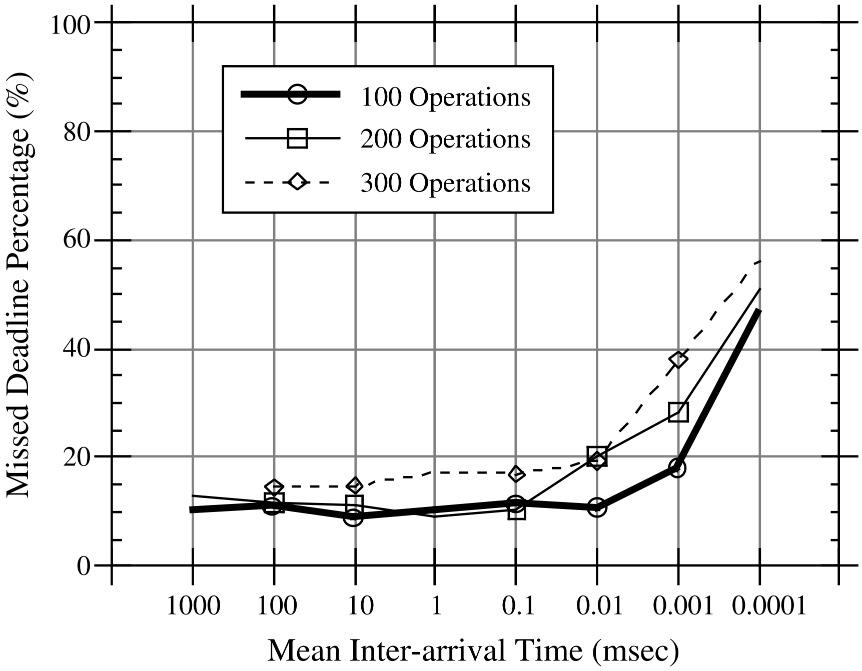

Figure 26: MDP Characteristics of HVF (Conservative)

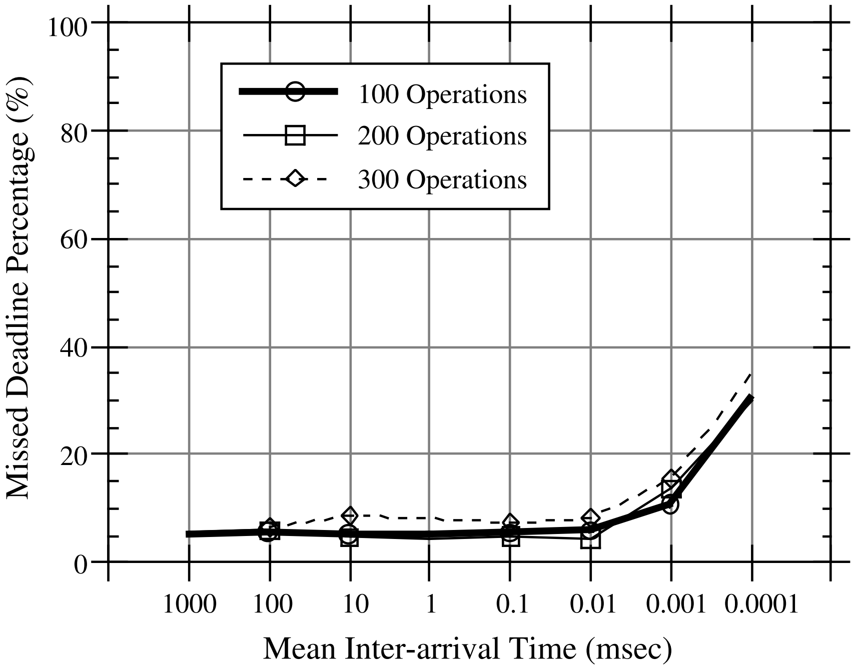

Figure 27: MDP Characteristics of HVF (Optimistic)

5.3 Impact of Communication Delay

Although the evaluation has been done in one workstation, TG4 operations assumes a distributed transaction. In order to simulate such a distributed environment, each operation in TG4 includes a communication delay parameter. The communication delay in the previous results was 10 msec, which is an average performance for a simple RPC (Remote Procedure Call). In this section, we discuss the impact of communication delay. In this evaluation, management information includes data for 100 trails, and the statistics have been collected after executing 200 operations.

Figures 28 - 30 show the MDP of four implementations: EDF (CSV), EDF (OPT), HVF (CSV), HVF (OPT). The three figures correspond to communication delays of 0 msec, 10 msec, and 50 msec respectively.

The trends when communication delay is 0 msec and 10 msec are not that different. This is because the slack time of TG4 operations is substantially larger than the communication delay in our evaluation environment. However, when the communication delay is 50 msec, we note that both EDF (CSV) and HVF (CSV), which require larger number of remote calls than the OPT implementation, significantly degrade their performance. Because of this, if a management system faces heavy communication traffic, it is better to utilize the OPT implementation that requires fewer number of calls.

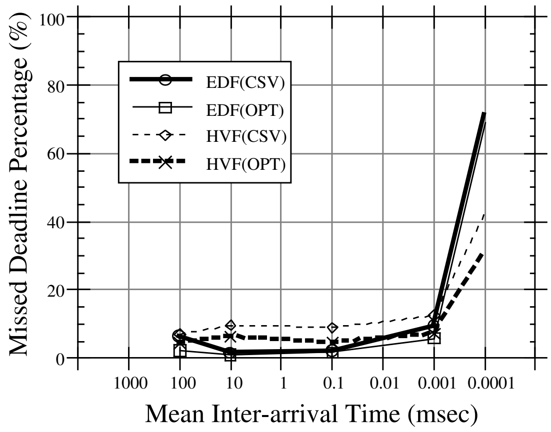

Figure 28: MDP for 0msec Communication Delay

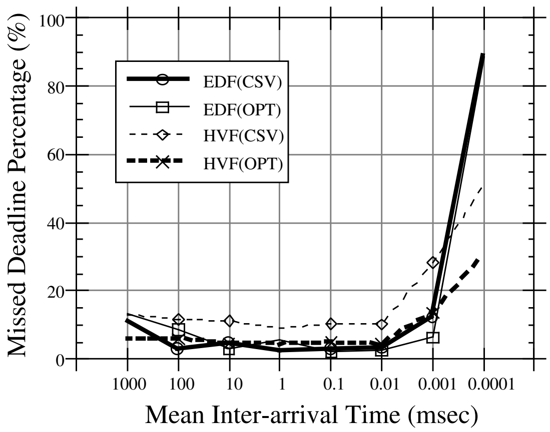

Figure 29: MDP for 10msec Communication Delay

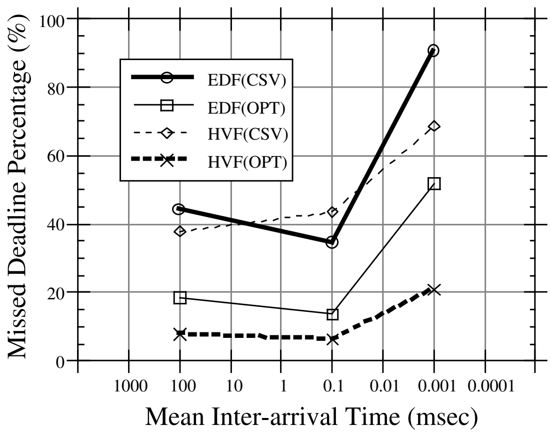

Figure 30: MDP for 50msec Communication Delay

In order to better understand the differences between the scheduling algorithms, next four figures (i.e., Figures 31 - 34) show the MDP in a different way.

