Fall 2018

Improved CRPD analysis and a secure scheduler against information leakage in real-time systems

Ying Zhang

IMPROVED CRPD ANALYSIS AND A SECURE SCHEDULER AGAINST INFORMATION LEAKAGE IN REAL-TIME SYSTEMS

by

YING ZHANG

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2018

Approved by:

George Markowsky, Advisor
Venkata Sriram Siddhardh Nadendla
Linda Markowsky

ABSTRACT

Real-time systems are widely applied to the time-critical fields. In order to guarantee that all tasks can be completed on time, predictability becomes a necessary factor when designing a real-time system. Due to more and more requirements about the performance in the real-time embedded system, the cache memory is introduced to the real-time embedded systems.

However, the cache behavior is difficult to predict since the data will be loaded either on the cache or the memory. In order to taking the unexpected overhead, execution time are often enlarged by a certain (huge) factor. However, this will cause a waste of computation resource. Hence, in this thesis, we first integrate the cache-related preemption delay to the previous global earliest deadline first schedulability analysis in the direct-mapped cache. Moreover, several analyses for tighter G-EDF schedulability tests are conducted based on the refined estimation of the maximal number of preemptions. The experimental study is conducted to demonstrate the performance of the proposed methods.

Furthermore, Under the classic scheduling mechanisms, the execution patterns of tasks on such a system can be easily derived. Therefore, in the second part of the thesis, a novel scheduler, roulette wheel scheduler (RWS), is proposed to randomize the task execution pattern. Unlike traditional schedulers, RWS assigns probabilities to each task at predefined scheduling points, and the choice for execution is randomized, such that the execution pattern is no longer fixed. We apply the concept of schedule entropy to measure the amount of uncertainty introduced by any randomized scheduler, which reflects the unlikelihood of for such attacks to success. Comparing to existing randomized scheduler that gives all eligible tasks equal likelihood at a given time point, the proposed method adjusted such values so that the entropy can be greatly increased.

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who have helped me with my studies at Missouri S&T. Foremost, I would like to thanks Dr. George Markowsky, my advisor, for giving me selfless support and kindly help. His motivation, enthusiasm, and immense knowledge encourage me to overcome many difficulties in my graduate study. Secondly, I would like to express my appreciation to Dr. Zhishan Guo and Dr. Wei Jiang. I am grateful to them for giving me an opportunity to work with such outstanding researchers during my graduate study. Their patient and valuable suggestions guided my research path, which helps me to complete this work.

Further, I would like to thank my committee members, Dr. Venkata Sriram Siddhardh Nadendla and Dr. Linda Markowsky, for serving on my thesis committee and for taking the time to review this work. The feedback and discussion were of great value to me. I appreciate the tremendous help from all the faculty members and staff of the Department of Computer Science at Missouri S&T.

Last but not the least, I would like to thank my parents Yubin Zhang and Guiling Zhao and my boy friend Lingxiang Wang for their constant and unconditional love throughout my life.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOMENCLATURE

COTS	Commercial Off the Shelf
CRPD	Cache-related preemption Delay
DBF	Demand Bound Function
GEDF	Global Earliest Deadline First
RWS	Roulette Wheel Selection
WCET	Worst-Case Execution Time

1. INTRODUCTION

Real-time systems are widely used to support various critical computations and control systems, such as navigation of automobiles and avionics (Mohan *et al.*, 2014; Sampigethaya *et al.*, 2008; Shaout and McGirr, 2013). These systems are usually designed in a multi-task programming pattern with a sequence of sub-tasks for execution. The completion of sub-tasks is constrained by predefined deadlines in order to meet certain constraints in real life (Puaut *et al.*, 2003). For instance, for the autopilot functionality in an automobile, the system should be able to finish every sub-task on time, such as collecting images from sensors and making decisions based on these images. A logical and temporal failure of this sub-task would bring a sequence of serious problems.

In order to guarantee temporal correctness, *predictability* is a significant property when designing real-time systems (Stankovic and Ramamritham, 1990), (i.e., in order to guarantee every task's response within the assigned deadline, it is important to have a priori estimation of the task's execution time). In recent decades, with the rapid advancement of real-time embedded systems, commercial off-the-shelf components (COTS) are widely used. Regarding the performance requirements (Liptay, 1968; Prete *et al.*), the cache memory is introduced to real-time systems. However, the cache behavior is difficult to predict (Ferdinand and Wilhelm, 1999). Obtaining a reasonable estimation of task execution time becomes a challenge when tasks are running with the cache. Meanwhile, recent research illustrates that cache side-channel attacks are powerful tools for attackers to compromise the confidentiality of the target system, especially for real-time systems. Most tasks are running periodically including the encryption tasks. This gives a higher chance for attackers to steal information in the system.

In this thesis, the impact of cache in real-time systems will be studied twofold. First, in order to avoid pessimistic estimation of the worst case execution time (WCET) due to the cache interference, provide a more precise method for analyzing global earliest deadline first (GEDF) schedulability test, we integrate the cache-related preemption delay (CRPD) into GEDF for multiprocessor scheduling analysis; Second, we propose a randomized scheduling algorithm that can mitigates information leakage in real-time systems. The following sections will present the importance of integrating CRPD into GEDF and information leakage in real-time systems.

1.1. CACHE-RELATED PREEMPTION DELAY IN GEDF

With the rapid growth of Cyber-Physical Systems (CPS) and Internet of Things (IoT) (Lee, 2008), multiprocessors (Brucker, 1998) have been widely used in embedded real-time systems in the last decade. The increasing computing power of multiprocessors provides the embedded real-time systems with higher capacity but lower cost. For example, as a leading microprocessor provider in embedded systems, ARM has released its ARMv8-A structure with multi-core architectures, while a series of real-time operating systems, such as VxWorks, have been upgraded to fully support multi-core processors.

For further improvement, CPU cache is considered, it increases the average speed of accessing data , which fills in the gap between processor and memory speeds. However, the complexity of cache behaviors lead to pessimistic estimate of WCET. For example, when a task is running, references are loaded from the cache or the memory, the time of loading instruction and data becomes unpredictable. Furthermore, in real-time preemption systems, multiple reloads from memory to cache might be conducted due to preemptions. This additional delay is CRPD. It will cause an identical delay in the worst-case execution time (WCET) (Pellizzoni and Caccamo, 2007). Usually, when researchers and system designers estimate the WCET, the overhead and hardware-related costs (including CRPD, context switch and scheduler costs) will be included in the WCET of each task. In terms of schedule, preemption has no cost, and the WCET of each task is fixed. As a result, the scheduling problem becomes more relaxed. But on the other hand, the WCET is overestimated. In the real system, the execution time will be far less than the WCET. The CPU will be idle in most of the time, which will cause a drastic waste of resources. Therefore, an improved approach that decouples the WCET and CRPD is proposed. Under this circumstance, the previous analysis cannot be used since the task execution time depends on its behavior whether the data can be referenced on the cache or from the memory. To address these issues, various methods have been proposed. In this thesis, we mainly focus on bounding CRPD to WCET during the schedulability analysis.[1]

1.2. INFORMATION LEAKAGE IN REAL-TIME SYSTEMS

Predictability may lead to information leakage in real-time systems. Assuming that all tasks finish execution at their WCET and that the released patterns are regular (periodic), then execution sequences of given tasks are identical from one hyper-period to another, under any classic scheduler (e.g., earliest deadline first (EDF) or deadline monotonic). Thus, a duplicated execution sequence can be detected and used to derive the executing information.

Since tasks access memory with regularity (the execution sequences are the same in different hyper-periods), adversaries can exploit the access time stamps of cache addresses during execution (Lipp *et al.*, 2016) and derive the execution pattern of the system. Moreover, attackers are able to gauge the precise execution time range of the critical/target task (Chen *et al.*, 2015) and further obtain the critical information of the critical task stored in the cache by launching *side-channel attacks* (Chen *et al.*, 2015; Yarom and Falkner, 2014) on the critical task.

[1]The original work was published on ICESS 2017 (Proceedings of the 14th IEEE International Conference On Embedded Software and Systems) Zhang *et al.* (2017). The introduction part have been rewritten by the author.

To defend against such attacks, the key is *to decrease predictability of the execution sequence while guaranteeing schedulability of the given task set*. Since these two aspects seem to contradict each other, an intuitive solution is to break the execution pattern so that the task execution sequences become varied in different hyper-periods. Hence, randomized scheduling schemes are applied to real-time critical systems, which makes it difficult for the attacker to derive a proper time range for a targeted task. Yoon *et al.* (2016) proposed a randomization scheduling protocol, *TaskShuffler*, which randomly enumerates the execution sequences of tasks. At each scheduling point, it first forms a candidate task set based on the inversion budget, then selects a task from the candidates with equal probability to decrease the regularity in the execution sequence.

Unfortunately, there are some limitations to the existing work. First, a pessimistic inversion time budget leads to a limited candidate task set. Due to the overestimation of higher priority jobs' workloads, some lower priority jobs are not in the candidate set though they are eligible for executing without affecting the schedulability. Second, the uniform distribution of task selection overlooks the side effect of job selection in further time slots; the job's execution is less uncertain because of the current decision.

Based on the aforementioned observations, this thesis proposes a novel scheduling method, roulette wheel scheduler (RWS), for preventing execution pattern leakage while guaranteeing the real-time correctness of a given system. The details of RWS will be given in Section 5.

1.3. CONTRIBUTION AND ORGANIZATION

In this paper, we study the schedulability of EDF on multiprocessor systems taking into account the CRPD and derive a tight bound of CRPD under such settings. We propose a novel CRPD analytical approach that extends the existing the state of the art of CRPD analysis (Ju *et al.*, 2007; Lunniss *et al.*, 2013) to GEDF scheduling. Specifically, while existing works (Ju *et al.*, 2007; Lunniss *et al.*, 2013) assumes that each released job of tasks causes a preemption of shared cache to the job in execution, therefore all cache blocks are inferred densely in an uniprocessor, our work leverages the nature of the *sparse interferences* between cache blocks distributed on multi-cores/multi-processors (Sebek, 2001).

To address the scheduling information leakage, a new scheduling algorithm, RWS is proposed, which can help the system to prevent the information leakage from the scheduling pattern. RWS provides a new scheduling rule for generating the task execution pattern in a randomized manner while guaranteeing timing correctness. By carrying uncertainty for the task's execution at each scheduling point, it tries to maximize the *randomness* of the schedule and distribute the different execution sequences evenly at each hyper-period.

