PPBRI

PAN-PACIFIC JOURNAL OF BUSINESS RESEARCH

Volume 5, No. 1 Spring, 2014

Table of Contents

1. Topics: All areas of business, economics, and information systems

2. Manuscript Guidelines/Comments:
Pan-Pacific Journal of Business Research (PPJBR) is a double blind peer reviewed Journal focusing on integrating all areas of business, economics, finance, and Information Systems. PPJBR pursues high quality researches significantly contributing to the theories and practices of all areas of business, economics, and Information Systems. PPJBR is an academic journal listed on Cabell Directory. PPJBR consider for publication the following topics in all areas of business and economics including Accounting, Economics, Entrepreneurship, Finance, Hospitality Management, International Business, Marketing, Human Resource Management, Operation Management, Information Systems, Strategy, and Supply Chain Management:

- Current and new theories.
- New regulations and policies.
- Application of business and economic theories.
- Case studies exploring current issues
- Pedagogical issues in business education

3. Submission:
Authors are required to submit their article or manuscript electronically at info@ppbri.org. Before submission, the article or manuscript should not be published in any other journal. The article or manuscript should be in MS Office Word format. It should be written in a single space with a maximum number of 15 pages and 12 font size. Title, the name(s), affiliation(s), address (es), phone number(s), and email(s) of authors should be on the cover page. Contact author should be indicated. Only an abstract of the article or manuscript in 250 words, title, and 4 key words should be shown on the second page.

PPJBR generally follows the American Psychological Association (APA) guidelines. Reference should be presented in a separate sheet at the end of the article or manuscript. Tables, figures, footnotes, and their numbering should appear on the appropriate page. The usage of footnotes should be minimized. The decision of acceptance usually takes three months. After acceptance, PPBRI has a copy right for the accepted article and manuscript.

The article or manuscript should be submitted to: Dr. Kyung Joo Lee, Editor, Kiah Hall Suite 2110, Princess Anne, MD 21853. Phone: 410-621-8738. Email: kjlee@umes.edu.

Security/Performance Tradeoffs in Hybrid Real-Time Scheduling Algorithms

Joon Son, California State University, San Bernardino*
Jim Alves-Foss, University of Idaho

ABSTRACT

Typical real-time systems handle a hybrid task set consisting of periodic and aperiodic tasks. This paper addresses the covert timing channel issues in scheduling a set of hybrid tasks for Multi-Level Secure (MLS) real-time systems. After identifying timing vulnerabilities in several existing hybrid scheduling algorithms, we propose security measures for eliminating covert timing channels. Usually, security measures applied to satisfy the security requirements adversely affect system performance or timeliness requirements of real-time systems. Since the timeliness requirements of many real-time systems cannot be compromised over security requirements, the tradeoffs between the security and timeliness requirements in hybrid scheduling algorithms are also discussed.

Keywords: Multi-Level Security (MLS), covert timing channel, security-performance tradeoff

JEL Classification: A1

*Corresponding author: Information Decision Sciences, College of Business and Public Administration, JB-551, 5500 University Parkway, San Bernardino, CA 92407-2397 Office: (909) 537-5778, Fax (909) 537 7176, email:json@csusb.edu.

Date Availability: Data are available from sources identified in the paper.

1 Introduction

Multi-Level Security (MLS) is concerned with controlling the flow of information in systems. The main goal of MLS policies is to ensure that information at a high security classification cannot flow down to a lower security classification. A typical approach to implement a Multi-Level Secure (MLS) system is to assign classification labels to all objects and clearance labels to all subjects. To determine whether a specific access mode is allowed, the clearance of a subject is compared to the classification of the object to determine if the subject is authorized for the specific access mode. This is the approach taken in the well-known Bell-La Padula access control model [5]. For simplicity, throughout this paper, we assume that a MLS system has two security levels, $High$-secrecy and Low-secrecy.

It has been known that access control models are not sufficient to ensure that users cannot obtain information for which they do not have the necessary clearance. This is the due to an existence of covert channels through which information can be transmitted from a $High$ level entity ($High$) to a Low level entity (Low). In this paper, we are interested in a specific type of covert channel, a timing channel. A covert timing channel exists if it is possible for $High$ to interfere with its use of system resources in such a way that this manipulation affects the response time observed by Low.

Numerous papers [7, 19, 20, 22, 29, 35, 38] have presented mathematical frameworks for analyzing the possible information leakage through a covert timing channel for various systems. In our previous works [26, 27], we analyzed how a covert timing channel can be created and exploited by a set of independent periodic tasks having different secrecy levels and running under a well-known fixed-priority preemptive scheduling algorithm. In addition, we developed a a general trace-based framework to carry out a covert timing channel analysis of a real-time system. [28]. Our focus was primarily on the security aspects of a real-time system. Since the timeliness requirements (e.g., deadlines) or performance overhead of real-time systems cannot be compromised over security requirements, in this paper, we investigate how a security measure taken to eliminate a covert timing channel can cause performance overhead. In particular, we quantify a relationship between performance overhead and covert channel capacity as the security measure is applied.

Many real-time systems require an integrated approach suitable for scheduling hard deadline periodic tasks along with aperiodic tasks with no firm deadline. In this paper, we carry out a covert timing channel analysis of real-time systems which employ fixed-priority preemptive scheduling algorithms for scheduling a set of hybrid (periodic and aperiodic) tasks. Typically, periodic tasks are time-driven and execute critical control activities with hard timing constraints. Aperiodic tasks are usually event-driven and may have soft, or non-real-time requirements depending upon the specific application. In fixed-priority-based scheduling, control of the CPU is always given to the highest priority task ready to run and the scheduling priority assigned a task is fixed. How a scheduling priority is assigned to a task, however, is determined by the type of scheduling algorithms used. Among a group of hybrid fixed-priority scheduling algorithms, we choose the three most well-known algorithms for our work: Polling Server (PS), Deferrable Server (DS), and Priority Exchange (PE) [4, 16, 18, 32]. The detailed description of these algorithms are presented in Section 2.

The objectives of this paper are twofold. The first is to identify the possible existence of covert timing channels in each scheduling algorithm and propose a security measure to eliminate them. The second is to present a mathematical model of controlling the amount of information leakage which causes performance overhead.

2 Real-time scheduling algorithms

In this section, we present a number of scheduling algorithms for handling a hybrid task set consisting of hard periodic tasks and soft aperiodic tasks. The deadline of a hard task must be met (if the deadline is missed, the system is in fault) while the deadline of a soft task does not need to be always met. The simplest method of dealing with a set of soft aperiodic tasks is to schedule them only when there are no periodic instances ready to execute. This simple scheduling method is called Background Scheduling (BS). Obviously, the major problem with BS is that the average response time of aperiodic tasks can be too long when periodic task loads are high. We discuss three basic fixed preemptive scheduling algorithms devised to improve the average response time of aperiodic tasks, compared to BS. Before explaining the hybrid scheduling algorithms, we introduce some of the mathematical notations and assumptions used throughout this paper.

2.1 Notations and assumptions

The notations used in this paper are:

- A periodic (or an aperiodic) task with label i is represented by $\mathbf{T}_i^{\mathcal{P}}$ (or $\mathbf{T}_i^{\mathcal{A}}$). A task i with scheduling priority π_i is denoted by $\mathbf{T}_{i,\pi_i}$. Note that $\pi_i \in \{1, 2, \ldots, n\}$ (1 being the highest scheduling priority and n being the lowest). We denote a periodic (or an aperiodic) task i with scheduling priority π_i by $\mathbf{T}_{i,\pi_i}^{\mathcal{P}}$ (or $\mathbf{T}_{i,\pi_i}^{\mathcal{A}}$).

- An instance of a task is called a job. For instance, a periodic task consists of a series of jobs with regular arrival times.

- The response time r_i of a task i is the time difference between the release time of a job of the task i and the completion of the job. We denote the worst case response time of a periodic task i by R_i.

- A periodic task i is characterized by the following four parameters: the phase Φ_i, the relative deadline D_i, the period T_i and the worst (maximum) computation time C_i. The phase Φ_i is the release time of the first job of $\mathbf{T}_i^{\mathcal{P}}$. The relative deadline D_i is the maximum allowable response time of $\mathbf{T}_i^{\mathcal{P}}$. We denote a periodic task with the four parameters by $\mathbf{T}_i^{\mathcal{P}(\Phi_i, D_i, T_i, C_i)}$.