Figures 31 and 32 show the MDP for three communication delays for EDF (CSV) and EDF (OPT) respectively. From the two figures, the significant performance degradation of EDF (CSV) with high communication delay clearly recognized. This is because that the large communication delay makes the completion time of TG4 operations longer, which overloads the system.

Figures 33 and 34 show the MDP for the three delay cases for HVF (CSV) and HVF (OPT) respectively. For HVF (OPT), the performance degradation is small even if the communication delay is larger. However, if the communication delay approaches the slack time of TG4 operations, then the performance will degrade even in the HVF (OPT). We note that the performance degradation of HVF (CSV) is smaller than that of EDF(CSV). The reason is that HVF tries to meet deadlines of TG4 operations that are significantly delayed by the remote calls.

We believe that communication delay is also the constituent of the system load. If the communication delay becomes larger, then the processing time for each transaction also becomes longer. This situation is similar to a situation when the system load becomes heavy. Take this into account, these results also show that EDF (CSV) is better at low load, while HVF (OPT) is better in an overload situation.

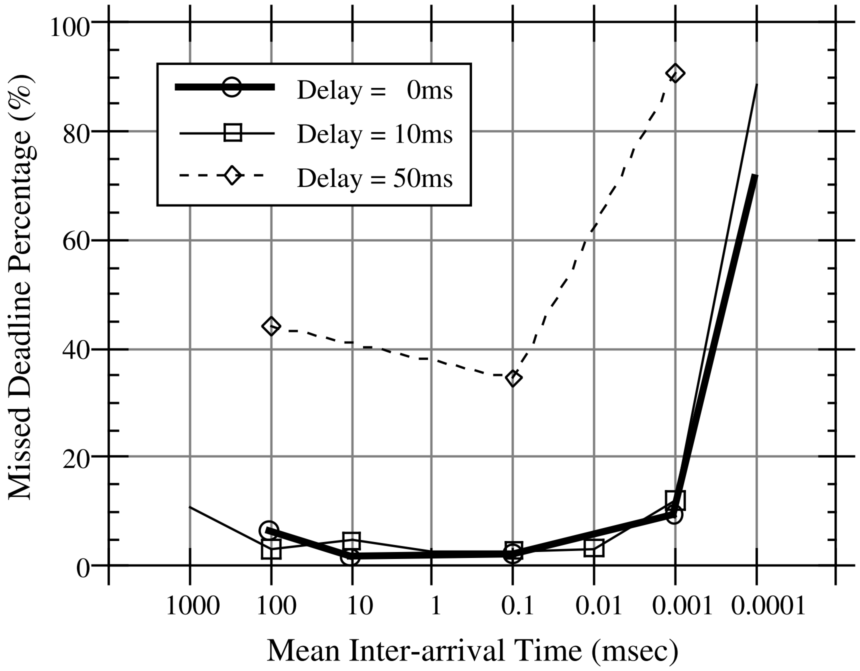

Figure 31: Impact of Communication Delay on EDF (Conservative)

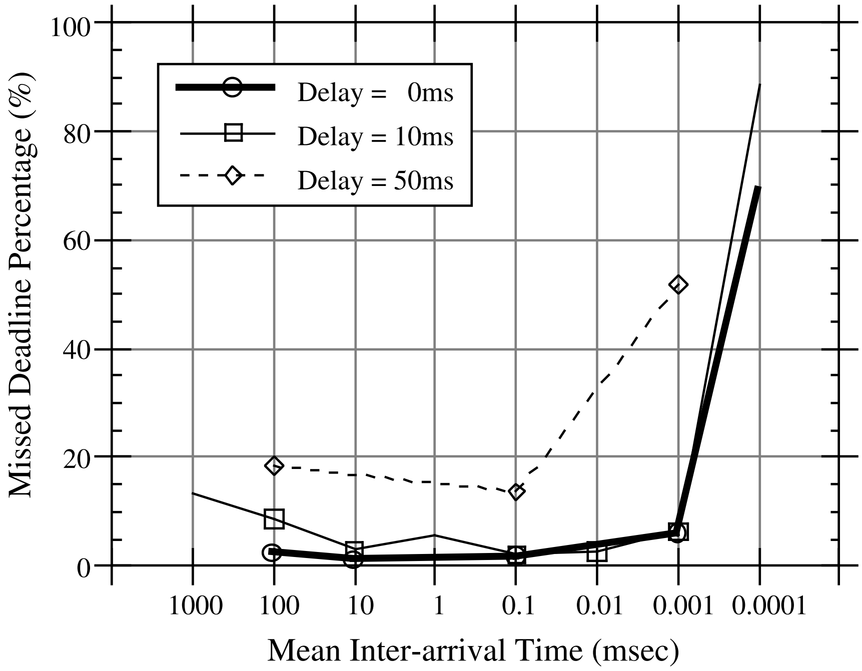

Figure 32: Impact of Communication Delay on EDF (Optimistic)