The main advantages of RWS are summarized as follows:

- Roulette wheel selection strategies have been widely applied in industry (Tao *et al.*, 2013); they are easy to utilize in practical systems to deal with the scheduling problems (Omara and Arafa, 2010). Hence, RWS can be applied to real-time systems without modification of the system architecture.
- RWS can be used with fixed or dynamic priority scheduling algorithms.
- RWS considers all activated jobs as candidates at each scheduling point to enhance the anonymity of the execution sequence. In other words, every candidate can be selected to execute between two scheduling points while guaranteeing the schedulability.
- Offline RWS can reach the maximum scheduled entropy under system settings. As far as is known, this is the first work that can achieve optimal entropy under a given set of scheduling points.
- RWS is sustainable. The scheduling point can be set by adjusting the length of time slices according to various system settings and task sets.

Organization: The rest of the thesis is organized as follows: The next section (Section 2) consists of a literature survey of previous works. Section 3 presents the system model and notations for analyzing the CRPD in multi-core real-time systems. Then we introduce GEDF-CRPD Test – a new GEDF schedulability test on multiprocessors for CRPD analysis in Section 4. We extend the current CRPD analytic method to a multi-core system and propose an improved approach to bounds the CRPD via our GEDF-CRPD Test. Upon comparison to existing approaches, the experimental results show that our method converges to the demand bound function (DBF) with a tighter margin than other methods. Section 5 discusses a the detailed adversary model when considering the side-channel attack. The offline and online RWS randomization scheduling protocols for dealing with schedule information leakage are proposed. Section 6 summarizes the thesis.

2. LITERATURE REVIEW

In this section, three categories of papers will be reviewed: GEDF schedulability analysis, integration of CRPD into uni-processor EDF schedulability analysis, and cache side-channel attacks in real-time systems.

2.1. GEDF SCHEDULABILITY ANALYSIS

To understand the response time of embedded real-time systems, a number of studies (Baker, 2003, 2005; Baruah, 2007; Liu and Layland, 1973a) have been done to analyze schedulability in multi-processors.

As early as 1973, to analyze the performance of earliest deadline first (EDF) scheduling in the multiprocessor, Liu and Layland (1973a) studied a sufficient condition for guaranteeing schedulability of all tasks. Then, to derive the maximum execution (time) requirement for each task set, Baker (2003, 2005) proposed a global EDF (GEDF) schedulability test. Later in 2007, Baruah (2007) and Bertogna and Cirinei (2007) improved the GEDF test and developed a new schedulability test. Most recently, Sun and Lipari (2015) proposed a new schedulability test through response time analysis for GEDF.

However, existing studies on multiprocessors (multi-core) rarely take into consideration the delay caused by *preempting shared resources*. Most existing schedulability analyses are based on certain unrealistic assumptions, such as a zero time cost for preempting a shared resource. Note that preemption of shared resources commonly occurs in multiprocessor (multi-core) systems and could cause significant performance degradation (e.g., missing deadlines) in worst-case scenarios (Altmeyer *et al.*, 2012; Buttazzo, 2011; Davis and Burns, 2006). One common way to estimate the delay is to multiply the worst-case execution time parameters by a certain factor to cover potential delays caused by preempting shared resource – but this is often overly pessimistic (Staschulat *et al.*, 2005).

2.2. INTEGRATE CRPD INTO EDF ANALYSIS ON UNI-PROCESSOR

Among a wide range of delays caused by shared resources such as bus and main memory (Davis and Burns, 2006; Kim *et al.*, 2014), cache-related preemption delay (CRPD) (Negi *et al.*, 2003) is a crucial factor of schedulability guarantee in multiprocessor systems; however, CRPD is usually overestimated under EDF scheduling settings Baker (2003, 2005). In 2007, Ju *et al.* (2007) integrated the CRPD into EDF schedulability analysis in *uniprocessor* settings, where they took all possible direct preemptions of a single job into account. Following Ju's attempts, Lunniss *et al.* (2013) proposed an extended CRPD analysis for EDF, where they leveraged an ECB-union multiset approach and UCB-union multiset approach to bound the CRPD. This result provided a tighter bound of CRPD compared with the work of Ju *et al.* (2007).

Unfortunately, all aforementioned existing works only analyzes the CRPD for EDF on *uniprocessor* platforms. The upper bound of CRPD is yet unknown under EDF scheduling in multiprocessor embedded real-time systems.

2.3. CACHE SIDE-CHANNEL ATTACKS AND DEFENSE IN REAL-TIME SYSTEM

Cache side-channel attacks (Lipp *et al.*, 2016) have become a hot topic in recent years. Liu *et al.* (2015) showed an effective implementation of the PRIME+PROBE attack against the last-level cache. They conducted their implementation on x86 processors. Later on, 2016 introduced several attack case studies and extended the attacks to ARM processors which make embedded devices vulnerable. Chen *et al.* (2015) proposed a schedule-based side-channel attack on real-time systems. They injected a malware task into an idle period, which then captured the idle and busy period in the system. With the help of task parameters, attackers could reconstruct the execution sequence of the task set, which enabled them to perform a successful side-channel attack later.

Due to the wide usage of ARM processors in real-time systems, real-time researchers have proposed techniques to prevent information leakage. To prevent information leakage from high-priority to low priority tasks, Gruss *et al.* (2016) and Gülmezoğlu *et al.* (2015) introduced a *cache flushing* technique to clear up the items or invalidate the data from the cache when a task finished execution. Without the data leakage, attackers cannot further exploit the priority order and predict the execution pattern. In 2014, Mohan *et al.* (2014) provided a cache flushing method for defending against cache side-channel attack at the design phase. They mainly focused on fixed priority (FP) scheduling algorithms. Based on this work, Pellizzoni *et al.* (2015) proposed a generalized model for defending the potential information leakage in all possible shared resources. Meanwhile, some researchers tried to raise the level of difficulty needed to derive the execution sequence. Yoon *et al.* (2016) randomized the task execution so that attackers would have difficulty determining the narrow time range of the targeted task, which increased the difficulty of launching side-channel attacks.

3. MULTI-CORE SYSTEM MODEL AND NOTATIONS

In order to analyze the CRPD in GEDF schedulability test, we first describe the system model, terminology, and notations used in the rest of the paper.

Workload. We consider a multicore system which has a fixed number of processors shared on an on-chip one-level cache. Specifically, these processors do not have any private cache, as demonstrated in Figure 3.1. Henceforth, no migration delay for tasks will be considered due to *no partitioning and no private cache*. We assume a multiprocessor system with m

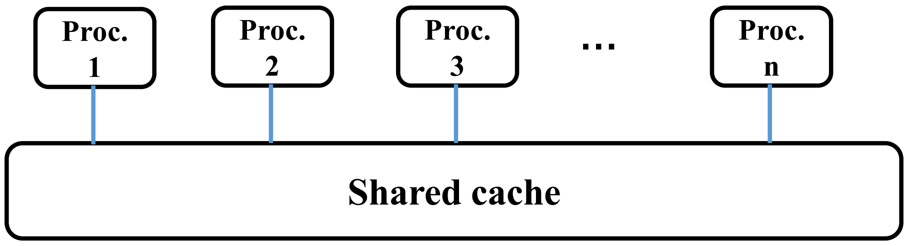


Figure 3.1. Cache model of the system: all cores share a same on-chip cache. Note that tasks executing on different cores do not have resource interference with each other although they share a common cache.

processors running a predefined sporadic task set under GEDF scheduling, with the total number of tasks $n \gg m$. Each task τ_i is defined as a 3-tuple $\{C_i, D_i, T_i\}$:

- C_i is the worst-case execution time for each job of the task.
- D_i is relative deadline for each job.
- T_i means each job of a task would be released at least every T_i time units.

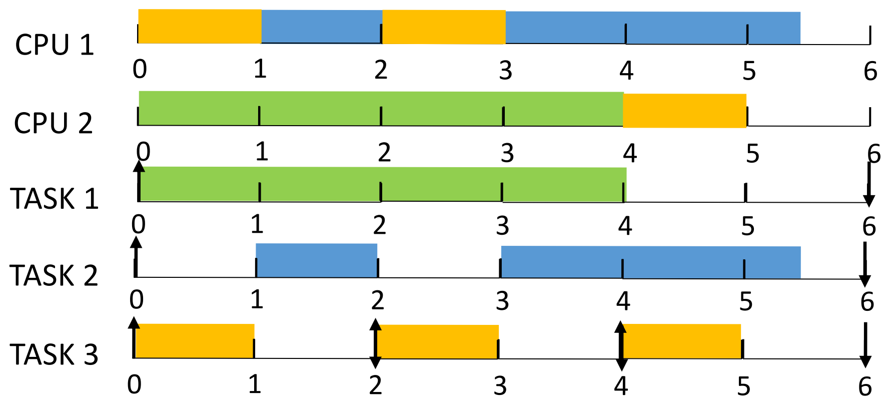
Each job has a absolute deadline d_i which occurs D_i time units after its release time.

We consider a constrained deadline in our system such that $D_i \leq T_i$ holds for all tasks. We consider a preemptive execution model, where during execution of a task, the executing job could be preempted or suspended at any instant, and its execution may resume later on the same processor or another.

Correctness. For a given scheduling algorithm, if all tasks can be scheduled without missing the deadline based on the specification of the system, we call the task set schedulable.

Task	Period	Deadline	WCET
τ_1	6	6	4
τ_2	6	6	3.5
τ_3	2	2	1

(a) A sample task set with three tasks.



(b) The GEDF schedule of the task set show in Table (a) (with two processors), where all tasks are released at time 0. The second job of task τ_3 preempts task τ_2 at time instant $t = 2$ when all processors are busy, while the third job of task τ_3 is scheduled into an idle slot at time $t = 4$.

Figure 3.2. GEDF scheduling of a sample task set.

In this paper, we consider the GEDF scheduling algorithm in multi-processor systems. GEDF is a dynamic scheduling algorithm that will place processes in a priority queue. A task's priority is assigned by the system based on its deadline. The task that has the earliest absolute deadline will have the highest priority; the task that has the latest absolute deadline will have the lowest priority Baruah (2007).