The scheduling algorithms presented in this section are based upon the following assumptions:

- The tasks running on a single processor are *independent* (no shared resources among the tasks other than the processor).
- The Periodic tasks are scheduled by the Rate-Monotonic (RM) scheduling algorithm [15, 16, 17]. The RM scheduling algorithm has the following characteristics: tasks with shorter periods (higher request rates) will have higher scheduling priorities, a scheduling priority assigned to a task is fixed and a currently executing task is preempted by a newly arrived task with a higher priority (shorter period).
- Unless otherwise specified, all periodic tasks start simultaneously at time 0 ($\Phi = 0$) and their relative deadlines are equal to their periods ($D = T$). Thus, the notation of a periodic task i can be simplified as $\mathbf{T}_i{}^{\mathcal{P}(T_i,C_i)}$. For a periodic task i to be schedulable, $R_i \leq D_i$.
- Arrival times of aperiodic tasks are unknown.

2.2 Polling Server

The Polling Server (PS) algorithm is based upon the following approach [18]. When an aperiodic task arrives, it is placed in an aperiodic job queue, waiting for execution. A periodic task, called the polling server, is created to serve aperiodic requests. Like any periodic task, the polling server s is characterized by the polling period T_s and the maximum computation time C_s, i.e. $\mathbf{T}_s^{\mathcal{P}(T_s,C_s)}$. In the PS algorithm, the parameter Cap, called the server capacity, is monitored to make sure that the polling server cannot execute an aperiodic task for more than C_s units of time. At the beginning of each server period T_s, Cap is set to C_s (we say that the server capacity Cap is replenished by C_s units of time) and the aperiodic job queue is examined for emptiness:

- If the queue is found to be not empty, the polling server executes the aperiodic job(s) until there is no job left to execute in the queue or it executes for C_s units of time, whichever occurs sooner. When the server executes the aperiodic jobs in the queue, it consumes its Cap at the rate of one per unit time. When the queue becomes empty or the polling server has consumed all of its server capacity Cap (i.e., Cap becomes zero - we say that the server capacity becomes exhausted), it is immediately suspended and wait for the next polling period for execution.
- If the queue is found empty, the polling server suspends immediately. The polling server will not be ready for execution and is not able to examine the queue again until the next polling period. An aperiodic task which arrives after the aperiodic job queue is examined and found empty must wait for the next polling period to be serviced.

In general, the polling server is scheduled with the same algorithm used for the periodic task, and, once active, it serves the aperiodic requests with the limit of its server capacity. Figure 1 illustrates an example where the aperiodic task j is serviced by the polling server s. Note that numbers above the arrows in Figure 1(a) indicate the computation time associated with the aperiodic requests. The example shows that the set Γ_{PS} of the polling server s with $T_s = 3$ and $C_s = 2$ and the periodic task i is scheduled by the RM algorithm. According to the RM scheduling rule, $\pi_s = 1$ and $\pi_i = 2$ since $T_s < T_i$. In the example, at time 0 ($t = 0$), the server suspends itself since there is no aperiodic request pending and the periodic task executes instead. The first aperiodic task, which arrives at time 5, cannot execute immediately since the polling server suspends itself at time 3; it must wait until the beginning of the next server period ($t = 6$). At time 6, the capacity of the server is replenished to its full value ($C_s = 2$) and is used to serve the aperiodic task; the periodic task activated at time 6 is preempted by the aperiodic task. At time 7, the aperiodic task finishes its execution and the server suspends itself; the periodic task which is preempted at time 6 is able to execute. We skip an explanation of the rest of the timing behaviors of the tasks.

We introduce a new notation to denote a real-time system Γ_{Alg} with the following conditions:

- The real-time system Γ_{Alg} consists of an aperiodic task j and a set of periodic tasks ($\mathbf{T}^{\mathcal{P}(T_{i_1},C_{i_1})}_{i_1,\pi_{i_1}}$, $\mathbf{T}^{\mathcal{P}(T_{i_2},C_{i_2})}_{i_2,\pi_{i_2}}$,...), scheduled by a hybrid scheduling algorithm $Alg \in \{PS, DS, PE\}$. The system is denoted by:

$$\Gamma_{Alg} = \{(\mathbf{T}^{\mathcal{A}}_{j,\pi_s}, \mathbf{T}^{\mathcal{P}(T_s,C_s)}_{s,\pi_s}), \mathbf{T}^{\mathcal{P}(T_{i_1},C_{i_1})}_{i_1,\pi_{i_1}}, \mathbf{T}^{\mathcal{P}(T_{i_2},C_{i_2})}_{i_2,\pi_{i_2}} \dots\}_{RM}$$

  In the above notation, $(\mathbf{T}^{\mathcal{A}}_{j,\pi_s}, \mathbf{T}^{\mathcal{P}(T_s,C_s)}_{s,\pi_s})$ denotes that the aperiodic task j is handled by the periodic server s and scheduling priority π_j of the aperiodic task j follows that of the server s, i.e. $\pi_j = \pi_s$.

- The periodic server s and periodic tasks are scheduled by the *RM* scheduling algorithm (i.e., the periods, T_s, T_{i_1}, T_{i_2}, ... are compared to determine a scheduling priority of each task). Note that both *RM* and hybrid scheduling (*PS*, *DS*, and *PE*) impact the timing behaviors of the aperiodic task j.

Using the above notation, the PS algorithm of Figure 1 can be formalized as (the aperiodic task j has a higher scheduling priority than the periodic task i since $T_s = 3$, $T_i = 6$, and $T_s < T_i$) :

$$\Gamma_{PS} = \{(\mathbf{T}^{\mathcal{A}}_{j,1}, \mathbf{T}^{\mathcal{P}(3,2)}_{s,1}), \mathbf{T}^{\mathcal{P}(6,1)}_{i,2}\}_{RM}$$

In the following two sub-sections, we introduce the Deferrable and the Priority Exchange algorithms proposed to improve an average response time of aperiodic tasks with respect to the PS algorithm.

2.3 Deferrable Server

Just as polling, the Deferrable Server (DS) algorithm creates a periodic task called a deferrable server s to service aperiodic requests [4, 16, 18, 32]. A capacity *Cap* of a deferrable server is also replenished periodically with period T_s. However, unlike polling,

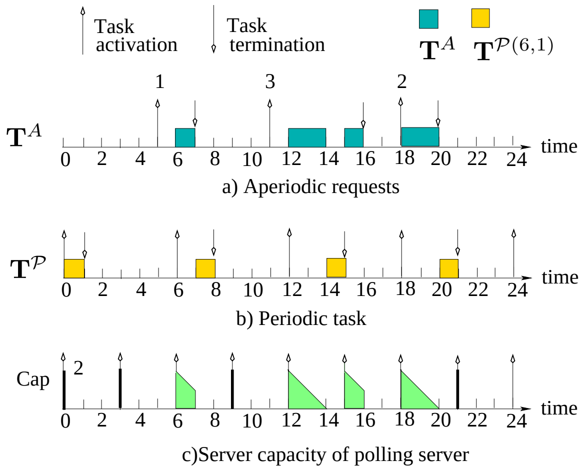

Figure 1: Example: PS scheduling

when a deferable server finds no aperiodic job ready for execution, it preserves its server capacity, rather than suspends itself. We use an example to explain the DS algorithm. Let us assume that a system Γ_{DS} of hybrid tasks has the following scheduling parameters:

$$\Gamma_{DS} = \{(\mathbf{T}^{\mathcal{A}}_{j,1}, \mathbf{T}^{\mathcal{P}(3,2)}_{s,1}), \mathbf{T}^{\mathcal{P}(6,1)}_{i,2}\}_{RM}$$

A possible sequence of executions of the hybrid tasks in the system Γ_{DS} is shown in Figure 2. There are a few points worth noting in the timing diagram: at time 0 ($t = 0$), the server is given 2 units of capacity ($Cap = C_s = 2$). The capacity Cap *remains* at 2 until time 5 since there is no aperiodic request. At time 5, the aperiodic task j arrives and the deferrable server s executes the task j; the capacity of the server decreases as it executes. At time 11, the aperiodic task arrives and is immediately executed by the server; Cap is decreased by 1. At time 12, Cap of the server is replenished to its full value ($Cap = 2$) and the server continues to execute the aperiodic task j. The execution of the periodic task which arrives at $t = 12$ is delayed. At time 14, the server finishes the execution and Cap is exhausted; the periodic task begins to execute. We skip an explanation of the rest of the timing behaviors.