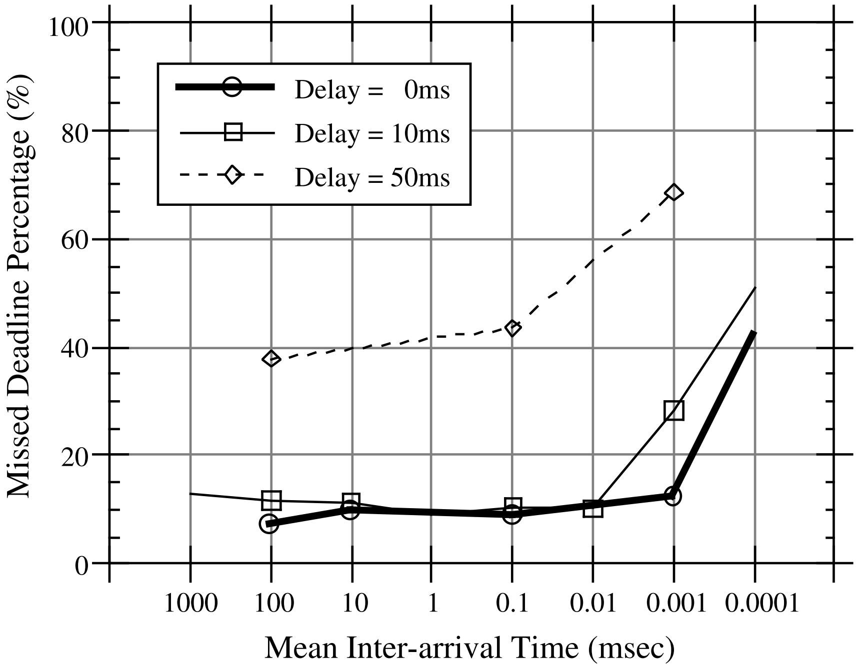

Figure 33: Impact of Communication Delay on HVF (Conservative)

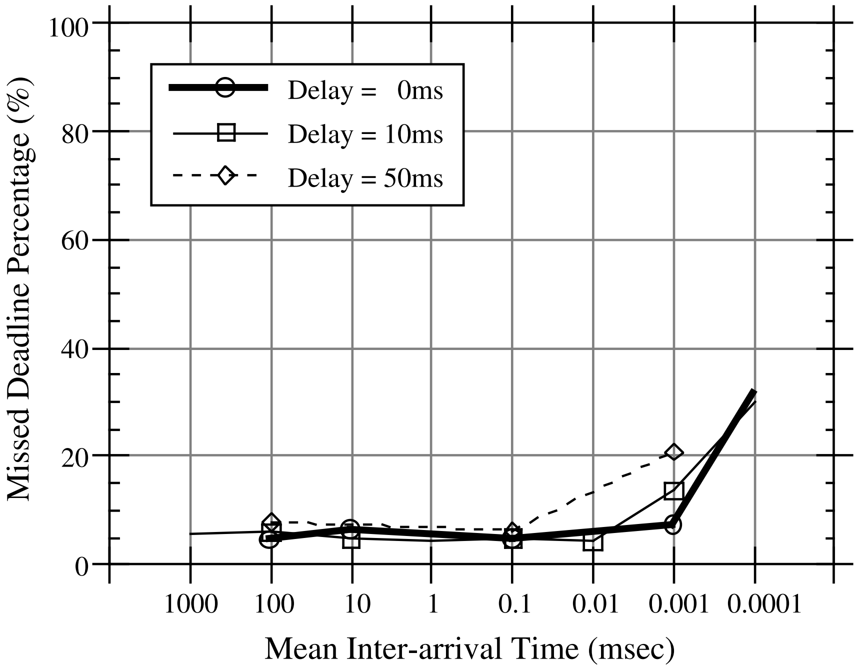

Figure 34: Impact of Communication Delay on HVF (Optimistic)

5.4 Impact of Concurrency Control

As described in Section 4, our evaluation system implements two phase locking (pessimistic) as concurrency control. In this section, we will discuss the impact of concurrency control on the scheduling algorithms.

Figure 35 and 36 show the MDP versus the number of trails for four implementations: EDF with CC (Concurrency Control), EDF without CC, HVF with CC, and HVF without CC. In this section, considering the previous discussions, EDF represents EDF (CSV), and HVF represents HVF (OPT). In this evaluation, statistics were collected after executing 200 operations.

The two graphs correspond to cases where the mean inter-arrival time is 100 msec and 10 msec respectively. We note that MDP for EDF in Figure 36 is smaller than that in Figure 35 even if the the mean inter-arrival time in Figure 36 smaller than that in Figure 35. This is because, as described in Section 5.2, increasing the number of triggered transactions make the system over-loaded in Figure 36.

In a memory based RTDBS, the difference in the number of trails (i.e., data size) is not so serious for the database access speed. However, in our evaluation scenario, the difference in the number of trails affects the number of triggered transaction. For example, reroute operations trigger delete and create operations of trails that are allocated in a certain link. Since the number of trails in a certain link becomes larger when data size becomes larger, the number of triggered transaction also becomes larger. Because of this, increasing the data size over-loads the system.

The two figures show that the impact of concurrency control is relatively small in an overload situation. In such a situation, almost all transactions might miss their deadlines anyway. We note that the effect of concurrency control for EDF is clearer under a low load. With a light load, TG3 and TG4 operations will have a chance to meet their deadlines with an efficient concurrency control algorithms (e.g., optimistic way) under EDF scheduling. For HVF, the impact of concurrency control is small. This is because that the ratio of TG1 and TG2 operations that miss their deadlines under HVF is small relative to the total number of executed transactions in our evaluation environment .

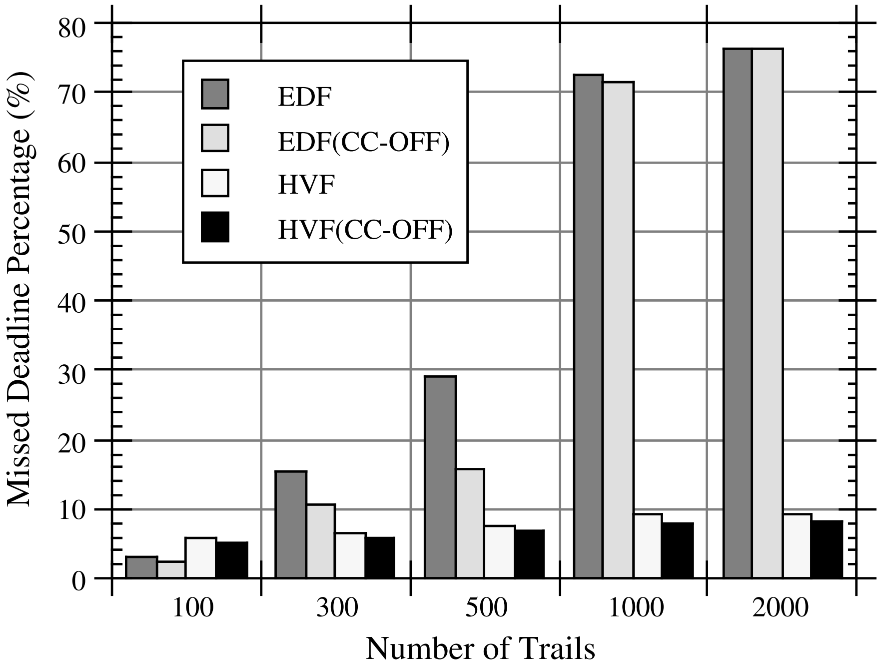

Figure 35: Impact of Concurrency Control when Mean Inter-Arrival time is 100 msec