Example 3.0.1. *Consider the task set shown in Figure 3.2 (a), which can be correctly scheduled under GEDF (with the absolute deadline) as demonstrated in Figure 3.2 (b). Assuming a job arrived with an earlier absolute deadline, it is first scheduled into the idle time slots. If all the m processors are busy at that time instant, this newly released job will preempt the job with the lowest priority. Upon completion, a processor would choose from the pending jobs the one with the highest priority to execute.*

Observation 1. *Under the system and cache model are shown in Figure 3.1, for any newly released job to begin its execution at time t_0, if all processors are busy, the following two conditions must hold:*

- *Among the executing jobs, there are lower priority ones (i.e., with later absolute deadlines) than the job of interest.*
- *Only the job with the lowest priority that was executing will be preempted while all other jobs will not be preempted – they will continue their executions until a new job releases or they are finished.*

Notation. Assume that a job with the earliest absolute deadline has a higher priority. Let $hp(i)$ denote the set of tasks with smaller relative deadlines (and which can preempt task τ_i); i.e.,

$$hp(i) = \{\forall \tau_j | D_j < D_i\}. \tag{3.1}$$

Let $P_j(D_i)$ denote the maximum number of jobs belonging to task τ_j that are invoked during the executing a single job of task τ_i:

$$P_j(D_i) = max\Big(0, \lceil \frac{D_i - D_j}{T_j} \rceil\Big). \tag{3.2}$$

Let $E_i(t)$ represents the maximum number of jobs from task τ_i, which have their release times and deadlines within the time interval of length t and that of task τ_i that can be invoked. We calculate $E_i(t)$ as follows:

$$E_i(t) = max\left(0, 1 + \lceil \frac{t - D_i}{T_i} \rceil\right) \tag{3.3}$$

Demand Bound Function. We use $DBF(\tau_i, t)$ (Goossens *et al.*, 2003) to generate the maximum execution requirement from the jobs of τ_i that have both the arrival time and deadline within the time interval of length t. It can be calculated as follows:

$$DBF(\tau_i, t) = max\left(0, \left(\lfloor \frac{t - D_i}{T_i} \rfloor + 1\right) \cdot C_i\right) \tag{3.4}$$

where C_i is the Worst Case Execution Time (WCET) for a task τ_i. In GEDF scheduling, tasks can execution in different cores simultaneously, Note that the inter-core interference when tasks are running is taken into account in WCET.

To analyze the CRPD, we use the concept of useful cache block (UCB) and evicting cache block (ECB).

Lee *et al.* (1998) provided the definition for *UCBs* as "A memory block *m* is called a UCB at program point *P*, if it is cached at *P* and will be reused at program point *Q* that may be reached from *P* without the eviction of *m*". The memory blocks are loaded into the cache when a preempting task evicts other tasks, which are called ECBs. Combining the concept of UCBs and ECBs can help us in deriving a bound for CRPD.

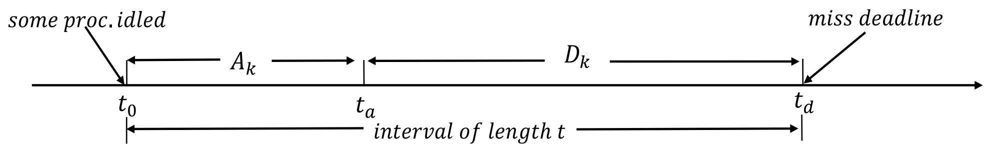


Figure 3.3. A job of task τ_k with arrival time at t_a and misses its deadline at t_d. t_0 is the time instant that at least one of the *m* processor is idle.

4. GEDF-CRPD: A MULTIPROCESSOR GEDF SCHEDULABILITY TEST FOR CRPD ANALYSIS

This section describes how CRPD analysis can be integrated into the existing schedulability test for GEDF on a multiprocessor platform. In order to do so, in Section 4.1, we briefly introduce the widely accepted GEDF schedulability analysis without incorporating CRPD. Then in Section 4.2, we propose four different methods to integrate CRPD into demand bound functions in GEDF schedulability analysis.

4.1. GLOBAL EDF SCHEDULABILITY TEST

Liu and Layland (1973a) explored the global multiprocessor scheduling of implicit deadline task. They gave a sufficient condition for guaranteeing that any task would not miss its the deadline:

$$u_{sum}(\tau) \leq m - (m-1) \cdot u_{max}(\tau). \tag{4.1}$$

In Equation 4.1, m denotes the number of processors, $u_{sum}(\tau)$ represents the total utilization and $u_{max}(\tau)$ represents the maximum utilization.

Later in 2007, Baker (2003, 2005) designed the GEDF schedulability test from a different perspective. He assumed that the task τ_k missed its deadline, and then determined the necessary conditions for other tasks, which resulted from task τ_k missing its deadline. As a result, the negation of the necessary condition is a sufficient condition to guarantee all deadline being met for the task set.

In 2007, Baruah (2007) designed a more sophisticated GEDF schedulability test that overcame some shortcomings in Baker's test. Similarly, he obtained a necessary condition that would let a job of task τ_k be the first to miss its deadline. When the necessary condition was not satisfied, then task τ_k would not have missed its deadline.

Based on this idea, we set t_d is the time instance that a job of τ_k first missed its deadline, we use t_a to denote this job's arrival time, where $t_a = t_d - D_k$ and t_0 as the latest time instant $\leq t_a$, at which at least one processor is idle in GEDF scheduling (Figure 3.3), In order to satisfy the deadline miss occurrence, it is necessary that all m processors execute jobs other than τ_k's job more than $(D_k - C_k)$ time units in the time interval $[t_a, t_d]$. Hence, the total amount of execution requirement that execution in this interval t should satisfy is:

$$\sum_{\tau_i \in \tau} I(\tau_i) > m \cdot (A_k + D_k - C_k). \tag{4.2}$$

We defined a time period $A_k = t_a - t_0$ in Equation 4.2, and $I(\tau_i)$ denotes the contribution of τ_i to work done in GEDF schedule during $[t_0, t_d]$.

If a task τ_i contributes no carry-in work[1] and the task τ_k does not miss its deadline, the contribution of τ_i to the total workload combined with the demand bound function should not exceed the Equation 4.3:

$$I_1(\tau_i) = min(DBF(T_i, A_k + D_k), A_k + D_k - C_k). \tag{4.3}$$

Based on Baruah's work, we establish that the total amount of execution demand for tasks should not exceed the total amount of slack time period in m processors. The Equation 4.1 can be extended to the following format:

$$\sum_{i=1}^{n} max(0, \lfloor (t - D_i)/T_i \rfloor + 1)C_i \leq m \cdot (A_k + D_k - C_k). \tag{4.4}$$

Without considering the CRPD in GEDF scheduling, the deadlines will be met, when the demand bound function satisfies Equation 4.4. However, for GEDF in a multiprocessor systems, when a higher priority task preempts lower priority tasks, the introduced CRPD will enlarge the demand bound function significantly, and the current sufficient condition cannot

[1]Carry-in work means that a job is released before t_0 and completes execution before t_d.

necessarily guarantee the schedulability of any sequence of tasks under GEDF scheduling. Figure 4.1 demonstrates that the given task set is no longer schedulable under GEDF when considering CRPD. Therefore, in the following subsection, CRPD will be integrated into the GEDF schedulability test framework introduced in this subsection.

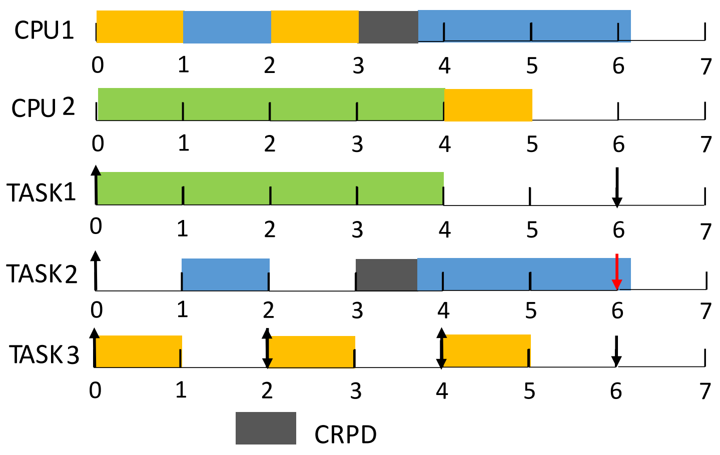


Figure 4.1. GEDF schedule of the taskset shown in Fig 2(a), where CRPD is taken into consideration and the first job of τ_2 misses its deadline at time $t = 6$.

4.2. INTEGRATE CRPD INTO GEDF SCHEDULING

For a given task, DBF calculates the execution requirement in the interval of length t. When considering the CRPD in GEDF scheduling, the execution requirement for each job of the task τ_i should integrate the cache reload time γ_i:

$$\sum DBF(\tau_i, t) = \sum_{i=1}^{n} max\Big(0, \lfloor \frac{t - D_i}{T_i} \rfloor + 1\Big) \cdot (C_i + \gamma_i). \tag{4.5}$$

In the remainder of the subsection, we will present four different approaches to calculating and bounding the CRPD, γ_i.

(A) Ju's Approach. Ju *et al.* (2007) presented an approach to integrate the CRPD analysis into uniprocessor EDF schedulability analysis. This approach first calculated the number of blocks belonging to τ_i that are directly preempted by task τ_j multiplied by $P_j(D_i)$. $P_j(D_i)$ is the maximum times that the task τ_j preempts a single job of task τ_i. In order to find all possible direct preemptions, the higher priority tasks that could preempt task τ_i are summed. These higher priority tasks τ_j are represented as $j \in hp(i)$ while γ_i^{Ju} represents the CRPD calculated by Ju *et al.* (2007):

$$\gamma_i^{Ju} = BRT \cdot \Big(\sum_{j \in hp(i)} P_j(D_i) \times \Big| UCB_i \cap ECB_j \Big| \Big). \tag{4.6}$$

where BRT is the per cache block reloading time. We modify Equation 4.5 and substitute γ_i from Equation 4.6, so that the CRPD can be calculated in DBF for GEDF schedulability analysis.

However, applying this method into GEDF would overestimate CRPD in a uniprocessor. For instance, if a task τ_j could preempt τ_i in a time instant, but the task τ_i has already been preempted by a higher priority task τ_k, this approach will calculate all possible preemptions into τ_i's response time, which pessimistically estimates the preemption time.

Lunniss *et al.* (2013) provided an improved CRPD analysis for EDF scheduling in uniprocessor systems in 2013. They used $\gamma_{t,j}$ to represent the $E_j(t)$ times of the preemptions cost for preempting tasks. These jobs would have had their release times and absolute deadlines in an interval of length t.

We can apply this concept in CRPD analysis under GEDF scheduling. Therefore, the DBF could be changed into the following format:

$$\sum DBF(\tau_j, t) = \sum_{j=1}^{n} \Big(max\{0, \lfloor \frac{t - D_j}{T_j} \rfloor + 1\} \cdot C_j + \gamma_{t,j} \Big). \tag{4.7}$$

There are mainly three approaches for calculating $\gamma_{t,j}$ in Equation 4.7: (B) ECB-union multiset[2], (C) UCB-union multiset approaches and (D) combined multiset approach, which are extended by Lunniss *et al.* (2013) to EDF scheduling based on the work of Staschulat *et al.* (2005) in fixed priority for the uniprocessor.