2.4 Priority Exchange

Just as PS and DS, the Priority Exchange (PE) algorithm creates a periodic server s for servicing aperiodic tasks. At the beginning of each server period, the capacity Cap is replenished to its full value ($Cap = C_s$). If aperiodic requests are pending and the higher-priority server is ready, the aperiodic tasks are serviced using the available server capacity.

However, unlike PS and DS, PE preserves its capacity C_s by exchanging it for the execution time of a lower-priority periodic task when no aperiodic requests are pending to use the capacity. When a priority exchange occurs between a periodic task and a PE server, the periodic task executes at the priority level of the server while the server

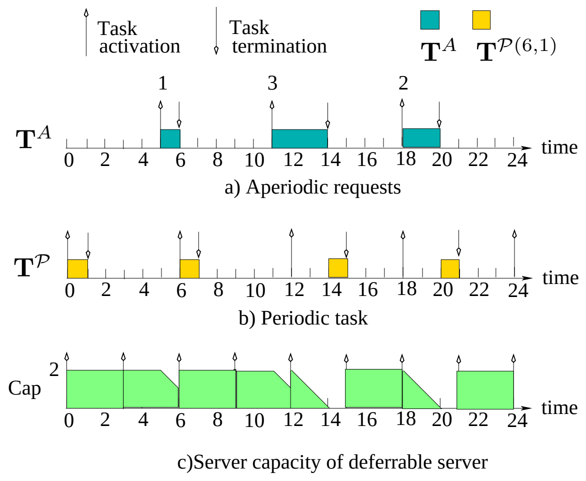

Figure 2: Example: DS algorithm

accumulates a capacity at the priority level of the periodic task. Thus, the periodic task advances its execution, and the server capacity is not lost but preserved at lower priority. If no aperiodic task arrives to use the capacity, priority exchange continues with other lower-priority tasks until either the capacity exhausted or no task is active. When no task is active, server capacity preserved at lower priority is gradually discarded. An example of the PE algorithm shown in Figure 3 assumes:

$$\Gamma_{PE} = \{(\mathbf{T}^{\mathcal{A}}_{j,1}, \mathbf{T}^{\mathcal{P}(4,1)}_{s,1}), \mathbf{T}^{\mathcal{P}(8,4)}_{i,2}\}_{RM}$$

At time 0, the capacity Cap of the server s is replenished to its full capacity ($Cap = C_s = 1$), but no aperiodic requests are pending; Cap is *exchanged* with the execution time of the periodic task i. As a result, the periodic task i advances its execution and the server *accumulates* one unit of time at the priority level of the task i. Note that the line shown in Figure 3 (b) overlapped with the schedule of the periodic task indicates the capacity Cap accumulated at the priority level of the corresponding periodic task. Also note that the line shown in Figure 3(c) overlapped with the sever capacity represents the total units of time available for executing aperiodic requests. At time 2, the periodic task finishes its execution. Cap, which is preserved at the priority level of the periodic task, diminishes since no task is active. At time 4, Cap is replenished ($Cap = 1$) and used to execute the aperiodic task j for 1 unit of time. At time 8, the newly replenished Cap is used to execute the remaining portion of the previous aperiodic task j . At time 16, C_s is exchanged (preserved at the priority level of the task i) and the periodic task advances to execute. At time 20, the newly replenished Cap (1 unit of time) is fully consumed to serve the second aperiodic request which requires 2 units of execution time. At time 21, the capacity accumulated at the priority level of the periodic task i is consumed to execute the remaining portion (one unit of time) of the second aperiodic request; this execution of the aperiodic request is what makes PE different from DS and PS. If DS or PS is used instead, the execution occurred at $t = 21$ is delayed until the beginning ($t = 24$) of the next server period.

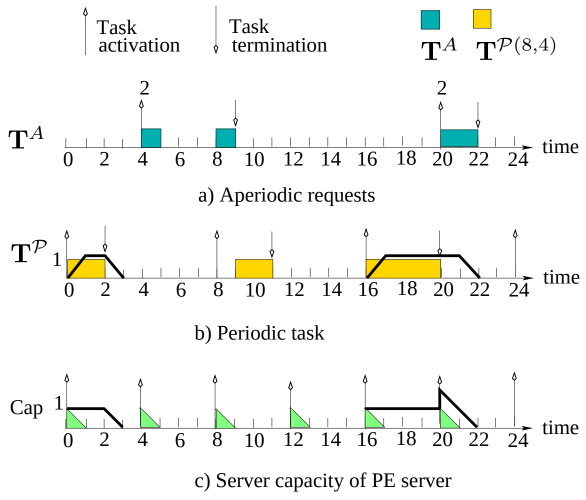

Figure 3: Example: PE algorithm

Since the PE algorithm has to track priority exchanges with lower priority tasks while DS algorithm only needs to monitor its capacity at the original priority level, PE is harder to implement. However, a PE server has a better CPU utilization and response time for aperiodic requests than DS [6].

3 Covert timing channel analysis

3.1 Our model & assumptions

In MLS real-time systems, it is assumed that both the security level and the scheduling priority of each task are defined. The label i of task $\mathbf{T}_{i,\pi_i}$ is used to indicate the security level, i.e. $i \in \{H, L\}$ (H for *High*-secrecy and L for *Low*-secrecy). For example, $\mathbf{T}_{H,2}$ is a *High*-secrecy task with the scheduling priority equal to 2 ($\pi_H = 2$).

To simplify our analysis, we only consider the cases where there is no third party task $\mathbf{T}_N$ running with a scheduling priority higher than any of a *High*-secrecy or a *Low*-secrecy task; such a third party task may affect the timing behaviors of *High*-secrecy or *Low*-secrecy tasks, thereby introducing noise into covert communication channels. With two parameters (security level and scheduling priority) associated with each task and the noiseless channel assumption, there are four different cases to consider. Let $Alg \in \{$ PS, DS, PE $\}$ denote the scheduling algorithm being used. The term such as higher- or lower-priority indicates a *scheduling* priority level.

- Case I - a real-time system Γ_{Alg} consisting of the *Low*-secrecy higher-priority aperiodic task, the *High*-secrecy lower-priority periodic task and the third party tasks is scheduled by the algorithm Alg:

$$\Gamma_{Alg} = \{(\mathbf{T}^{\mathcal{A}}_{L,1}, \mathbf{T}^{\mathcal{P}(T_S,C_S)}_{S,1}), \mathbf{T}^{\mathcal{P}(T_H,C_H)}_{H,2}, \mathbf{T}_{N,3}, \ldots\}_{RM}$$

	Case I	Case II	Case III	Case IV
PS	Secure	Insecure	Insecure	Secure
DS	Secure	Insecure	Insecure	Secure
PE	**Insecure**	Insecure	Insecure	Secure

Table 1: Existence of covert flows

- Case II - a real-time system Γ_{Alg} consisting of the $High$-secrecy higher-priority aperiodic task, the Low-secrecy lower-priority periodic task and the third party tasks is scheduled by the algorithm Alg:

$$\Gamma_{Alg} = \{(\mathbf{T}^{\mathcal{A}}_{H,1}, \mathbf{T}^{\mathcal{P}(T_S,C_S)}_{S,1}), \mathbf{T}^{\mathcal{P}(T_L,C_L)}_{L,2}, \mathbf{T}_{N,3}, \ldots\}_{RM}$$

- Case III - a real-time system Γ_{Alg} consisting of the Low-secrecy lower-priority aperiodic task, the $High$-secrecy higher-priority periodic task and the third party tasks is scheduled by the algorithm Alg:

$$\Gamma_{Alg}\{(\mathbf{T}^{\mathcal{A}}_{L,2}, \mathbf{T}^{\mathcal{P}(T_S,C_S)}_{S,2}), \mathbf{T}^{\mathcal{P}(T_H,C_H)}_{H,1}, \mathbf{T}_{N,3}, \ldots\}_{RM}$$

- Case IV - a real-time system Γ_{Alg} consisting of the $High$-secrecy lower-priority aperiodic task, the Low-secrecy higher-priority periodic task and the third party tasks is scheduled by the algorithm Alg:

$$\Gamma_{Alg} = \{(\mathbf{T}^{\mathcal{A}}_{H,2}, \mathbf{T}^{\mathcal{P}(T_S,C_S)}_{S,2}), \mathbf{T}^{\mathcal{P}(T_L,C_L)}_{L,1}, \mathbf{T}_{N,3}, \ldots\}_{RM}$$

3.2 Identification of covert timing channels

In fixed-preemptive real-time scheduling, information leaks through covert timing channels if $High$-secrecy tasks can preempt Low-secrecy tasks, thereby affecting response times of Low-secrecy tasks. In Case II and III (see Table 1), covert timing channels can be easily constructed since the $High$-secrecy task is able to preempt the Low-secrecy task.