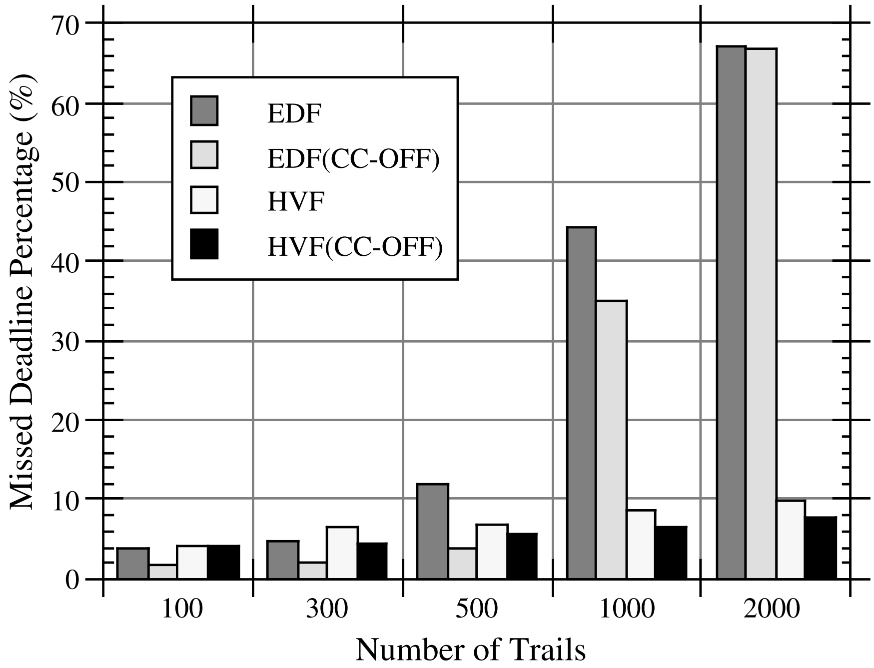

Figure 36: Impact of Concurrency Control when Mean Inter-Arrival time is 10 msec

5.5 Crossover Points

As discussed in the previous sections, the two real-time scheduling algorithm: EDF and HVF, have crossover points where one outperforms the other. In this section, we discuss the crossover points for EDF (CSV) and HVF (OPT) more precisely.

Figures 37 - 39 show the MDP of three scheduling algorithms : FCFS, EDF(CSV), and HVF(OPT). For EDF(CSV) and HVF(OPT), same data shown in Figures 21 - 23 is utilized. The three graphs reflect results after executing 100 operations, 200 operations, and 300 operations respectively. These figures show the same trends seen in Figure 16. As described earlier, both real-time scheduling algorithms outperform FCFS in any case. There are crossover points between EDF (CSV) and HVF (OPT). However, since the performance of HVF (OPT) is better than that of HVF (CSV) depicted in Figure 16, the region where HVF (OPT) outperforms EDF(CSV) becomes wider.

Figures 40 - 43 show the crossover points where HVF (OPT) outperforms EDF (CSV). The four figures correspond to a number of trails of 100, 300, 500, and 1000 respectively. In this evaluation, the statistics are collected after executing 200 operations. As described before, increasing of the number of trails means increasing of the system load. Because of this, the region where HVF (OPT) outperforms EDF (CSV) becomes wide when the number of trails becomes large. Especially in case of Figure 43, EDF (CSV) cannot outperform HVF in any cases. As discussed in Section 5.2, due to the increase in the number of triggered transaction, it is also observed that EDF degrades even if the mean inter-arrival time is still long.