(B) ECB-union Multiset Approach. Nested preemptions make the pessimistic assumption that for any preemption by task τ_j, the task τ_j itself may have already been preempted by a higher priority task. The total number of times that the jobs of task τ_k can be preempted by jobs of task τ_j is equal to $P_j(D_k) \times E_k(t)$. Therefore, the multiset $M_{t,j}$ could be formed as follows:

$$M_{t,j} = \bigcup_{\forall k \in aff(t,j)} \Big(\bigcup_{P_j(D_k) \times E_k(t)} \Big| UCB_k \cap (\bigcup_{h \in hp(j) \cup j} ECB_h) \Big| \Big). \tag{4.8}$$

In the time interval t for each processor, the job of task τ_j could at most invoke $E_j(t)$ times, therefore, the ECB-union multiset approach bounds the CRPD by summing the $E_j(t)$ largest value in the multiset $M_{t,j}$ as shown in the following equation:

$$\gamma_{t,j}^{ecb-m} = BRT \cdot \sum_{l=1}^{E_j(t)} |M_{t,j}^{l}|. \tag{4.9}$$

BRT is the per block reloading time and $\gamma_{t,j}^{ecb-m}$ represents the CRPD calculated by ECB-union multiset approach.

(C) UCB-union Multiset Approach. This approach also uses the concept of multiset. Lunniss *et al.* (2013) first forms the multiset $M_{t,j}^{ucb}$. This multiset includes $P_j(D_k) \times E_k(t)$ times preemption of each task τ_k caused by task τ_j. Each time of preemption is represented by a set of cache blocks that might be preempted by task τ_j. Task τ_k whose

[2]Multiset is like a set, but it allows duplicate elements. For instance, {a, a, b} and {a,b} are not the same multiset. However, the order does not matter. For example, {a, a, b} and {a, b, a} are the same multiset.

relative deadline is greater than task τ_j's in the time interval [0, t) is presented as $\mathit{aff}(t, j)$:

$$M^{ucb}_{t,j} = \bigcup_{\forall k \in \mathit{aff}(t,j)} \Big(\bigcup_{P_j(D_k) \times E_k(t)} UCB_k \Big). \tag{4.10}$$

This forms the ECB multiset $M^{ecb}_{t,j}$, which contains the cache blocks that could be evicted by the jobs of task τ_j. Since τ_j invoked at most $E_j(t)$ times, the $M^{ecb}_{t,j}$ contains $E_j(t)$ times repeated ECBs preempted by a single job of τ_j:

$$M^{ecb}_{t,j} = \bigcup_{E_j(t)} (ECB_j). \tag{4.11}$$

Finally, the intersection of $M^{ucb}_{t,j}$ and $M^{ecb}_{t,j}$ is multiplied by the BRT obtaining the CPRD, which is represented by:

$$\gamma^{ucb-m}_{t,j} = BRT \cdot |M^{ucb}_{t,j} \cap M^{ecb}_{t,j}|. \tag{4.12}$$

where BRT is the per block reloading time and $\gamma^{ucb-m}_{t,j}$ indicates the CRPD calculated by the UCB-union multiset approach.

(D) Combined Multiset Approach. Since the UCB-union multiset and ECB-union multiset approaches are not comparable Lunniss *et al.* (2013), we get the minimum of these two results applying to the total DBF equation, which is represented as follows:

$$\begin{aligned} &\sum_j DBF(\tau_j, t) = \\ &\sum_j min\{DBF(\tau_j, t)^{ucb-m}, DBF(\tau_j, t)^{ecb-m}\}. \end{aligned} \tag{4.13}$$

where $DBF(\tau_j, t)^{ucb-m}$ indicates DBF obtained through the UCB-union multiset approach, Similarly, $DBF(\tau_j, t)^{ecb-m}$ represents DBF obtained by applying the ECB-union multiset approach.

Until now, we studied the GEDF schedulability test in a multiprocessor systems and integrated the CRPD into GEDF in multiprocessor systems. Moreover, we proposed four approaches to analysis the CRPD under GEDF. However, the ECB-union multiset approach, UCB-union multiset approach, and combined multiset approach assume that each released job of tasks can cause preemption of a shared cache. The maximum number of preemption times is decreased in a multiprocessor when compared to a uniprocessor. We will present improved CRPD analysis in Section 4.3.

4.3. AN IMPROVED CRPD UPPER BOUND ANALYSIS

In a multiprocessor system, approaches (A), (B), (C) and (D) given in Section III-B, usually over-estimate the CRPD under GEDF scheduling since they assume that each released job of tasks could generate a preemption cost. However, the cache interference of tasks would be reduced in the multiprocessor. Thus, we leverage the nature of the sparse interference between cache blocks distributed on the multiprocessor to obtain a tighter bound of CRPD.

4.3.1. Condensing the Multiset. One of the main difference between a uniprocessor and a multiprocessor is that the first m tasks with the earliest relative deadline would not be preempted by other tasks in the multiprocessor. According to observation one, if one of these tasks begins to execute when released then there exists some task in execution with an absolute deadline later than the first m task's absolute deadline. If the released task has a lower priority compared with some task with a later absolute deadline, the task would wait until one of the jobs completes execution in m processors.

Multiset approaches would include all the useful cache blocks that may be evicted in the time interval of length t. Since the first m tasks with the earliest relative deadline would not be preempted, these approaches all overestimate the affected cache blocks.

In the ECB-union multiset approach shown in Equation 4.8, when the task τ_k belongs to the task set $\{\tau_1, \tau_2, \cdots, \tau_m\}$, the intersections between UCBs and ECBs are considered empty in a multiprocessor system. Therefore, unless these values are not the l^{th} largest value in multiset M, the result will overestimate the CRPD. The equation below rectifies the limitation of Equation 4.8:

$$\left| UCB_k \cap (\bigcup_{h \in hp(j) \cup j} ECB_h) \right| = \emptyset, \quad k = 1, \cdots, m. \tag{4.14}$$

Similarly, in the UCB-union multiset approach, since the first m tasks will not be preempted, we can simply treat the UCB of these tasks as empty for calculations; and then obtain the following bound:

$$\left(\bigcup_{P_j(D_k) \times E_k(t)} UCB_k \right) = \emptyset, \quad k = 1, \cdots, m. \tag{4.15}$$

4.3.2. Reducing the Maximum Number of Preemptions. In the m multiprocessor system with the GEDF scheduling algorithm (Figure 3.3), when we find that t_0 is the idle point in at least one processor, m jobs belonging to different tasks in execution are in any time instant between t_0 and t_a. If a single job of task τ_i is released at a time instant t_i, even if it has the earliest absolute deadline, it would only preempt the task τ_l with the latest absolute deadline. Other tasks are not interrupted by the task τ_l. In this situation, their response time would not be extended by preemptions. The total number of invocation times for the higher priority tasks would be reduced when compared with the times in a uniprocessor. Therefore, the total preemption time can be decreased.

In a multiprocessor, when a task is preempted, it could resume in any processor including the processor it utilized before. We also include this case into the CRPD analysis since it could bring extra cache reloading time. With this underlying assumption, we mainly focus on how many preemptions occur in the time interval of length t.

In order to find the worst-case preemption time, we first assume there are n tasks in the system, and every single job of a task released would cause a preemption. We let the tasks with the latest m absolute deadline execute first, then release the second m tasks with a higher priority. The higher m priority tasks could preempt all the tasks executed in the processor, until the task with the earliest absolute deadline is released in one of the processors.

In this situation, the total preemption time should be $n - m$ when all jobs belong to different task complete the first time release. In fact, no matter the sequence of tasks, the total preemption time in the first time invocation for different tasks would not exceed $(n - m)$ times. Hence, we can subtract the m^{th} least preemption cost from the total CRPD.

Through condensing the multiset M in the combined multiset approach and refining the estimation of maximum preemption time, we can further estimate the tighter bound of CRPD in the multiprocessor case:

$$\begin{aligned}\sum_j DBF(\tau_j, t) = -\sum_{i=1}^{m} G^m + \\ \sum_j min\{DBF(\tau_j, t)^{ucb-D}, DBF(\tau_j, t)^{ecb-D}\}\end{aligned} \tag{4.16}$$

where G denotes the interfered cache blocks for the first released job of each tasks and G^m is the m^{th} minimal interfered cache blocks set. We use $DBF(\tau_j, t)^{ucb-D}$ to represent the DBF calculated using condensed UCB-union multiset approach and $DBF(\tau_j, t)^{ecb-D}$ to indicate the DBF calculated by the condensed ECB-union multiset approach.

4.4. EXPERIMENTAL AND EVALUATION

In this section, we evaluate the effectiveness of the different approaches in preemption cost computation of a large number of task sets with varying task set parameters. The task parameters used in our experiments were randomly generated as follows:

– The number of cores(m) is 2, 4, 8.

– The default task size is 15.

– The total number of task sets is 100.

– Task utilizations were generated using the UUnifast-discard algorithm Bini and Buttazzo (2005).

– Task period were generated according to a uniform distribution with a factor of 100 differences between the minimum and maximum possible task period and minimum periods of 5ms to 500ms, as found in most automotive and aerospace hard real-time applications.

– Task execution times were set based on the utilization and period selected: $C_i = U_i \cdot T_i$

– Task deadlines were implicit, i.e., $D_i = T_i$

– Priorities were assigned in deadline monotonic order.

The following parameters affecting preemption costs are given below, the default values is given in parentheses:

– The number of cache-sets (CS=256).

– The cache reuse factor is 80%.

– The block-reload time (BRT = 8 μs)

– For each task, the UCBs of each task were assigned randomly based on the result on Altmeyer *et al.* (2011)'s result.

The experiment shows how the integrated CRPD and global EDF schedulability analysis performed under the default configuration for an implicit deadline task set. We varied the utilization from 0.5 to m, and record how many task sets were deemed schedulable by the global schedulability assuming no preemptions. Then we compared this experimental result with the cases under different CRPD analysis approaches in global EDF.

The Figure 4.2, Figure 4.3 and Figure 4.4 show the result of 2 cores, 4 cores and 8 cores respectively. Each figure compared five approaches we proposed before. GEDF_CRPD_Ju describes the global EDF schedulability test for CRPD analysis based on Ju *et al.* (2007)'s work; i.e., Approach (A) in Section 4.2. GEDF_CRPD_ECB represents the schedulability test based on ECB-union multiset approach; i.e., Approach (B) in Section 4.2. GEDF_CRPD_UCB represents the schedulability test based on UCB-union multiset approach; i.e., Approach (C) in Section 4.2. GEDF_CRPD_cb represents the schedulability test based on combined multiset approach (i.e., Approach (D) in Section III-B.) and GEDF_CRPD_cb represents the schedulability test based on condensed multiset approach with the technology described in Section 4.2.