A common approach to eliminate a covert timing channel is to assign a higher scheduling priority to a Low-secrecy task. This approach is called Lower-Secrecy First (LSF). Although lower-Secrecy First (LSF) [31] is one of the commonly adopted methods to eliminate covert timing channels, it has well known performance disadvantages: $High$-secrecy tasks experience a delay in response time under the LSF rule.

As expected, there is no covert flow in Case I-DS, I-PS, and IV (Table 1) since the real-time tasks are assigned scheduling priorities based upon the LSF rule (see Table 1). However, there is an interesting exception to the LSF rule in Case I-PE (where the real-time tasks run under PE scheduling and are assigned scheduling priorities based upon the LSF rule). In the next section, we demonstrate how $High$ can leak information to Low in Case I-PE.

3.3 Exception to Lower-Secrecy First (LSF)

Even though the LSF rule is applied, the $High$-secrecy task can still create a covert channel if PE scheduling is deployed. More specifically, under PE scheduling, the $High$-secrecy lower-priority periodic task can affect the response time of the Low-secrecy higher-priority aperiodic task, thereby allowing covert timing flow from $High$ to Low. The task execution diagram in Figure 3 is used as an example to illustrate how covert timing channels can be established in Case I-PE. In the example, the Low-secrecy higher-priority aperiodic task handled by the PE server s with $T_s = 4$ and $C_s = 1$ and the $High$-secrecy lower-priority periodic task with $T_{_H} = 8$ and $C_{_H} = 4$ are scheduled under the RM algorithm:

$$\Gamma_{_{PE}} = \{(\mathbf{T}^{\mathcal{A}}_{L,1}, \mathbf{T}^{\mathcal{P}(4,1)}_{s,1}), \mathbf{T}^{\mathcal{P}(8,4)}_{H,2}\}_{_{RM}}$$

$High$, as an information sender, and Low, as an information receiver, have the following strategy: the server capacity preserved in the $High$-secrecy lower scheduling periodic task is used as a shared communication medium between $High$ and Low to covertly transmit information. With the long execution of the $High$-secrecy task, the server capacity can be preserved long enough to be used for the Low-secrecy aperiodic request, which causes a short response time of the Low-secrecy task (Low interprets this as receiving 0 from $High$). With the short execution, the preserved server capacity quickly diminishes and the Low-secrecy task experiences a delay in response time (Low translates this as receiving 1). In the example, at $t = 0$, 16, ... ($t = 2 \cdot k \cdot T_{_H}$, $k = 0, 1, \ldots$), $High$ transmits either 1 or 0. To transmit '1', the $High$-secrecy task with an execution time less than $C_{_H}$ is submitted. To send '0', the $High$-secrecy task with an execution time equal to $C_{_H}$ is dispatched. To receive '1' or '0', Low submits a Low-secrecy higher scheduling aperiodic request at $t = 4$, 20, ... (i.e., $t = 2 \cdot k \cdot T_{_H} + T_s$, $k = 0, 1, \ldots$); the size[1] of the aperiodic request should be a little bit greater than the value of C_s. In this example, the size of the aperiodic task is chosen to be $C_s + 1$, which is 2. If the response time of the Low-secrecy aperiodic task is equal to 2, it is interpreted as receiving '0' and if the response time is longer than 2, it is interpreted as '1'. In Figure 3, at $t = 0$, $High$ transmits '1' and, at $t = 9$, Low receives '1' (the response time of the Low-secrecy task is greater than 2). At $t = 16$, $High$ transmits '0' and, at $t = 22$, Low receives '0' (the response time is 2).

The above strategy clearly shows that covert timing channels can be constructed under PE scheduling even though the LSF rule is applied.

3.4 Weakest covert timing channel

Information security is commonly characterized as a weakest link problem. The information which organizations are trying to protect is only as secure as the weakest entry point to that information. This makes knowledge of the weakest link critical. In this section, the weakest (most insecure) timing channel is determined from seven different covert channels (Case II, III, and I-PE in Table 1) identified in the previous sections.

[1]The size means the time it requires to complete an aperiodic task without any interruption.

Our analysis in the subsequent sections is focused on the weakest covert timing channel case.

Since PE is a more complex scheduling algorithm than both DS and PS, a sophisticated transmission mechanism is required to construct a covert timing channel as illustrated in the section 3.3. If PS scheduling is employed, an execution of an aperiodic task can be delayed (when the aperiodic task arrives after the polling server suspends itself) and this delay makes it hard for attackers to build an efficient covert timing channel.

Since DS is a capacity preserving algorithm, aperiodic requests can be serviced by a server at any point in time as long as a server capacity is available to consume. Thus, in Case II-DS and III-DS, aperiodic requests can be efficiently used to create a covert timing channel. From an attacker's perspective, Case II-DS is more or equally preferable to Case III-DS for the following reason. In Case II-DS, the *High*-secrecy aperiodic task is invoked only when it is necessary to covertly transmit information. In addition, the delay time of the *Low*-secrecy task can be always measured since it runs periodically and continuously to meet real-time requirements. However, in Case III-DS, the *Low*-secrecy aperiodic task must be continuously invoked and monitored to measure its delay (caused by the *High*-secrecy periodic task). Note that, in real-time systems, it is very rare that an aperiodic task runs continuously and periodically since it is typically event-driven and has soft or no real-time requirements.

Case II-DS (weakest covert timing channel) is used to illustrate a way to eliminate a covert timing channel (Section 3.5) and calculate the security-performance tradeoffs (Section 4).

3.5 Elimination of covert timing channel

To eliminate or reduce the impact of covert timing attacks, many different defensive measures have been devised [12, 13, 14, 21, 34]. One popular defensive measure is to distort the accuracy of a system clock to prevent a task from accessing an accurate time source. Another is to schedule tasks in a round-robin fashion so that each task can only run during an allocated time slot. However, these measures are not acceptable for applications running on real-time systems. In real-time systems, tasks must have access to an accurate time source and are executed according to scheduling algorithms to meet their real-time constraints. Another common way to eliminate covert timing channels is to add a service request (noise) in order to cause unwanted delays in the response time of a *Low*-secrecy task. For real-time systems, an injection of noise into the system should not result in violation of real-time requirements such as deadlines, utilization bounds, etc. In this section, we demonstrate a way to remove covert timing channels for the weakest (most insecure) scenario, II-DS.

In Case II-DS, covert timing channels exist since the execution time of the *High*-secrecy aperiodic task $\mathbf{T}_H^{\mathcal{A}}$ varies from 0 to C_s (deferrable server capacity) units of time over every server period T_s and this causes various amounts of delay to the *Low*-secrecy periodic task $\mathbf{T}_L^{\mathcal{P}}$. A security measure to remove a covert flow is to submit an aperiodic task to a scheduler and run it until a server capacity *Cap* is fully consumed during each server period T_s. This full aperiodic task execution (i.e., full server capacity consumption)

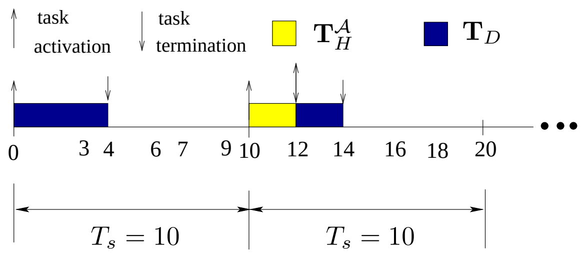

Figure 4: Injection of noise ($\mathbf{T}_D$) to eliminate covert timing channels

causes a maximum preemption delay in the response time of the *Low*-secrecy task $\mathbf{T}_L^{\mathcal{P}}$. As a result, *Low* always observes the worst (maximum) response time. In order to fully execute an aperiodic task (i.e, fully consume server capacity), a trusted aperiodic dummy task $\mathbf{T}_D$ with a scheduling priority higher than $\mathbf{T}_L^{\mathcal{P}}$ is dispatched and executed if there is no $\mathbf{T}_H^{\mathcal{A}}$ to execute or $\mathbf{T}_H^{\mathcal{A}}$ runs less than C_s units of time. Once $\mathbf{T}_D$ is dispatched, it continues to execute until Cap is fully consumed ($Cap = 0$). The execution of $\mathbf{T}_D$ can be considered as noise injected into a covert communication channel to block any information transmitted by $High$.