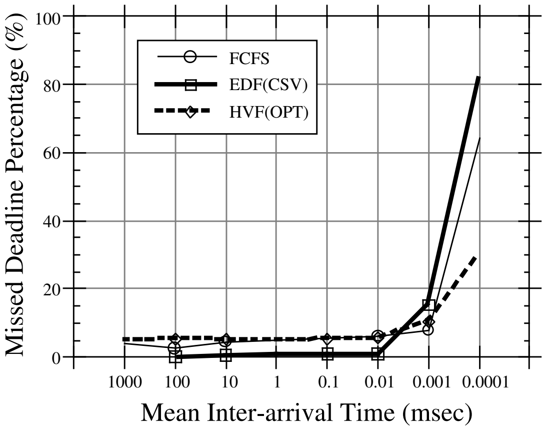

Figure 37: Missed Deadline Percentage after 100 Operations Execution

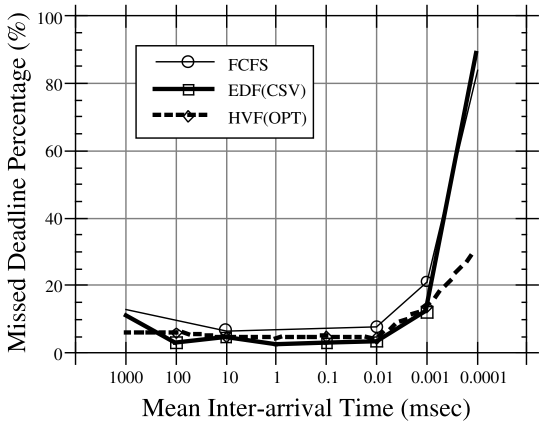

Figure 38: Missed Deadline Percentage after 200 Operations Execution

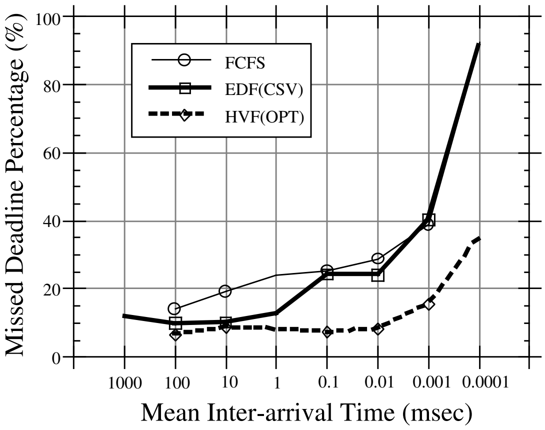

Figure 39: Missed Deadline Percentage after 300 Operations Execution

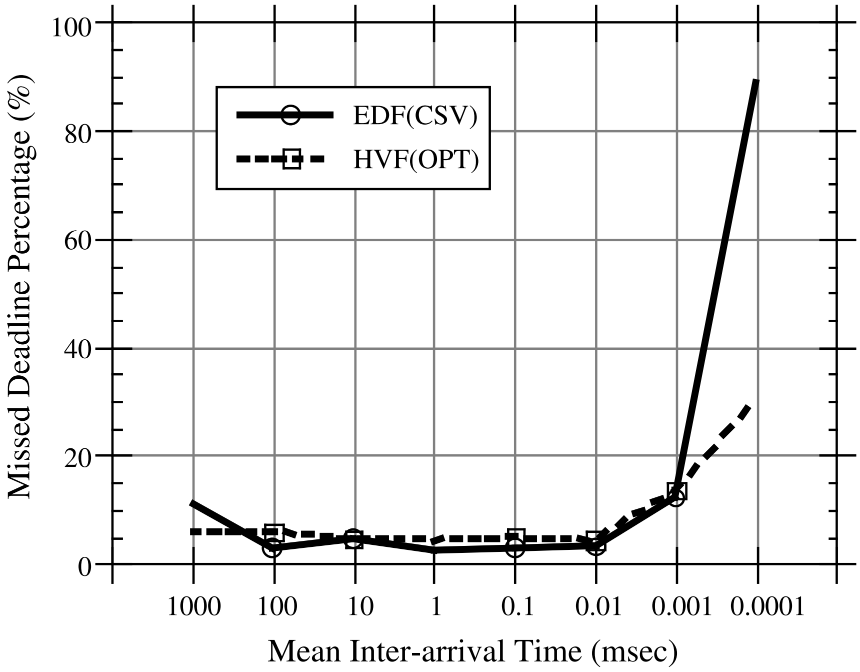

Figure 40: MDP Comparison between EDF and HVF for 100 Trails

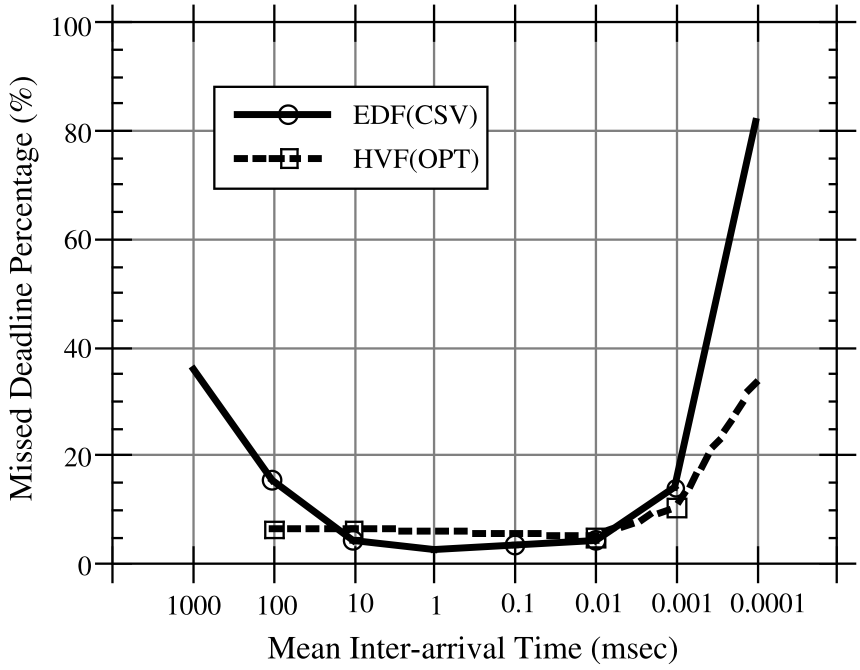

Figure 41: MDP Comparison between EDF and HVF for 300 Trails

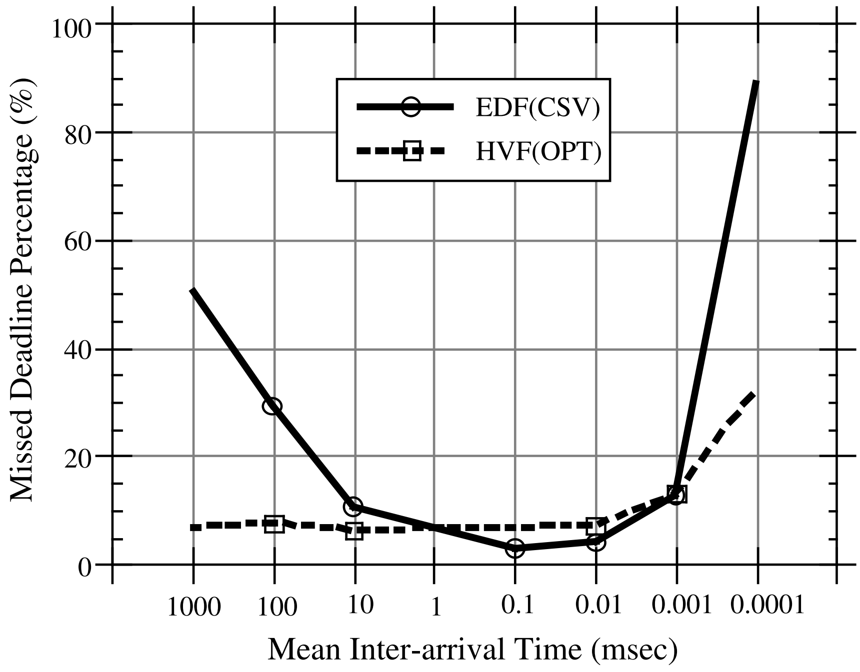

Figure 42: MDP Comparison between EDF and HVF for 500 Trails

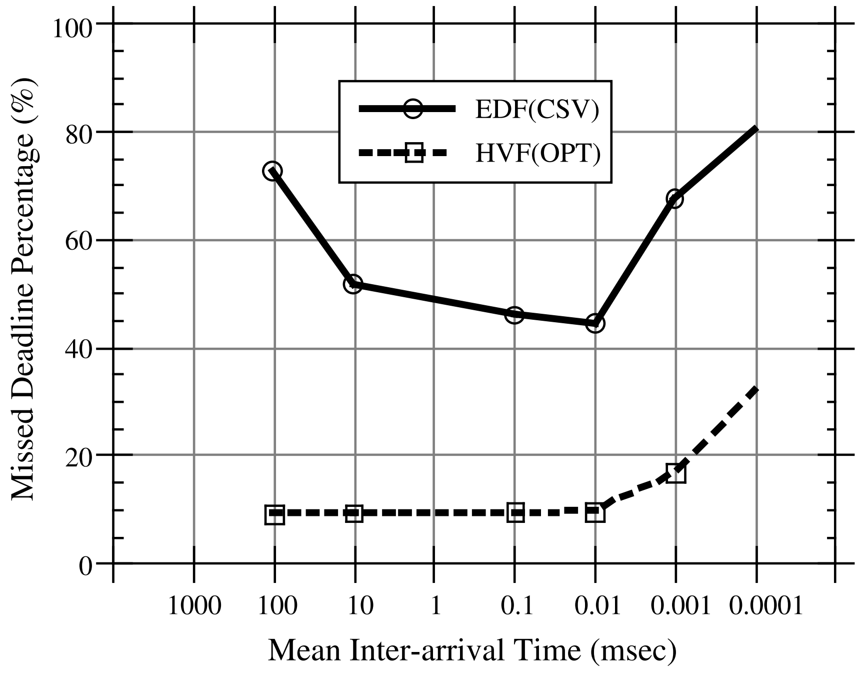

Figure 43: MDP Comparison between EDF and HVF for 1000 Trails

In order to analyze the precise crossover conditions, Figures 44 and 45 show the MQL for four cases of 100, 300, 500, and 1000 trails. The two graphs show EDF (CSV) and HVF (OPT) respectively. The trends for the MQL curves are similar to the trends for MDP curves shown in Figure 40 - 43. As discussed in Section 5.1, we believe that MQL precisely represents the system load. Therefore, we can observe the precise conditions where EDF (OPT) outperforms HVF (OPT) from the two figures:

- If the queue length under EDF (CSV) becomes larger than 20, the system would better off switching the scheduling algorithm from EDF (CSV) to HVF (OPT).
- If the queue length under HVF (OPT) becomes less than 20, the system would better off switching the scheduling algorithm from HVF (OPT) to EDF(CSV).

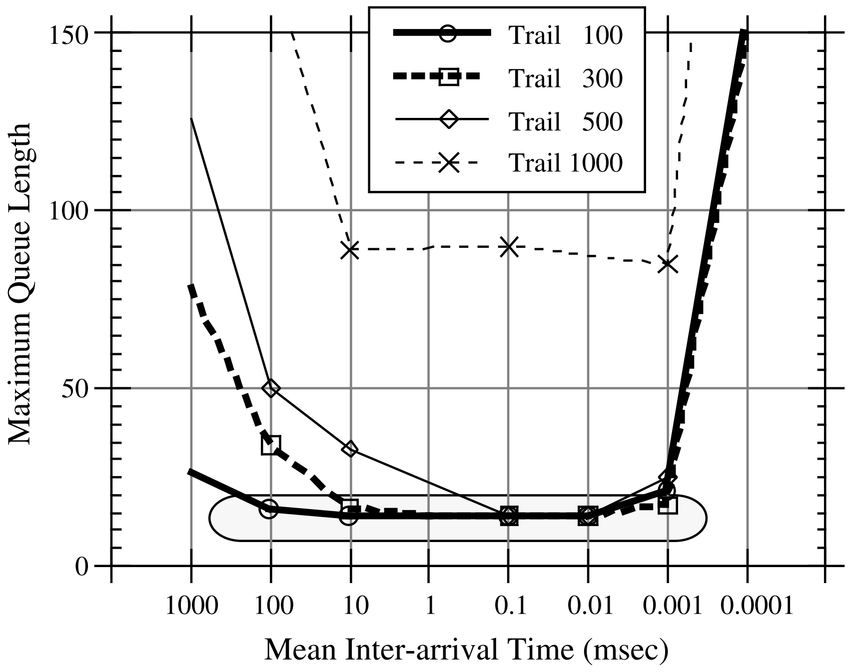

Figure 44: Characteristics of Maximum Queue Length in EDF

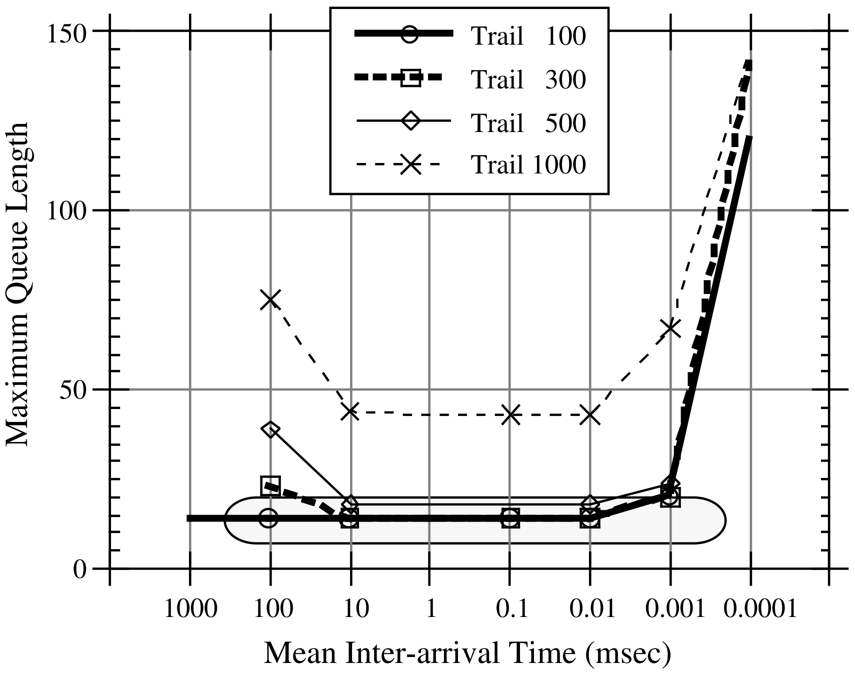

Figure 45: Characteristics of Maximum Queue Length in HVF

6 Conclusion