1

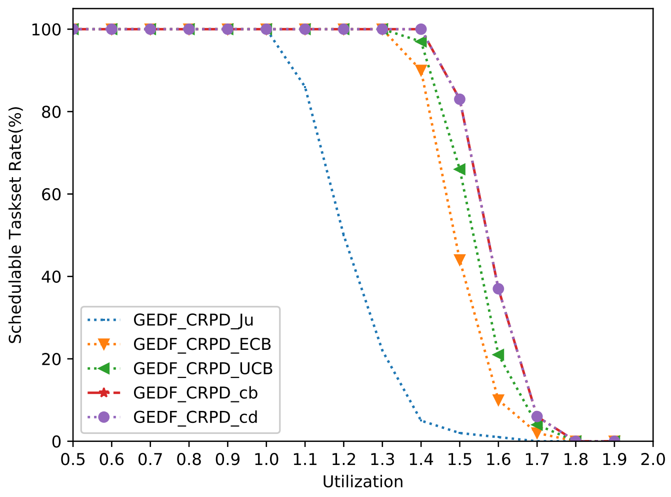


Figure 4.2. Evaluation for five CRPD analysis approaches: the number of tasksets could be schedulable at different total utilization in two processors.

After analyzing the figures, we find that GEDF_CRPD_Ju approach performs worst. Since it computes all possible preemptions caused by higher priority into a single job of tasks in DBF. Although the single direct preemption costs are precise, the total cost is very pessimistic. It overestimates the total cost of preemption. GEDF_CRPD_ECB and GEDF_CRPD_UCB approaches outperformed the Ju's approach. These two approaches have a very close performance with our task set. GEDF_CRPD_cb approach adopts the minimum value of GEDF_CRPD_ECB and GEDF_CRPD_UCB. Therefore, It performs better than these two approaches sometimes. Due to considering the sparse cache-block interference and refining the estimation of maximum preemption time in the multiprocessor, GEDF_CRPD_cd approach has the best performance with our task set. The experimental result shows that GEDF_CRPD_cd gives a tighter bound of CRPD.

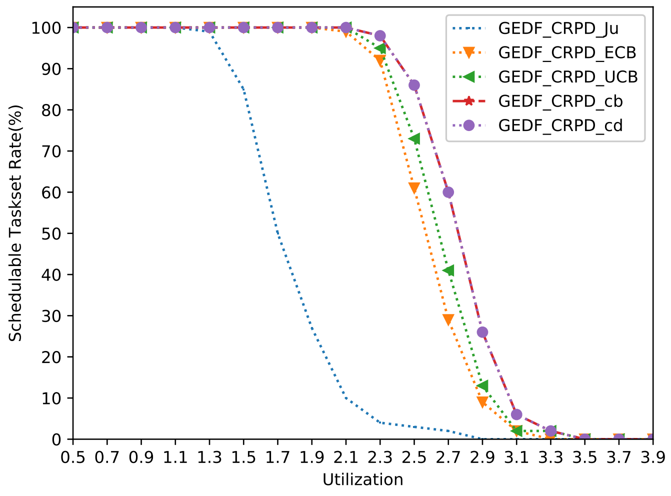


Figure 4.3. Evaluation for five CRPD analysis approaches: the number of tasksets could be schedulable at different total utilization in four processors.

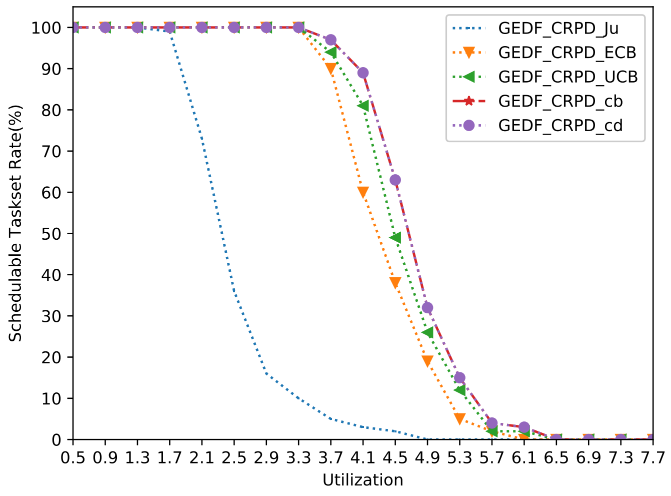


Figure 4.4. Evaluation for five CRPD analysis approaches: the number of tasksets could be schedulable at different total utilization in eight processors.

5. RWS - A ROULETTE WHEEL SCHEDULER FOR PREVENTING EXECUTION PATTERN LEAKAGE

As the introduction presents, the cache side-channel attacks will leak critical information to the attackers. Hence, in this section, a novel scheduling algorithm will be presented - RWS. The section is organized as follows: in Section 5.1, the system model will be redefined, and the potential threat of cache side-channel attacks will be discussed. RWS will be elaborated in Section 5.2. Finally, Section 5.3 is the evaluation and the discussion of RWS.

5.1. SYSTEM MODEL AND ADVERSARY MODEL

5.1.1. System Model and Terminology. We consider a predefined workload to be run on a fully preemptive uni-core discrete system. All parameters of tasks are *integers*. The workload consists of a set of periodic tasks $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ is synchronous. All tasks have implicit deadlines, so each task can be denoted as (C_i, T_i), where C_i is the WCET, and T_i is the period. A periodic task τ_i may generate an infinite number of jobs $\{\tau_{i,1}, \tau_{i,2}, \cdots\}$, where consecutive releases must be T_i time units apart. As an instance of a task, each job $\tau_{i,j}$ can be characterized by 3-tuple $(a_{i,j}, c_{i,j}, d_{i,j})$:

- $a_{i,j} \geq 0$ denotes its release time (the first moment that the job can start to execute);
- $c_{i,j} = C_i$ is the WCET;
- $d_{i,j} = a_{i,j} + T_i$ is the absolute deadline.

A job's priority is assigned by the the absolute deadline: any job with an earlier absolute deadline holds a higher priority. We use $hep(\tau_{i,j})$ to represent the jobs with deadlines at or before $d_{i,j}$, and $lp(\tau_{i,j})$ to denote the jobs with deadlines after than $d_{i,j}$.

System. We consider a uniprocessor platform. The technique can apply to a multiprocessor system if it is scheduled under fully partitioned scheme.

Remark 1. *In this thesis, we assume the predefined task set is feasible (since we are considering a uni-processor system), i.e., schedulable under the EDF scheduling algorithm, that satisfies:*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{5.1}$$

its demand bound function $dbf(t)$ Baruah et al. *(1990) at a time instant t should satisfy:*

$$dbf(t) = \sum_{i=1}^{n} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq t. \tag{5.2}$$

5.1.2. Attack Model and Motivation. Under such system settings, we assume that an attacker has *a priori* knowledge of the task set's parameters and the scheduling policy. The main goal of an adversary is *to utilize the deterministic schedules to launch an attack at a specific time instant on the targeted task.* Attackers are able to obtain sensitive information (e.g., a cryptography key) of the targeted task by side-channel attacks. For example, a system might be running an encryption task[1] periodically with a given input. The attacker can fill in the cache set first at the beginning of the task. After the targeted task finishes executing, attackers can read the cache blocks, measure the latency, and derive the crypto key (Lipp *et al.*, 2018), as demonstrated in Figure 5.1.

To perform such attacks successfully, attackers should be aware of the execution of a targeted task. With the prior knowledge of task parameters and the scheduling algorithms, attackers can derive the *precise* time range of the targeted task so that they are able to launch cache side-channel attacks at the beginning and the end of the targeted task to get the unencrypted data stored in the cache. However, if attackers launch the attack at arbitrary

[1]A encryption task is a task which performs an encryption algorithm to generate a ciphertext, such as RSA (Rivest *et al.*, 1978). The plaintext is hard to retrieve without the key. Attacks like brute-force are unlikely to succeed to retrieve the plaintext. Recently, side-channel attacks on RSA which can retrieve the plaintext in a short time have been proved (Bauer *et al.*, 2014).

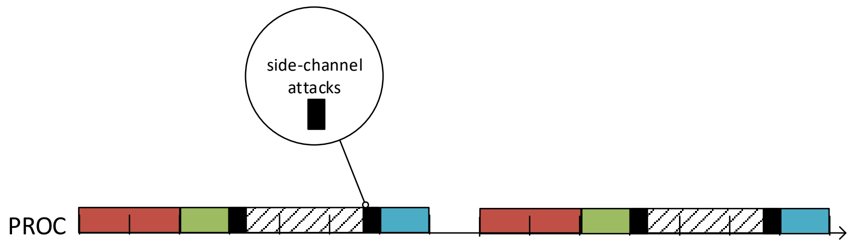


Figure 5.1. A potential attack by utilizing the deterministic schedules in real-time systems. The task filled with diagonal lines is an encryption task. After the attacks, attackers are capacble to retrieve the un-encrypted data.

instants, it is highly possible that the cache is filled with other unrelated data that does not belong to the targeted task. It would not know whether the attack has been successful or not (key data is retrieved).

Hence, the first step of such attacks is to monitor the system activities and to exploit the precise execution sequence of the task set (Chen *et al.*, 2015), which can help obtain the narrow execution time range of the targeted task. Take EDF as an example, attackers can build the schedule offline according to the task parameters; then, based on current system activities like the idle and busy period, the attackers can find the beginning of a hyper-period if a repetitive execution sequence exists. Therefore, the narrow time range of the targeted task can be derived, leading to very high success rate to data intrusion attacks.

As demonstrated in Example 5.1.1, the given task set is scheduled under the EDF scheduling algorithm, where task executions are fixed at each scheduling point, so the execution sequence is the same in two different hyper-periods. Attackers can successfully launch attacks because of the *predictability* of the scheduler.

Example 5.1.1. *Consider a task set $\tau = \{\tau_1, \tau_2\}$ with parameters shown in Table 5.1. Assume both tasks release their first job synchronously. The task set's execution sequence under EDF scheduling algorithm is shown in Figure 5.2 (*Proc*), where the execution sequence in [0,6) is the same as the execution sequence in [6,12). The execution pattern is $\langle \tau_1, \tau_2, \tau_1, idle \rangle$ in every hyper-period. From the attackers' view, it is possible that the* *execution sequence can be derived with the execution information in one hyper-period as shown by Chen* et al. *(2015). Assuming Task τ_2 contains sensitive information, the attacker can now launch a cache attack at time t=9 to obtain it. While for the randomized scheduler, it is unlikely that the second job of τ_2 will be at hot-cache state at t=9 (and in the example t=12 is the only time point for the attack to work, which is becomes a challenging guess for the attacker).*

Table 5.1. Parameters of a task set.