An example (Figure 4) is used to explain a method of preventing covert timing channels for Case II-DS. The example assumes that a server s has the period $T_s = 10$ and the worst computation time $C_s = 4$. At $t = 0$, the first server period begins and the server capacity Cap is replenished to its full capacity C_s, (i.e., $Cap = C_s = 4$). Since there is no $\mathbf{T}_H^{\mathcal{A}}$ to execute, $\mathbf{T}_D$ is dispatched and executed until Cap is exhausted. At $t = 4$, as Cap is fully depleted ($Cap = 0$), the execution of $\mathbf{T}_D$ is completed. At $t = 10$, the second server period begins and Cap is replenished to its full capacity ($Cap = C_s = 4$). The server is about to execute $\mathbf{T}_H^{\mathcal{A}}$ (which arrives at $t = 10$ or after the first dummy task is dispatched and is placed in an aperiodic job queue). At $t = 12$, the execution of $\mathbf{T}_H^{\mathcal{A}}$ is completed and Cap is reduced to 2 units of time. Since 2 units of time are still available in Cap, the second $\mathbf{T}_D$ is dispatched and the server starts to execute it. At $t = 14$, the execution of $\mathbf{T}_D$ is terminated as Cap is fully diminished ($Cap = 0$). In the both periods, the aperiodic task (either $\mathbf{T}_H^{\mathcal{A}}$ or $\mathbf{T}_D$) runs for 4 units of time during each server period. This always adds the maximum amount of delay to $\mathbf{T}_L^{\mathcal{P}}$ and *Low* observes a (single) worst response time for all requests it submits to a scheduler.

4 Security - performance tradeoffs

If the existence of a covert timing channel is identified, a quantitative analysis should be carried out to know how dangerous a given covert channel is. Traditionally, Shannon's information theory [8, 37] is used to quantify the amount of information flow through covert channels. In the following section, we provide an overview of Shannon's information theory.

4.1 Shannon's information theory

Mathematically, one can view a channel as a probabilistic function that transforms a sequence of input symbols, $x \in X$=$\{x_1, \ldots, x_k, \ldots, x_K, \}$, into a sequence of output symbols, $y \in Y$=$\{y_1, \ldots, y_j, \ldots, y_J\}$. We assume that the number of inputs and outputs of a channel are finite and the current output depends on only the current input. Such a channel is called a discrete memoryless channel (DMC).

Because of noise in a communication system, this transformation is typically not a one-to-one mapping from the set X of input symbols to the set Y of output symbols. Instead, any particular input symbol $x_k \in X$ may have some probability $p(y_j \mid x_k)$ of being transformed to the output symbol $y_j \in Y$. $p(y_j \mid x_k)$ is called a (forward) transition probability. Given a DMC, the probability distribution of the output set Y, denoted by Q_Y, can be calculated in matrix form as:

$$Q_Y = \begin{pmatrix} p(y_1) \\ p(y_2) \\ \vdots \\ p(y_J) \end{pmatrix} = \begin{pmatrix} p(y_1 \mid x_1) & \ldots & p(y_1 \mid x_K) \\ p(y_2 \mid x_1) & \ldots & \\ \vdots & \vdots & \vdots \\ p(y_J \mid x_1) & \ldots & p(y_J \mid x_K) \end{pmatrix} \begin{pmatrix} p(x_1) \\ p(x_2) \\ \vdots \\ p(x_K) \end{pmatrix} \tag{1}$$

Let $Q_{Y|X}$ be a matrix which has the transition probabilities of a noisy channel as its entities and Q_X represent the probability distribution of the input set X. Then, Eq (1) is abbreviated as:

$$Q_Y = Q_{Y|X} Q_X$$

According to Shannon's information theory, the entropy $H(X)$ is a measure of the information per input symbol $x \in X$ and is defined as: $H(X) = \sum_{k=1}^{K} p(x_k) \log{(1/p(x_k))}$. Similarly, the entropy $H(Y)$ is defined as $H(Y) = \sum_{j=1}^{J} p(y_j) \log{(1/p(y_j))}$. The conditional entropy $H(X \mid Y)$ and $H(Y \mid X)$ are defined as:

$$H(X \mid Y) = -\sum_{k=1}^{K}\sum_{j=1}^{J} p(y_j)p(x_i \mid y_j) \log p(x_i \mid y_j)$$

and

$$H(Y \mid X) = -\sum_{k=1}^{K}\sum_{j=1}^{J} p(x_i)p(y_j \mid x_i) \log p(y_j \mid x_i)$$

The average amount of the information transmitted over a channel is defined in information theory as the mutual information $I(X;Y)$. For notational convenience, $Q_Y(j)$ and $Q_X(k)$ represent the j^{th} and k^{th} entries of the column vectors Q_Y and Q_X. $Q_{Y|X}(j,k)$ or $Q_{j|k}$ represents the entry that lies in the j^{th} row and the k^{th} column of the matrix $Q_{Y|X}$.

$$\begin{aligned} I(X:Y) &= \sum_{k=1}^{K}\sum_{j=1}^{J} Q_X(k)Q_{j|k}\ log \frac{Q_{j|k}}{\sum_{i=1}^{K}\ Q_X(i)Q_{j|k}} \\ &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \end{aligned} \tag{2}$$

For a fixed transition probability matrix $Q_{Y|X}$, the mutual information $I(X;Y)$ is a function of the probability distribution Q_X of the set X of input symbols. The maximum mutual information achieved for a given transition probability matrix is the channel capacity $\mathbf{C}$:

$$\mathbf{C} = \underset{Q_X}{max}\ I(X;Y) \tag{3}$$

Note that channel capacity (bits/symbol) is found by maximizing $I(X;Y)$ with respect to Q_X for a given transition probability matrix. We denote the Q_X which maximizes $I(X;Y)$ as $Q^{max}_{X_H}$. When a channel is noiseless, there exists a one-to-one mapping between the input set X and the output set Y. The channel capacity (bits/symbol) of a noiseless channel is $\log n$, if $|X| = |Y| = n$. $\frac{\log n}{D}$ (bits/sec) is also commonly used as a channel capacity where D denotes the transmission time of a symbol.

4.2 Modeling covert timing channels

In this section, using Shannon's information theory, we model the covert timing channel identified in the II-DS (worst timing channel) case. A covert channel can be modeled as an unintended communication path through which an information receiver (*Low*) observes or receives a symbol transmitted by an information sender ($High$). In the case of II-DS, the $High$-secrecy higher-priority aperiodic task $\mathbf{T}^{\mathcal{A}}_H$ can add various amounts of delay to the *Low*-secrecy low-priority periodic task $\mathbf{T}^{\mathcal{P}}_L$ as an information sender. As an information receiver, *Low* can observes various response times due to the delays introduced by $High$.

Let e_H be a total amounts (units) of time for which $\mathbf{T}^{\mathcal{A}}_H$ runs to *influence* a response time of $\mathbf{T}^{\mathcal{P}}_L$ during each period T_L of $\mathbf{T}^{\mathcal{P}}_L$. In addition, e^{max}_H is denoted as the maximum value of e_H and, thus, $0 \leq e_H \leq e^{max}_H$. e_H is viewed as an input symbol to a covert communication channel. The value e^{max}_H is affected by the worst computation time C_L of $\mathbf{T}^{\mathcal{P}}_L$. Figure 5 shows how e^{max}_H (or the range of e_H) is influenced by C_L. In the example, $\mathbf{T}^{\mathcal{A}}_H$ and $\mathbf{T}^{\mathcal{P}}_L$ are running under DS scheduling (Case II-DS) with the deferrable server s with $T_s = 5$ and $C_L = 2$. Figure 5(b) shows that the worst response time R_L of $\mathbf{T}^{\mathcal{P}}_L$ is influenced only by the full execution of the aperiodic task in the first server period ($R_L = 3$). Thus, $e^{max}_H = 2$ or $0 \leq e_H \leq 2$. However, as shown in Figure 5(c), when the length ($C_L = 5$) of the task is long enough to carry over to the second server period, the delay of $\mathbf{T}^{\mathcal{P}}_L$ is caused by the aperiodic task executions in the first and second period ($R_L = 9$). Thus, $e^{max}_H = 4$ and $0 \leq e_H \leq 4$.