This report proposed an active and real-time MIB (ART-MIB) architecture for a network management system. The architecture consists of the memory resident database, management tasks, and real-time database kernel. Among these components, the real-time database kernel is the most important for providing real-time capabilities such as transaction scheduling and concurrency control.

We believe that the ART-MIB architecture can provide more flexible data management functions with higher performance. However, as described in Section 2.1, some research issues arise in a network management environment. We note that the policies for transaction scheduling, priority assignment, and concurrency control have a great impact on system performance and timeliness. In order to analyze an actual system and to evaluate its implementation alternatives, we have developed a prototype system that supports real-time database capabilities (i.e., STRIP) and several network management operations for managing an ATM transmission network.

Through our studies we have verified the effectiveness of an RTDB in network management systems, especially in a MIB implementation. The evaluation results showed that real-time scheduling algorithms (both EDF and HVF) outperform non real-time scheduling algorithm (FCFS) in almost all cases. Concerning the real-time scheduling algorithms, we studied their performance characteristics taking into account two types of transaction implementation: conservative and optimistic. In summary, the EDF conservative implementation outperforms the others when the system load is low, and the HVF optimistic implementation outperforms the others in an overload situation. We examined the crossover points where an HVF optimistic implementation starts outperforming an EDF conservative implementation. The crossover points closely depend on the magnitude of the scheduler queue. Furthermore, we showed that EDF conservative was very sensitive to the communication delay and concurrency control. On the other hand, the impact of these is not so significant in HVF optimistic implementation.