Task	WCET (C_i)	Period (T_i)
τ_1	1	3
τ_2	2	6

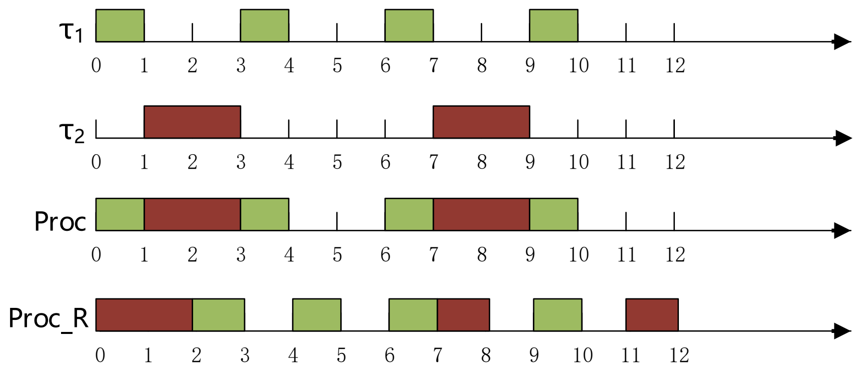


Figure 5.2. The line Proc is the execution sequence of the task set under the EDF scheduling algorithm, while the line Proc_R represents one possible scheduling scenario under randomized scheduling.

Assumption. We assume that the scheduler is trustworthy, i.e., the scheduler cannot be attacked. Otherwise, adversaries have a large attack surface to utilize. For example, adversaries can change the scheduling algorithm at an arbitrary instant in time to break the timing correctness, or they can make specific tasks always miss their deadline by changing the priorities. Also, we do not consider denial-of-service attacks (Neumann, 2000) as they are easily detected by the system. For example, when attackers insert dummy tasks, the response time of several tasks can exceed their deadlines. The system can detect these abnormal behaviors and shut down the current task (Abdi *et al.*, 2016; Gujarati *et al.*, 2017) running in the system. Hence, if the adversary launches a denial-of-service attack at an arbitrary moment without the knowledge of the scheduling information, it is highly possible that the inserted task will be dropped. Therefore, attackers' intrusions remain in the system as a normal task, carefully supervise the execution of the tasks before launching a precise attack on the critical task.

5.1.3. Problem Definition. To protect the critical information in real-time systems and to break the execution pattern of the task set under traditional scheduling algorithm, we want to generate a new scheduler which can randomize the current execution sequence. Instead of executing the job with the the highest priority (e.g., earliest deadline) at each scheduling point, the new scheduling algorithm will randomly select a job among a subset of ready jobs. Therefore, the tasks' execution sequence would be generated with *randomness*, as shown in *Proc_R* of Figure 5.2, which greatly increases the chance of failure of cache side-channel attacks.

To formalize the *randomness* in schedulers and to quantify the disorder among the execution sequence, we apply slot entropy and schedule entropy (Son *et al.*, 2006; Yoon *et al.*, 2016) to quantify the disorder, which refers to the concept of Shannon entropy (Shannon, 1948) in information theory.

Definition 5.1.1. (Slot entropy). Slot entropy measures the uncertainty of the task execution at a time instant t.[2]

$$H(t, \mathcal{S}) = -\sum_{\tau_{i,j} \in J(t)} p_{i,t}(\mathcal{S}) \times \log p_{i,t}, \tag{5.3}$$

where $p_{i,t}(\mathcal{S})$ is the probability assigned to job $\tau_{i,j}$ at the scheduling point t for a schedule $\mathcal{S}$. Note that at any time, there will be at most one job belonging to a task in the ready queue. Thus, we use i to index the release jobs. Slot entropy $H(t, \mathcal{S})$ represents the *randomness*

[2]We use log instead of $\log_2$ from now on.

metric of the job set $J(t)$ at scheduling point t. For the task execution sequence under a given scheduling algorithm such as EDF, at any scheduling point, the selection of a job is deterministic, so slot entropy remains zero for any slot.

Definition 5.1.2. (Schedule entropy). Schedule entropy $H(\mathcal{S})$ is a measure of the non-regularity associated with a given schedule $\mathcal{S}$ and a set of scheduling points $\mathcal{L}$ in a hyper-period of length L.

$$H(\mathcal{S}) = \sum_{t \in \mathcal{L}} H(t, \mathcal{S}). \tag{5.4}$$

Schedule entropy is associated with slot entropy since the probability distribution of a job would influence the *randomness* of the execution sequence. Schedule entropy quantifies the disorder of the execution sequence among different hyper-periods through measuring the randomness of task execution in one hyper-period. For traditional scheduling algorithms, execution sequence is the same for two hyper-periods so $H(\mathcal{S})$ is 0. Hence, we should try to maximize the value of $H(\mathcal{S})$ by increasing the randomness in the tasks' execution sequence, so that the possibility that execution sequences are the same in every hyper-period decreases.

Therefore, our problem can be defined as follows:
given a job set $J(t) = \{\tau_{1,j}, \tau_{2,k}, \cdots, \tau_{m,l}\}$ in the ready queue at time instant t, where $m \leq n$. All these jobs are ordered by their deadline in an ascending order. The assigned probabilities $p_{i,t}$ to each job must guarantee that every job completes its execution on or before the deadline, while the summation of the probability $p_{i,t}$ for executing each job cannot exceed 1 at any time instant t:

$$\forall \tau_{i,j} \in J, \mathcal{R}_{i,j} \leq d_{i,j} - a_{i,j} \tag{5.5}$$

$$\forall t, \sum_{i=1}^{n} p_{i,t} \leq 1, \tag{5.6}$$

where $\mathcal{R}_{i,j}$ is the response time of $\tau_{i,j}$ which is the total amount of time between the job's release and completion time.

Our goal is to maximize the schedule entropy in a hyper-period under Constraint (5.5) and (5.6):

$$\max_{\mathcal{S}} \ H(\mathcal{S}). \tag{5.7}$$

5.2. ROULETTE WHEEL SCHEDULER

To solve the scheduling problem mentioned above, we first propose offline RWS to randomize the execution sequence and generate different execution sequences with equal probability at each hyper-period for a periodic task set. Due to the high time complexity of the offline algorithm, we further provide an online RWS randomization scheduling algorithm that can generate the schedule in real-time. Finally we provide the schedulability test for the online RWS algorithm.

Table 5.2 lists the variables used in this section[3].

Table 5.2. Preliminary Variables

Variable	Definition
$m_{i,t}$	the length of the executed part of a job $\tau_{i,j}$ before time instant t
$r_{i,t}$	the remaining workload of a job $\tau_{i,j}$ at time instant t, i.e., $r_{i,t} = c_{i,j} - m_{i,t}$
$p_{i,t}$	the probability assigned to the job $\tau_{i,j}$ at time t
$hep(\tau_{i,j})$	a set of jobs with higher or equal priority than $\tau_{i,j}$
$lp(\tau_{i,j})$	a set of jobs with lower priority than $\tau_{i,j}$
Δ	the length of time slot

5.2.1. Offline RWS.

RWS is a strategy used to choose an item (among a set of items) proportional to its probability. RWS is a scheduling method, which makes scheduling decisions based on the probabilities of tasks while taking the schedulability into consideration.

[3]Index j is omitted for many variables for simplicity. It is safe to do so due to the reason that task can only release at most one job at any time instant.

In detail, RWS works under slice setting; i.e., the timeline is sliced into mini-slots (slices) of predefined length Δ. During run-time, each slice is assigned to one of the active jobs by the scheduler according to their probabilities.

Example 5.2.1. *Consider again the task set shown in Table 5.1 and Figure 5.2 of Example 5.1.1, the timeline Proc_R is divided into small slices of length* $\Delta = 1$. *At time instant* 0*, we assume that two jobs are in the ready queue* $J(0)$ *where their assigned probabilities are* $p_{1,0} = 0.4$ *and* $p_{2,0} = 0.6$*. Then, RWS has a 40% chance of picking* $\tau_{1,1}$ *and a 60% chance of picking* $\tau_{2,1}$ *at this time instant. Note that in RWS, one of the key factors is to assign the time slices properly to jobs without violating the timing constraints.*

We propose the offline RWS to randomize the execution sequence based on the time slice of the length equal to Δ. Offline RWS will sort the jobs in one hyper-period to a job queue J based on their priority allocation (the priority is determined by EDF scheduling algorithm; for tasks with the same deadline, the task with a smaller period has a higher priority). For every job $\tau_{i,j}$ in J, offline RWS randomly selects C_i/Δ time slots with in its deadline, then sequentially assigns sufficient time slices for every job until an execution sequence in a hyper-period is obtained. Then, it will iteratively execute this assignment process at the beginning of each hyper-period.

Example 5.2.2. *In Example 5.1.1, the sorted job queue is* $J = \{\tau_{1,1}, \tau_{1,2}, \tau_{2,1}\}$ *in the first hyper-period. Assume* $\Delta = 1$*, the timeline is divided into small slices as shown in Figure 5.3. At time instant 0, the scheduler will assign one of the three slices* $\{s_1, s_2, s_3\}$*,* $\{s_4, s_5, s_6\}$ *to* $\tau_{1,1}$ *and* $\tau_{1,2}$ *respectively (Figure 5.3). Then, it would pick two time slices from the remaining four to execute* $\tau_{2,1}$.

The process of choosing sufficient slices for jobs can be treated as a combination problem (Table 5.3). Assume Δ is divisible by C_i and T_i every job $\tau_{i,j}$ and at time instant t, the slots which is available for $\tau_{i,j}$ to run is $\mathcal{B}_{i,j}$. we randomly pick C_i/Δ time slices out of $\mathcal{B}_{i,j}$ time slices for executing job $\tau_{i,j}$, each job can generate $\binom{C_i/\Delta}{\mathcal{B}_{i,j}}$ different execution

Table 5.3. Parameters of a task set.