Since $0 \leq e_H \leq e^{max}_H$, a set X_H of input symbols to the covert channel is $X_H = \{x_1, x_2, \ldots, x_n\} = \{0, 1, 2, \ldots, e^{max}_H\}$, where x_i represents e_H being $(i-1)$ units of time and $n = e^{max}_H + 1$. A set Y_L of output symbols consists of different response times *Low* can measure during its period T_L, assuming that Low always submits a task with fixed computation time to a scheduler to observe or detect a response time of its own task. As long as the *Low*-secrecy task is schedulable, *Low* can observe n different response times, namely, $Y_L = \{y_1, y_2, \ldots, y_n\}$. Note that y_n is the worst response time R_L of $\mathbf{T}^{\mathcal{P}}_L$.

The covert timing channel identified in the II-DS case be viewed as a (noiseless) channel which maps a sequence of input symbols, $x_i \in X_H = \{x_1, x_2, \ldots, x_n\} = \{0, 1, 2, \ldots, e^{max}_H\}$, directly into a sequence of corresponding output symbols, $Y_L = \{y_1, y_2, \ldots, y_n\}$.

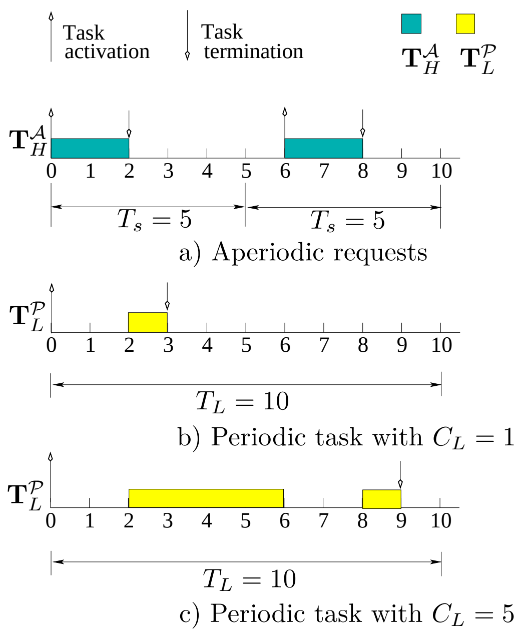
Task activation
Task termination
$\mathbf{T}_H^{\mathcal{A}}$
$\mathbf{T}_L^{\mathcal{P}}$
0 1 2 3 4 5 6 7 8 9 10
$T_s = 5$
$T_s = 5$
a) Aperiodic requests
$T_L = 10$
b) Periodic task with $C_L = 1$
$T_L = 10$
c) Periodic task with $C_L = 5$

Figure 5: The worst computation time C_L influences e_H^{max}

Through this covert timing channel, an input symbol $x_i \in X_H$ can be transformed into $y_i \in Y_L$ every T_L units of time and $|X_H| = |Y_L| = n = e_H^{max} + 1$. Thus, the covert timing channel capacity is $\mathbf{C} = \log(e_H^{max} + 1)$ (bits/symbol) or $\log(e_H^{max} + 1)/T_L$ (bits/sec).

4.3 Security-Performance tradeoff analysis

Many real-time systems must process both hard and soft real-time tasks [2, 33]. Their primary goal is to guarantee Quality of Service (QoS) requirements of soft tasks without jeopardizing strict schedulability requirements (no deadline miss) of hard tasks. The consequence of missing deadlines of hard tasks could be catastrophic. In contrast, miss-

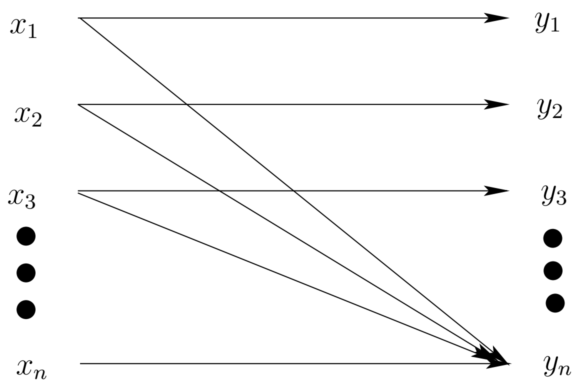

Figure 6: Injection of noise with probability ρ into covert timing channel

ing deadlines of soft tasks can be tolerated (e.g., missing deadlines of a few multimedia video frames is merely annoying). The typical performance metrics for guaranteeing QoS requirements are a job failure rate (miss rate), average response time, etc. The job failure rate[2] or miss rate is the average percentage of missed deadlines. A key indicator for the performance of real-time systems is CPU utilization. Maintaining CPU utilization of a system under a bound is very important. In general, as CPU utilization of a system reaches or exceeds a utilization bound, the job failure ratio starts to increase dramatically [1]. Thus, an appropriate method should be applied to effectively control the utilization under a bound condition, e.g. new incoming requests are admitted into the system only if a utilization bound is not exceeded.

In addition to real-time requirements, real-time systems must satisfy security requirements to be usable in MLS environments. The security mechanism introduced in Section 3.5 injects a dummy trusted task as noise into a covert communication channel to block covert communication. The injection of noise induces extra CPU overhead, thereby increasing CPU utilization. This increased CPU utilization may cause a sudden increase in the job failure rate or rejection of newly arrived real-time tasks. Since the goals of timeliness and security requirements are usually conflicting, there is a strong need to develop a solution which will balance between the requirements for security and the resulting performance [29, 36]. In addition, the solution must accompany a rigorous analytical framework which allows one to quantitatively measure the performance-security tradeoffs.

Security and performance can be optimized by employing a technique that switches back and forth between two modes of operations with a probability ρ: an insecure mode, in which all tasks are scheduled and executed without security measures and a secure mode, in which security measures are applied to eliminate covert timing channels. The noisy communication channel shown in Figure 6 has a transition probability $p(y_n|x_i) = \rho$. This means that noise is injected into the communication channel in such a way that all the inputs are transformed into y_n with probability ρ. We call ρ a noise factor. For example, with $\rho = 1$, y_n is the only output symbol that *Low* as an information receiver can observe and thus the channel capacity becomes zero.

The remaining of the section is devoted to perform the trade-off analysis for the most insecure case, II-DS case (see Section III-D). The security measure (injection of a dummy task as noise in the II-DS case) introduced in Section 3.5 can be modeled as the following noisy communication channel (Figure 6). All the notations used to model the (noiseless) communication channel in Section 4.2 can be reused with the addition of $p(y_n|x_i) = \rho$, $i \neq n$: as a security measure, a trusted dummy task is injected as noise with probability ρ, causing the maximum delay to the *Low*-secrecy task $\mathbf{T}_L^{\mathcal{P}}$ and, thus, the worst response time y_n of $\mathbf{T}_L^{\mathcal{P}}$.

Channel capacity formulation

Let $Q_{X_H}^{max}$ be the probability distribution of input set X_H which maximizes the amount of information flow R through the noisy channel shown in Figure 6. Since the noisy channel has $p(y_n|x_i) = \rho$, $i \neq n$, $Q_{X_H}^{max}$ has the same input probability distribution for all input

[2]The job failure rates are also an important performance metric to hard tasks when statistical analysis is used [3].

symbols except x_n, i.e. $p(x_1) = p(x_2) = \ldots = p(x_{n-1}) = \alpha$ and $p(x_n) = 1 - (n-1)\alpha$; from this, the range of the input probability α can be evaluated as $\alpha \leq \frac{1}{(n-1)}$. The output probability distributions are $p(y_1) = p(y_2) = \ldots = p(y_{n-1}) = \alpha(1-\rho)$ and $p(y_n) = 1 - (n-1)(1-p)\alpha$. To find the channel capacity, it is necessary to differentiate the mutual information R with respect to α. However, R is not differentiable when $\alpha = \frac{1}{(n-1)}$ (R is bounded by $\alpha = \frac{1}{(n-1)}$ and not continuous at $\alpha = \frac{1}{(n-1)}$). Thus, we have two cases to consider: when $\alpha < \frac{1}{(n-1)}$ (boundary condition I) and when $\alpha = \frac{1}{(n-1)}$ (boundary condition II).