As a future work, we plan to develop mechanisms to switch the scheduling algorithm without stopping the running MIB. In order to widely deploy the new MIB utilizing an RTDBS, further study is expected in the following fields.

- Evaluation of the other types of Value Functions (i.e., not a step function)
- Optimistic Concurrency Control Algorithms in the context of Network Management
- End-to-end Deadline Guarantee in the context of Network Management
- Implementation and Evaluation of Recovery Mechanism for Memory based RTDBS
- Implementation and Evaluation of Cooperation Mechanism with Conventional Database Systems
- Implementation and Evaluation of Buffer Management and Disk Scheduling in context of Network Management

Acknowledgments

We wish to thank Hector Garcia-Molina, Ben Kao, and Brad Adelberg of Stanford University, for their technical support and useful discussions. Furthermore, we would like to give special thanks to Masahiro Yamamoto, Satoshi Hasegawa, and Shoichiro Nakai of C&C Research Laboratories, NEC.

References

[Abbott92] R. K. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 17, No. 3, pp. 513-560, 1992.9.

[Adelberg94] B. Adelberg, H. Garcia-Molina and B. Kao, "A Real-Time Database System for Telecommunication Applications," *Proceedings of Second International Conference on Telecommunication Systems*, 1994.3.

[Adelberg95] B. Adelberg, H. Garcia-Molina and B. Kao, "Applying Update Streams in a Soft Real-Time Database System," to be appeared in *Proceedings of ACM SIGMOD95*, 1995.5.

[Bapat91] S. Bapat, "OSI Management Information Base Implementation," *Proceedings of IFIP International Symposium on Integrated Network Management (ISINM) II*, pp. 817-831, 1991.4.

[Garcia92] H. Garcia-Molina and K. Salem, "Main Memory Database Systems: An Overview," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6., pp. 509-516, 1992.12.

[Haritsa90] J. R. Haritsa at el., "Dynamic Real-Time Optimistic Concurrency Control," *Proceedings of IEEE Real-Time System Symposium*, pp. 94-103, 1990.12.

[Huang90] J. Huang, J. Stankovic et al., "Real-Time Transaction Processing: Design, Implementation and Performance Evaluation," *University of Massachusetts COINS TR 90-65*, 1990.6.

[ITU-TMN] ITU-T Recommendation M.3010, "Principles for a Telecommunication Management Network(TMN)," 1991.

[Jensen85] E. D. Jensen, "A Time-Driven Scheduling Model for Realtime Operations System," *Proceedings of IEEE Real-Time System Symposium*, pp. 112-122, 1985.

[Kao93] B. Kao and H. Garcia-Molina, "An Overview of Real-Time Database Systems," *Proceedings of NATO Advanced Study Institute on Real-Time Computing*, Springer-Verlag , 1993.10.

[Kao94] B.Kao and H.Garcia-Molina, "Subtask Deadline Assignment for Complex Distributed Soft Real-Time Tasks," *Proceedings of 14th International Conference on Distributed Computing System*, 1994.6.

[Kiriha91] Y. Kiriha et al., "An Automatic Generation of Management Information Base (MIB) for OSI based Network Management System," *Proceedings of IEEE GLOBECOM 91*, pp. 649-653, 1991.12.

[Kiriha94] Y. Kiriha, "MIB Design and Implementation using Object Oriented Database Technologies," *Proceedings of IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM) 94*, 1994.10.

[Purimetla94] B. Purimetla at el., "Priority Assignment in Real-Time Active Databases," *Proceedings of IEEE Parallel and Distributed Information Systems*, pp. 176-184, 1994.

[Ramamritham93] K. Ramamritham, "Real-Time Databases," *Distributed and Parallel Database I*, Kluwer Academic Publishers, pp. 199-226, 1993.

[Wasson90] J. Wasson at et., "Database Management for an Integrated Network Management System," *Proceedings of Network Management and Control Workshop*, 1990.

[Yemini91] Y. Yemini at el., "Network Management by Delegation," *Proceedings of IFIP 2nd International Symposium on Integrated Network Management*, 1991.4.