Job	WCET (C_i)	$\mathcal{B}_{i,j}$	Combinations
$\tau_{1,1}$	1	3	3
$\tau_{1,2}$	1	3	3
$\tau_{2,1}$	2	4	6

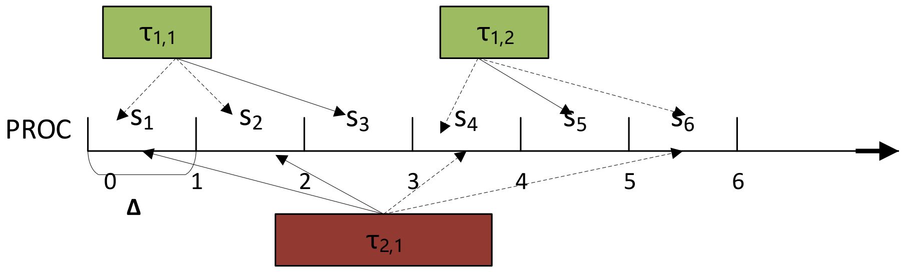


Figure 5.3. The timeline PROC is divided into small slices; each slice is one-time unit. The scheduler will assign sufficient time slices to each job at the beginning of a hyper-period.

sequences, combine all N job together to form a sequence in one hyper-period, total number of combinations D is:

$$D = \prod_{i=1}^{N} \binom{C_i/\Delta}{\mathcal{B}_{i,j}} \tag{5.8}$$

Hence, the scheduler is able to generate all possible combinations of execution sequences D. However, not every execution sequence in D satisfies the schedulability. Thus, there exists a $D' \in D$ which contains all correct solutions. Moreover, at the beginning of each hyper-period, it randomly generates one of the combinations out of D and verifies if this combination guarantees the time correctness for each task. Given that the possible combinations can be exponentially huge, it is unlikely that the execution patterns from two hyper-periods are identical. (e.g., Proc_R shows two possible execution sequences in different hyper-periods). Specifically, since the combination problem generates equally likely outcomes within the sample space D', the probability of every valid combination is $1/D'$. Under this condition, the schedule entropy would be maximized. The schedule entropy can be calculated as follows:

$$H(\mathcal{S}) = -(\sum_{i=1}^{D'} \frac{1}{D'} \log_2 \frac{1}{D'}) = \log_2 D' \tag{5.9}$$

Limitations. Offline RWS runs in pseudo-polynomial time to generate the one possible execution sequence due to the potentially exponential number of time slots in a hyper-period. Moreover, it cannot guarantee the schedulability of the given task set when generating the execution sequence.

5.2.2. Online RWS. To obtain a more efficient scheduler that is able to reduce the time complexity of verifying the sequence when generating, we propose the online RWS randomization scheduling approach.

Compared with the offline approach which generates the execution sequence at the beginning of the hyper-period, the online RWS constructs the execution sequence slot by slot. In detail, online RWS treats the beginning of every time slice Δ as a scheduling point and calculates the execution probabilities of each job in the waiting queue $J(t)$. Then, it makes the scheduling decision according to the execution probabilities.

In online RWS, jobs can be executed with probabilities since they allow priority inversion during execution while guarantee the schedulability. Hence, the probabilities should be assigned based on the $r_{i,t}$ and the possible idle time slots $B_{i,t}$ which allow priority inversion before its deadline.

The demand bound function $dbf(t)$ (Baruah *et al.*, 1990) represents the total workload belonging to jobs that deadline before or in t. However, to guarantee the schedulability of the task set, some jobs released before time t while having a deadline after the time instant t should be executed some time slices before time instant t. We derive the *adjusted demand bound function* $Adbf(d_i)$ to bound the actual workload that should be done before the jobs deadline d_i. We set the $Adbf = dbf$ at the end point of one hyper-period, then we calculate the $Adbf(d_i)$ from the end of a hyper-period to the beginning interactively based on $dbf(d_i)$ (Figure 5.4).

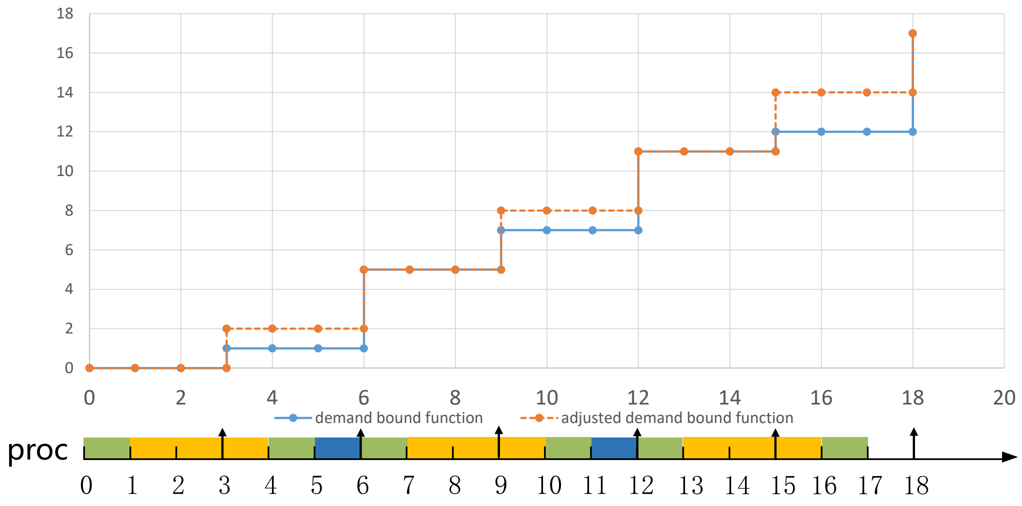


Figure 5.4. Assume a task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ with parameters $\tau_1 = (1, 3)$, $\tau_2 = (3, 6)$ and $\tau_3 = (1, 9)$, the demand bound function is shown as a solid line in one hyper-period, while the *adjusted demand bound function* is calculated from the end of the hyper-period shown as the dash line.

Assume that the time period between two adjacent deadlines is l, we check the workload that should be added in this period. The workload in this period is:

$$workload(l) = Adbf(d_{i+1}) - dbf(d_i) - (d_{i+1} - d_i). \tag{5.10}$$

if the workload is less than 0, we keep the $dbf(d_i)$ unchanging, else we reset the $dbf(d_i)$ as the following equation:

$$Adbf(d_i) = dbf(d_i) + workload(l); \tag{5.11}$$

based on the $Adbf$, we can determine the time slices that are available for each job to execute, we define the semi-inversion budget.

Definition 5.2.1. (Semi-inversion time budget). For a job $\tau_{i,j}$ in the ready queue $J(t)$ is released at time instant t, the semi-inversion time budget is:

$$B_{i,j}(t) = d_{i,j} - t - \Big(Adbf(d_{i,j}) - \sum_{k,l:d_{k,l} \leq d_{i,j}} M_{k,l}(t)\Big) + C_i. \tag{5.12}$$

$B_{i,j}(t)$ represents the total amount of available time (from $\tau_{i,j}$'s perspective) for lower priority jobs (including job $\tau_{i,j}$) to execute before the deadline $d_{i,j}$. $M_{k,l}(t)$ is the total executed workload amount of job $\tau_{k,l}$ before time instant t. Note that the budget $B_{i,j}(t)$ should be greater or equal to $r_{i,t}$ at any time instant t if the task set is schedulable under a given scheduling algorithm; otherwise, $\tau_{i,j}$ will miss its deadline.

Since the online RWS makes scheduling decision slot by slot, the scheduling decision would influence the value of the budget, online RWS updates the budget based on the current scheduling decision, which follows budget updating rule:

Budget Updating Rules. A job $\tau_{i,j}$ is selected to executed at a time slice, the budget for all $hep(\tau_{i,j})$ and itself $B_{i,j}(t)$ should be decreased by the length of a time slice.

According to the budget updating rule, we could get a new $B_{i,j}(t)$ for each job at a scheduling point. Online RWS uses a greedy approach to make the scheduling decision: the smaller the budget job, the higher probability of timely execution; so the later execution decision will not be affected by jobs which are close to their deadline. Therefore, we categorize jobs into three catehories: *normal, urgent* and *protected* according its updated budget.

Definition 5.2.2. At a time instant t,

$$\text{a job's } \tau_{i,j} \text{ type} = \begin{cases} \textit{Normal}, & \text{if } B_{i,j}(t) > r_{i,t} \\ \textit{Urgent}, & \text{if } B_{i,j}(t) = r_{i,t} \\ \textit{Protected}, & \text{if } r_{i,t} = d_{i,j} - t \end{cases} \tag{5.13}$$

For the jobs in the current state at a scheduling point, online RWS make the scheduling decision according to the following **Probability Assignment Rules**:

- If a job $\tau_{i,j}$ is *normal*, then this job allows priority inversion; the probability of job execution follows a certain distribution as shown in Equation (5.14).
- If a job $\tau_{i,j}$ is an *urgent* job, then this job cannot allow priority inversion. If jobs in $hep(\tau_{i,j})$ do not execute in this time slice, $\tau_{i,j}$ must be executed, otherwise it will miss the deadline. Thus, the probabilities of jobs in $lp(\tau_{i,j})$ will be zero, while the jobs in $hep(\tau_{i,j})$ and $\tau_{i,j}$'s execution follow a certain distribution as shown in Equation (5.14).
- If a job $\tau_{i,j}$ is a *protected* job, then it must be executed in this time slice. The probabilities for every other job is zero regardless of its priorities[4].

In the combination problem, the jobs assigned to time slices follow a conditional distribution in offline RWS; Therefore, to maximize the schedule entropy, the probability assignment in online RWS aims at generating the equally likely execution sequence based on the process in the offline RWS. Equation (5.14) calculates the job's probability as:

$$\begin{aligned} Proc(\tau_{i,j}) &= \frac{r_{i,t}}{B_{i,j}(t)}, \\ p_{i,t} &= \frac{Proc(\tau_{i,j})}{\sum_{\tau_{i,j}\in Q} Proc(\tau_{i,j})}. \end{aligned} \tag{5.14}$$

Where $Proc(\tau_{i,j})$ is the execution probability for a single job in combination problem at a time instant t. Since the summation of the probability should less or equal to 1 (Equation 5.6) at any time instant t, we set the exact execution probability as $p_{i,t}$ for each job.

The concrete methodology of a probability assignment is described in Algorithm 1. Function `Prob_Asn` is called at the beginning of every time slice. It takes ready queue J as the input, then outputs all active jobs' probabilities, which is based on the roulette wheel selection.

[4]Note that in any time instant t_0, only one protected exists. The proof is provided on Subsection 5.2C.