Case I, when $\alpha < \frac{1}{(n-1)}$ (boundary condition I):

$H(Y_L)$ and $H(Y_L|X_H)$ are computed as:

$$\begin{aligned} H(Y_L) &= -(n-1)(1-p)\alpha\log(1-\rho)\alpha \\ &\quad - \{1-(n-1)(1-p)\alpha\}\log\{1-(n-1)(1-p)\alpha\} \end{aligned}$$

$$\begin{aligned} &H(Y_L|X_H) = \\ &\quad -(n-1)(1-\rho)\alpha\log(1-\rho) - (n-1)\rho\alpha\log\rho \end{aligned}$$

Then, we can compute the mutual information R_1 as:

$$\begin{aligned} R_1 &= H(Y_L) - H(Y_L|X_H) \\ &= -(n-1)(1-\rho)\alpha\log\alpha + (n-1)\rho\alpha\log\rho \\ &\quad - \{1-(n-1)(1-\rho)\alpha\}\log\{1-(n-1)(1-\rho)\alpha\} \end{aligned}$$

To determine the channel capacity $\mathbf{C}_1$, we differentiate R_1 with respect to α (R_1 is differentiable when $\alpha < \frac{1}{(n-1)}$). This gives:

$$\begin{aligned} \frac{dR_1}{d\alpha} &= -(n-1)(1-\rho)\log\alpha + (n-1)\rho\log\rho \\ &\quad + (n-1)(1-\rho)\log\{1-(n-1)(1-\rho)\alpha\} \end{aligned}$$

The value α which satisfies $\frac{dR_1}{d\alpha} = 0$ is denoted by α_1. After a few algebraic steps, the expression for α_1 is found to be:

$$\alpha_1 = \frac{1}{(n-1)(1-\rho) + \rho^{\frac{\rho}{\rho-1}}} \tag{4}$$

We substitute α_1 of Eq(4) for α used in the expression R_1 to obtain a formula for the channel capacity $\mathbf{C_1}$. Thus, we can express $Q_{X_H}^{max}$ and $\mathbf{C_1}$:

$$Q_{X_H}^{max} = \{\alpha_1, \alpha_1, \ldots, 1-(n-1)\alpha_1\} \tag{5}$$

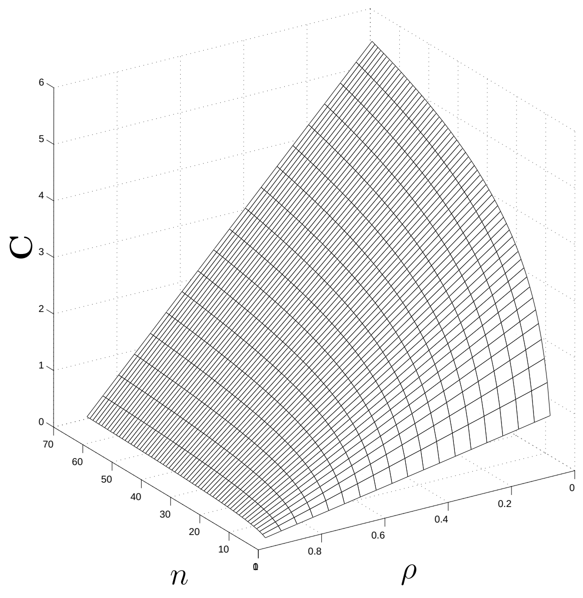

Figure 7: Covert timing channel capacity $\mathbf{C}$ vs. noise factor ρ and number of symbols, n.

$$\mathbf{C_1} = \log[1 + (n-1)(1-\rho)\rho^{\frac{\rho}{1-\rho}}]$$

To find the ranges of n (a number of symbols) and ρ (noise factor) where the formula $\mathbf{C_1}$ holds, α in the boundary condition I is replaced by α_1:

$$\begin{gathered}
\alpha < \frac{1}{(n-1)} \quad \text{(boundary condition I)} \\
\alpha_1 < \frac{1}{(n-1)} \\
\frac{1}{(n-1)(1-\rho) + \rho^{\frac{\rho}{\rho-1}}} < \frac{1}{(n-1)} \\
\rho^{\frac{1}{\rho-1}} > n-1 \qquad (6)
\end{gathered}$$

Thus, with the boundary condition of Eq(6), the precise formulation of the channel capacity for the case 1 is:

$$\mathbf{C_1} = \log[1 + (n-1)(1-\rho)\rho^{\frac{\rho}{1-\rho}}] \quad if\ \rho^{\frac{1}{\rho-1}} > n-1 \qquad (7)$$

Case II, when $\alpha = \frac{1}{(n-1)}$ (boundary condition II):

When $\alpha = \frac{1}{(n-1)}$, the following conditions hold:

- $p(x_1) = p(x_2) = \ldots = p(x_{n-1}) = \alpha = \dfrac{1}{(n-1)}$
- $p(x_n) = 0$
- $p(y_1) = p(y_2) = \ldots = p(y_{n-1}) = (1-\rho)\alpha = \dfrac{1-\rho}{n-1}$
- $p(y_n) = 1-(n-1)(1-p)\alpha = \rho$

From the above conditions, $H(Y_L)$ and $H(Y_L|X_H)$ can be evaluated as:

$$H(Y_L) = -(1-\rho)\log\frac{1-\rho}{n-1} - \rho\log\rho$$
$$H(Y_L|X_H) = -(1-\rho)\log(1-\rho) - \rho\log\rho$$

Using the $H(Y_L)$ and $H(Y_L|X_H)$ calculated in the above step, the formula for the mutual information R_2 is:

$$\begin{aligned} R_2 =& H(Y_L) - H(Y_L|X_H) \\ =& (1-\rho)\log(n-1) \end{aligned}$$

Since R_2 is not a function of α, the channel capacity $\mathbf{C_2}$ of the noisy channel is simply R_2. The channel capacity $\mathbf{C_2}$ and the input probability distribution $Q^{max}_{X_H}$ for the case 2 are expressed as:

$$\mathbf{C}_2 = (1-\rho)\log(n-1) \tag{8}$$
$$Q^{max}_{X_H} = \{\alpha_2, \alpha_2, \ldots, 0\},\ where\ \alpha_2 = \frac{1}{(n-1)} \tag{9}$$

Based upon $\mathbf{C}_1$ of Eq(7) and $\mathbf{C}_2$ of Eq(8), the noisy covert channel capacity $\mathbf{C}$ is formularized as:

$$\mathbf{C} = \begin{cases} \log[1+(n-1)(1-\rho)\rho^{\frac{\rho}{1-\rho}}] & if\ \rho^{\frac{1}{\rho-1}} > n-1 \\ \\ (1-\rho)\log(n-1) & otherwise. \end{cases}$$

Figure 7 shows how the channel capacity of the noisy covert channel changes as the number of symbols n and the noise factor ρ vary. As shown in the figure, the channel capacity $\mathbf{C}$ with n symbols decreases as the noise factor increases (more noise is injected into the channel as a security measure).

Performance overhead formulation

A real-time system in the secure mode experiences some performance degradation if noise (dummy task) is injected to reduce the covert channel capacity. The total execution time e_N of the dummy task $\mathbf{T}_D$ during the period T_L of the *Low*-secrecy task is a

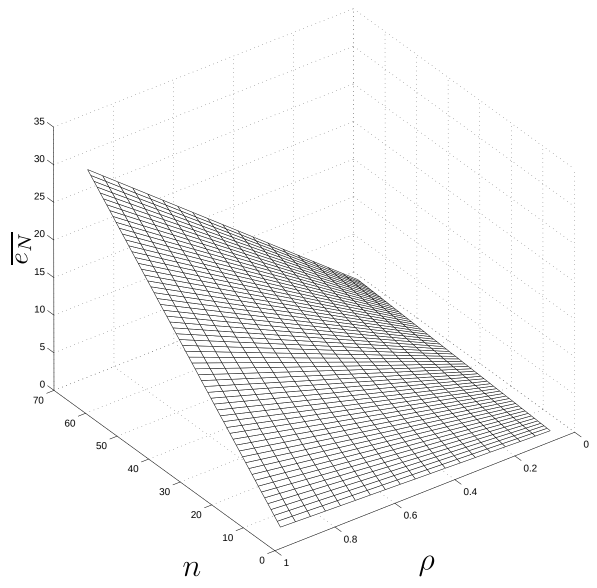

Figure 8: Average execution time $\overline{e_N}$ v.s noise factor ρ and number of symbols, n.

performance overhead to the real-time system. In the noisy channel model, the transmission of x_i means the *High*-secrecy task runs for $i-1$ units of time and, as a result, the dummy task is executed for $e_H^{max} - (i-1)$ units (i.e., $e_N = e_H^{max} - (i-1)$). Since the execution time e_N varies depending upon timed behaviors of the *High*-secrecy task, we use an average execution time $\overline{e_N}$ as the quantity of the performance loss. The formula for $\overline{e_N}$ is:

$$\begin{aligned}\overline{e_N} &= \rho \cdot \sum_{i=1}^{n-1} p(x_i) \cdot (e_H^{max} - (i-1)), \\ &\quad where \;\; x_i \in X_H, \;\; e_H^{max} = n-1 \\ &= \rho \cdot \sum_{i=1}^{n-1} p(x_i) \cdot (n-i) \end{aligned} \tag{10}$$