Attachment A: Specification of Management Operations

```
/************************************************************/
/*         Network Management Application for STRIP         */
/*                                              - Yoshi  */
/************************************************************/

/* Aperiodic Task */
----------------------------------
do_CreateTrail(bw, route, altpath)
----------------------------------
        "trailid" = increase "lastid";
        add record to "TRAIL" with "trailid" "bw" "route" "altpath";
        ->>> CreateConn(route, bw, trailid, count=0);
         or
        ->>> CreateConnTri(route, bw, trailid, dctp, pointer, count);

----------------------------------------------------
do_DeleteTrail(trailid, flag=0:required by operator)
----------------------------------------------------
        read "bw" "route" record from "TRAIL" identified by "trailid";
        delete record from "TRAIL" identified by "trailid";
        ->>> DeleteConn(trailid, route, bw, flag);
         or
        ->>> DeleteConnTri(trailid, route, bw, flag);

-------------------------------------------------------------
do_TraceTrail(trailid, minabw=MAX:If the result is required )
-------------------------------------------------------------
        read "route" from "TRAIL" indentified by "traidid";
        ->>> ReadConn(trailid, route);
         or
        ->>> ReadConnTri(trailid, route, minabw);

--------------------------
do_BWUpdate(trailid, diff)
--------------------------
        read "route" from "TRAIL" indentified by "traidid";
        update "bw" += "diff" on "TRAIL" identified by "trailid"
        ->>> UpdateConn(trailid, route, diff);
         or
        ->>> UpdateConnTri(trailid, route, diff, pointer);

------------------------------------
do_Reservation(trailid, diff, start)
------------------------------------
        increase "lastid";
```

```
        "end"="start"+1 (if end > 3 then decrease 3);
        "status"="W" (means Wait);
        add the above records on "BWRSV"

----------------------------
do_AlarmReport(plinkid, NE#)
----------------------------
        Add record which identified by plinkid and NE#;
               (contents is always same value (1==QoS inferior)

-----------------------------------
do_Reroute(trailid)
-----------------------------------
        oldtrailid = trailid
        read "bw" "route" "altpath" from "TRAIL" idendentified by "oldtrailid";
        ->>> DeleteTrail(oldplinkid, flag=1:required by do_Reroute);
        /* invoke concurrently */
        /*    another solution is invoked by DeleteConn(Tri) */
        ->>> CreateTrail(bw, altpath, route);


/* Triggered Transactions invoked by task_interface */
/*                  Internal API                    */
-------------------------------
updatettp(trailid, sttp, dttp)
-------------------------------
        update "sttp" "dttp" on record identified "trailid" on "TRAIL";

-------------------------------------------
CompensateCreateTrail(trailid, pointer, bw)
-------------------------------------------
        for (each NE#x in "pointer"){
                read "plinkid" identified by "trailid" on "NE#x";
                delete record identified by "trailid" on "NE#x";
                update "abw" += "bw" identified by "plinkid" on "PLINK"
        }
        delete recored identified by "trailid" on "TRAIL";

------------------------------------------
CompensateBWUpdate(trailid, pointer, diff)
------------------------------------------
        for (each NE#x in "pointer"){
               read "plinkid" identified by "trailid" on "NE#x";
               update "abw"+="diff" on recored identified "plinkid" on "PLINK";
        }
        update "bw" -= diff on record identified "trailid" on "TRAIL";
```

```
------------------------------------
CreateConn(route, bw, trailid, count)
------------------------------------
        dctp = NULL;
        pointer = NULL;
        for (reverse order of each NE#x in "route"){
                "ctpid" = increase "lastid"
                "dctp" = dctp;
                read "plinkid" "abw" from "PLINK" identified by "NES#";"
                if ("abw" > "bw") then {
                        add record on "NE#x" with "ctpid" "dctp" ... "plinkid";
                        update "abw"-=bw on "PLINK" identified by "plinkid";
                        pointer <- lisp_car("route");
                        "route" = list_cdr("route");
                        count ++;
                        "dctp" = ctpid;
                        if ("route" == NULL) then
                                "sttp"= ctpid (only last loop);
                        if (count == 1) then
                                "dttp" = ctpid (only first loop);
                } else {
                        /* CreateConn Fail */
                        ->>> CompensateCreateTrail(trailid, pointer, bw);
                }
        }
        ->>> updatettp(trailid, sttp, dttp);

--------------------------------------------------------
CreateConnTri(route, bw, trailtid, dctp, count, pointer)
--------------------------------------------------------
        "ctpid" = increase "lastid"
        "dctp" = dctp;
        read "plinkid" "abw" from "PLINK" identified by "NES#";"
        if ("abw" > "bw") then {
                add record on "NE#x" with "ctpid" "dctp" ... "plinkid";
                update "abw"-=bw on "PLINK" identified by "plinkid";
                pointer <- lisp_car("route");
                "route" = list_cdr("route");
                count ++;
                "dctp" = ctpid;
                if (count == 1) then
                        "dttp" = ctpid (only first loop);
                if ("route" == NULL) then {
                        "sttp"= ctpid (only last loop);
                        ->>> updatettp(trailid, sttp, dttp);
```

```
                }
        } else {
                /* CreateConn Fail */
                ->>> CompensateCreateTrail(trailid, pointer, bw);
        }
        /* Triggerd Transaction */
        ->>> CreateConnTri(route, bw, trailid, dctp, count, pointer);

------------------------------------
DeleteConn(trailid, route, bw, flag)
------------------------------------
        for (each NE#x on "route"){
                read "plinkid" on "NE#x" identified by "trailid";
                delete record identified by "trailid";
                update "abw"+="bw" on "PLINK" identified "plinkid";
                if flag == 1 (required by Reroute) then {
                        update "pfm" = 1.0 on "PLINK" identified "plinkid";
                }
        }

---------------------------------------
DeleteConnTri(trailid, route, bw, flag)
---------------------------------------
        read "plinkid" on "NE#x" identified by "trailid";
        delete record identified by "trailid" on "NE#x";
        update "abw"+="bw" on "PLINK" identified "plinkid";
        if flag == 1 (required by Reroute) then {
                update "pfm" = 1.0 on "PLINK" identified "plinkid";
        }
        if ("route"!=NULL) then
                ->>> DeleteConnTri(trailid, route, bw, flag);

------------------------
ReadConn(trailid, route)
------------------------
        minabw=MAX;
        for (each NE#x on "route"){
                read "plinkid" from "NE#x" identified by "trailid";
                read "abw" from "PLINK" identified by "plinkid";
                if minabw > "abw" then minabw="abw";
        }

-----------------------------------
ReadConnTri(trailid, route, minabw)
-----------------------------------
        #x = lisp-car(route);
```

```
        "route" = lisp-cdr(route);
        read "plinkid" from "NE#x" identified by "trailid";
        read "abw" from "PLINK" identified by "plinkid";
        if minabw > "abw" then minabw="abw";
        if ("route"!=NULL) then
                ->>> ReadConnTri(trailid, route, abw)

--------------------------------
UpdateConn(trailid, route, diff)
--------------------------------
        for (each NE#x on "route"){
                read "plinkid" from "NE#x" identified by "trailid";
                read "abw" from "PLINK" identified by "plinkid";
                if ("abw" > "diff") then {
                        update "abw"-=diff on "PLINK" identified by "plinkid";
                } else {
                        ->>> CompensateBWUpdate(trailid, pointer, diff)
                }
                pointer <- "NE#x";
        }

--------------------------------------------
UpdateConnTri(trailid, route, diff, pointer)
--------------------------------------------
        read "plinkid" from "NE#x" identified by "trailid";
        read "abw" from "PLINK" identified by "plinkid";
        if ("abw" > "bw") then {
                update "abw"-=diff on "PLINK" identified by "plinkid";
        } else {
                ->>> CompensateBWUpdate(trailid, pointer, diff)
        }
        pointer <- "NE#x";
        if ("route"!=NULL) then
                ->>> UpdateConnTri(trailid, route, diff, pointer)


/* Periodic Task */
-----------------------
do_ExecReservation(time)
-----------------------
        read all records "BWRSV";
        store records which "start" or "end" == "localtime";
        for (each record identified by "trailid"){
                if "status" == "s" then
                        delete the recod from "RSV";
                        "diff"=-1*"diff";
```

```
                update "status" on "RSV";
                /* triggerd trasaction */
                ->>> do_BWUpdate (trailid, diff);
        }

---------------------
do_QosReport(plinkid)
---------------------
        read "NE#" value on the record identified by "plinkid";
        update "pfm" value on the same recored (means monitoring of QoS );
        if "pfm" < "Threshold" then (means QoS inferior of the Line){
                /* triggerd trasaction */
                ->>> do_AlarmReport(plinkid, NE#);
        }

--------------
do_ScanAlarm()
--------------
        read all records from "ALARM" and count alarms on each plinkid
        if "alarmcount(plinkid)" > "MAXALARM" then (means failure prediction){
                read "trailid" form "NE#x" identified by "NE#" in "ALARM"
                /* triggerd trasaction */
                ->>> do_Reroute(trailid);
        }
```