To evaluate $\overline{e_N}$, the input probability distribution Q_{X_H} should be known beforehand. Typically, it is expected that timed behaviors of *High*-secrecy aperiodic tasks follow a set of well-known probability distributions such as exponential arrival rates and uniform distribution for task execution times. However, we cannot assume that the *High*-secrecy (malicious) task follows well-defined statistical distributions. Instead, an appropriate assumption should be that *High* always tries to utilize the covert channel to maximize its capacity. Thus, $Q_{X_H}^{max}$ of Eq(5) and Eq(9) are used to evaluate $\overline{e_N}$. If $\rho^{\frac{1}{\rho-1}} > n-1$ (boundary condition I), $p(x_i)$ in Eq(10) is replaced by α_1 of Eq(4). Otherwise, $\alpha_2 = \frac{1}{(n-1)}$ is substituted for $p(x_i)$:

$$\overline{e_N} = \begin{cases} \frac{1}{2} \cdot \rho \cdot \alpha_1 \cdot (n-1) \cdot n & if \quad \rho^{\frac{1}{\rho-1}} > n-1 \\ \\ \frac{1}{2} \cdot \rho \cdot n & otherwise. \end{cases}$$

Figure 8 shows how $\overline{e_N}$ varies as the noise factor ρ and the number of symbols n change; the overhead $\overline{e_N}$ gradually goes up (performance suffers) as ρ increases (more noise is injected to the covert communication channel with n symbols).

5 Discussion & conclusion

In this paper, we address the covert timing channel issues in scheduling a set of hybrid tasks for MLS real-time systems. Specifically, the timing vulnerabilities of three fixed-priority scheduling algorithms such as Polling Server (PS), Deferrable Server (DS), and Priority Exchange (PE) are identified and security measures for removing covert timing channels are proposed. Since the goals of timeliness and security requirements of real-time systems are usually conflicting, we show a way to build an analytical model which allows one to quantitatively measure the performance-security tradeoffs and formally specify both security and real-time requirements.

There are many different approaches that deal with scheduling a set of hybrid tasks other than the fixed-priority scheduling algorithms on which we concentrate in this paper. Another approach is to use a server to service aperiodic requests in a *dynamic*-priority system where scheduling priorities of tasks running change over time [11, 30]. Other approaches are based on the concept of *slack stealing* [24, 25] and the *dual priority* mechanism [9, 10]. In our future research, we plan to identify the timing vulnerabilities on each scheduling approach (dynamic, slack-stealing and dual-priority mechanism) and build an analytical model which allows developers to exercise performance-security tradeoffs. In addition, we are going to investigate how our analysis method can be used in the area of information flow control in the avionic domain [23].

References

[1] T. F. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *RTAS '01: Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01)*, page 15, Washington, DC, USA, 2001. IEEE Computer Society.

[2] B. Al-Duwairi and G. Manimaran. Combined scheduling of hard and soft real-time tasks in multiprocessor systems. In *High Performance Computing, HiPC*, pages 279–289, 2003.

[3] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proc. of the 19th IEEE Real-time Systems Symposium*, pages 123–132, Dec. 1998.

[4] N.C. Audsly, A. Burns, M.F. Richardson, K. Tindell, and A.J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[5] D. E. Bell. Looking back at the bell-la padula model. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 337–351, Washington, DC, USA, 2005. IEEE Computer Society.

[6] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[7] S. Cabuk, C. Brodley, and C. Shields. IP Covert Timing Channels: Design and Detection. In *Proc. ACM conference on Computer and Communications Security*, 2004.

[8] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[9] R. Davis. Dual priority scheduling: A means of providing flexibility in hard real-time systems. Tech. Rep YCS230, University of York, UK, 1994.

[10] R. Davis and A. Wellings. Dual priority scheduling. In *RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 100, Washington, DC, USA, 1995. IEEE Computer Society.

[11] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Syst.*, 9(1):31–67, 1995.

[12] W. Hu. Reducing Timing Channels with Fuzzy Time. In *IEEE Symposium on Research in Security and Privacy*, May 1991.

[13] J. Janeri, D. Darby, and D. Schnackenberg. Building Higher Resolution Synthetic Clocks for Signaling in Covert Timing Channels. In *The Eighth IEEE Computer Security Foundations Workshop*, 1995.

[14] M. H. Kang, I. Moskowitz, and S.Chincheck. The pump: A Decade of Covert Fun. In *21st Annual Computer Security Applications Conference*, 2005.

[15] J. P. Lehoczky, L. Sha, and Y. Ding. Rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. of the 11th IEEE Real-time Systems Symposium*, pages 166–171, Dec. 1989.

[16] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *IEEE Real-Time Systems Symposium*, pages 261–270, 1987.

[17] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[18] J. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[19] H. Mantel and H. Sudbrock. Comparing Countermeasures against Interrupt-Related Covert Channels in an Information-Theoretic Framework. In *20th IEEE Computer Security Foudnations Workshop*, pages 326–340, 2007.

[20] R. Meyden and C. Zhang. Information flow in systems with schedulers, part 1: Definitions. *Theoretical Computer Science*, 467:68–88, 2013.

[21] I. Moskowitz and M. H. Kang. Covert Channels - Here to Stay. In *Proc. COMPASS 94*, pages 235–243, 1994.

[22] S. Mou, Z. Zhao, S. Jiang, Z. Wu, and J. Zhu. Feature extraction and classification algorithm for detecting complex covert timing channel. *Computers & Security*, 31(1):70–82, February 2012.

[23] K. Müller, M. Paulitsch, S. Tverdyshev, and H. Blasum. Improving performance of network covert timing channel through huffman coding. *Mathematical and Computer Modelling*, 55(1-2):69–79, 2012.

[24] I. Ripoll, A. Crespo, and A. García-Fornes. An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems. *IEEE Trans. Softw. Eng.*, 23(6):388–400, 1997.

[25] R. Sandra and J. P. Lehoczky. On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems. In *IEEE Real-Time Systems Symposium*, pages 160–171, 1993.

[26] J. Son and J. Alves-Foss. Covert Timing Channel Analysis of Rate Monotonic Real-Time Scheduling Algorithm in MLS Systems. In *Proc. IEEE Workshop on Information Assurance*, pages 361–368, 2006.

[27] J. Son and J. Alves-Foss. Covert Timing Channel Capacity of Rate Monotonic Scheduling Algorithm in MLS Systems. In *The IASTED International Conference on Communication, Network, and Information Security*, 2006.

[28] J. Son and J. Alves-Foss. A Formal Framework for Real-Time Information Flow Analysis. *Computers & Security*, 28(6):421–432, 2009.

[29] S. Son, R. Mukkamala, and R. David. Integrating security and real-time requirements using covert channel capacity. *IEEE Trans. Knowledge and Data Eng.*, 12(6):865–879, 2000.

[30] M. Spuri, G. Buttazzo, and F. Sensini. Robust aperiodic scheduling under dynamic priority systems. In *RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 210, Washington, DC, USA, 1995. IEEE Computer Society.

[31] P. A. Srihari, R. M. Venkatesan, and S. Bhattacharya. Opportunistic scheduling of secure tasks in a multiprocessor environment. *J. Integr. Des. Process Sci.*, 3(2):63–78, 1999.

[32] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Trans. Comput.*, 44(1):73–91, 1995.

[33] P. Tan, H. Jin, and M. Zhang. A hybrid scheduling scheme for hard, soft and non-real-time tasks. In *ISORC '06: Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, pages 20–26, Washington, DC, USA, 2006. IEEE Computer Society.

[34] J. Gray III. On introducing noise into the Bus-Contentiion Channel. In *IEEE Symposium on Research in Security and Privacy*, 1993.

[35] M. Völp, C.J. Hamann, and H. Härtig. Avoiding timing channels in fixed-priority schedulers. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 44–55, 2008.

[36] R. Watson, W. Morrison, C. Vance, and B. Feldman. The TrustedBSD MAC Framework: Extensible Kernel Access Control for FreeBSD 5.0. In *USENIX Annual Technical Conference, FREENIX Track*, pages 285–296, 2003.

[37] W. Weaver and C. E. Shannon. *The Mathematical Theory of Communication*. University of Illionois Press, 1963.

[38] J. Wu, Y. Wang, L. Ding, and X. Liao. Improving performance of network covert timing channel through huffman coding. *Mathematical and Computer Modelling*, 55(1-2):69–79, 2012.