Algorithm 1 Probability Assignment Rules

```
 1: function PROB_ASN($Q, t_0$)
 2:     Proc = $\emptyset$
 3:     for each job $\tau_{i,j} \in J$ do
 4:         if $\tau_{i,j}$ is protected then
 5:             Proc[$\tau_{i,j}$] = 1
 6:             for each job $\tau_{k,l} \in Q$ do
 7:                 if $\tau_{k,l} \neq \tau_{i,j}$ then
 8:                     Proc[$\tau_{k,l}$] = 0
 9:                 end if
10:             end for
11:             break
12:         end if
13:         if $\tau_{i,j}$ is urgent then
14:             Proc[$\tau_{i,j}$] = $r_{i,t}/B_{i,t}$
15:             for each job $\tau_{k,l} \in lp(\tau_{i,j})$ do
16:                 Proc[$\tau_{k,l}$] = 0
17:             end for
18:             break
19:         end if
20:         if $\tau_{i,j}$ is normal then
21:             Proc[$\tau_{i,j}$] = $r_{i,t}/B_{i,t}$
22:         end if
23:     end for
24:     for each job $\tau_{i,j} \in Q$ do
25:         $p_{i,t_0}$ = Proc[$\tau_{i,j}$]$/\sum_{\tau_{i,j} \in Q} Proc[\tau_{i,j}]$
26:     end for
27:     return $p$
28: end function
```

In Algorithm 1, lines 4-12 indicate that if $\tau_{i,j}$ is protected, then the Proc of other jobs' value are zero. Line 13-19 shows that if $\tau_{i,j}$ is urgent, the value of Proc of jobs in $hep(\tau_{i,j})$ and $\tau_{i,j}$'s are defined by the ratio of the remaining workload and its budget, while $lp(\tau_{i,j})$'s Proc value is zero. Line 20-22 means if $\tau_{i,j}$ is a normal job, then we use the same equation to assign the Proc. In line 24-26, we apply the roulette wheel selection to calculate the probabilities of jobs and return the probabilities. We illustrate the algorithm by presenting the Example 5.2.3.

Example 5.2.3. *A periodic task* $\tau_3 = (2, 12)$ *is added to the task set shown in Table 5.1. Online RWS generates an execution sequence* $\langle \tau_{3,1}, \tau_{2,1}, \tau_{1,1}, \tau_{3,1} \rangle$ *in the first four slices. At the time instant 4, there are two jobs in the waiting queue, the parameters and the remaining workload are shown in Table 5.4. According to Equation (5.14), the probability ratio at time instant 4 is* $1 : 1$*. Therefore, the probability assigned for each job should be* $p_{1,4} = 50\%$; $p_{2,4} = 50\%$*. However,* $\tau_{2,1}$ *cannot be executed, even though it has 50% chance of executing, because* $\tau_{1,2}$ *is in the urgent state and does not allow priority inversion. Thus, in this scenario, when calculating the probability, we do not consider the* $\tau_{2,1}$ *as shown in Figure 5.5. The new probabilities assigned to them are* $p_{1,2} = 100\%$; $p_{2,1} = 0\%$.

Table 5.4. Probability distribution of a task set.

Job	Remaining Workload ($r_{i,t}$)	Budget($B_{i,t}$)	Probability(Pr)
$\tau_{1,2}$	1	1	100%
$\tau_{2,1}$	1	1	100%

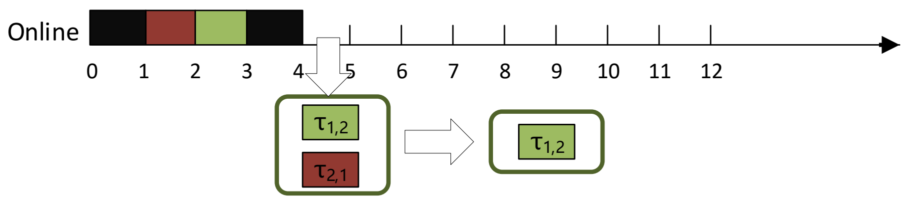


Figure 5.5. The scenario for resizing the waiting queue when some tasks are going to miss their deadlines. The rectangle denotes the waiting queue. Since $\tau_{1,2}$ or $\tau_{2,1}$ will miss its deadline if $\tau_{3,1}$ executes, we do not allow $\tau_{3,1}$ to execute in this scheduling point.

As Example 5.2.3 shows, online RWS updates the $B_{i,t}$ for each job $\tau_{i,j}$ at each scheduling point, it considers allocating execution probabilities to the jobs that are in the ready queue. Moreover, online RWS adjusts the probabilities within the waiting queue to guarantee schedulability. We discuss the correctness of online *RWS* in the next subsection.

5.2.3. The correctness of the online RWS. In Subsection B, we discussed the methodology of assigning the probability to jobs. We did not consider the correctness of Δ. An improper Δ will make a feasible task set unschedulable. We discuss the constraints of Δ here.

Comparing to the selection of the frame size in the cyclic execution Baker and Shaw (1989), the requirements of Δ are as follows:

1. Δ should divide some T_i (or Δ should divide the hyper-period).

2. Δ should divide all C_i.

3. Only one job can execute in Δ.

To simplify the calculation of schedule entropy, Δ should divide the hyper-period. The second and third requirements concern schedulability and entropy. When designing Δ, jobs should finish execution at the boundary of Δ, or there would be some idle time inside the Δ within the busy period, which might delay the execution of some tasks and cause these tasks to miss their deadline, as shown in Example 5.2.4.

Example 5.2.4. *Given two tasks $\tau_1 = (5, 3), \tau_2 = (6, 2)$ and $\Delta = 2$, a possible execution sequence under online RWS is shown in Figure 5.6. $\tau_{1,2}$ misses its deadline at $t = 10$ under the system settings. Because at time instant $t = 6$, $\tau_{2,2}$ and $\tau_{1,2}$ are* normal *jobs, online* RWS *will randomly pick a job to execute according to their priorities. $B_{1,6}$ cannot be updated until the next scheduling point, and the execution of $\tau_{2,2}$ cannot be interrupted within Δ. Hence, if the task does not finish its execution on the boundary of Δ, the task might miss its deadline.*

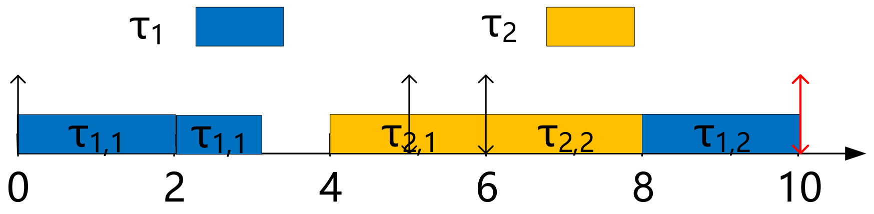


Figure 5.6. One possible execution sequence of the task set $\tau_1 = (5,3), \tau_2 = (6,2)$ under $\Delta = 2$ online *RWS*.

Moreover, in online *RWS*, $B_{i,j}(t)$ would be set as $\left\lfloor \frac{B_{i,j}(t)}{\Delta} \right\rfloor \Delta$.

Lemma 5.2.1. *In online* RWS, *only one job can be protected at any time instant t_0.*

Proof. Assuming that a job $\tau_{i,j}$ is protected at the time instant t_0 as shown in Figure 5.7, $d_{i,j}$ is the deadline of $\tau_{i,j}$, i.e.,

$$d_{i,j} - t_0 = r_{i,t_0}. \tag{5.15}$$

let

$$\mathcal{D}_i = Adbf(d_{i,j}) - \sum_{k,l:d_{k,l} \le d_{i,j}} M_{k,l}(t) - C_i. \tag{5.16}$$

Then $\tau_{i,j}$'s budget is

$$B_{i,j}(t_0) = d_{i,j} - t_0 - \mathcal{D}_i \ge r_{i,t_0}. \tag{5.17}$$

From Equation (5.15) and Equation (5.17), we can get

$$\mathcal{D}_i = 0. \tag{5.18}$$

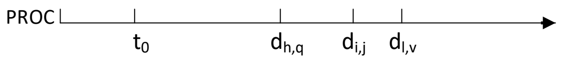


Figure 5.7. Demonstration of time instants in the proof for Lemma III.1.

We prove by contradiction. after considering two cases:

Case 1: At time instant t_0, there exists another protected job $\tau_{h,q}$ with the deadline $d_{h,q}$, which has a higher priority than $\tau_{i,j}$.

If $\tau_{h,q}$ with higher priority is protected at time instant t_0, then $r_{h,t_0} > 0$. Therefore $\mathcal{D}_i$ should be greater than zero in Equation (5.16). It contradicts Equation (5.18).

Hence, $\tau_{h,q}$ does not exist.

Case 2: At time instant t_0, there exists another protected job $\tau_{l,v}$ with the deadline $d_{l,v}$ which has a lower priority job than $\tau_{i,j}$.

If $\tau_{l,v}$ is protected, then its remaining workload r_{l,t_0} should be equal to its remaining time $d_{l,v} - t_0$; its budget should satisfy:

$$\begin{aligned} B_{l,v}(t_0) = d_{l,v} - t_0 - \mathcal{D}_l \geq r_{l,t_0} \\ \mathcal{D}_l = 0. \end{aligned} \tag{5.19}$$

However, $\tau_{i,j}$ has a higher priority than $\tau_{l,v}$, which is protected at time t_0, According to Equation (5.16), D_l is greater than zero. It contradicts to Equation (5.19). So $\tau_{l,v}$ does not exist.

□

Theorem 5.2.2. *A task set is schedulable under Online* RWS *if and only if the following condition (Equation (5.1)) holds:*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

Proof. We prove the “if" part by contrapositive reasoning. If the utilization of a task set is greater than 1, no scheduler can schedule such task set. The proof is completed by Liu and Layland (1973b).

We prove the “only if" part by contrapositive, i.e., online *RWS* is not schedulable $\Longrightarrow$ $U > 1$. Suppose $\tau_{k,l}$ misses its deadline at d_k as shown in Figure 5.8. t_r is $\tau_{k,l}$'s release time. t_{-1} is the last idle instant[5]. t_x is the first time instant where $B_{k,t_x} < r_{k,t_x}$, which indicates that the job $\tau_{k,l}$ would miss its deadline under online *RWS*.

[5]The idle instant is that each task whose next job has a deadline at or before d_k either has no ready job or has just release a job before t_r, and no job with a deadline after d_k executes in (t_{-1}, d_k).

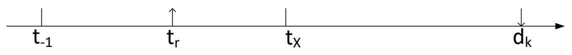


Figure 5.8. Timeline set up for proof of Theorem III.2.

If $t_x > t_r$ and Equation (5.16):

$$B_{k,l}(t_x) = d_k - t_x - \mathcal{D}_k < r_{k,t_x} \tag{5.20}$$

Since t_x is the first time instant that $B_{k,t_x} < r_{k,t_x}$, the release time t_r which is before time t_x must satisfy:

$$B_{k,l}(t_r) = d_k - t_r - \mathcal{D}_k \geq r_{k,t_r} \tag{5.21}$$

Then, we consider three cases at t_r:

Case 1: $hep(\tau_{i,j})$ jobs is chosen to run in the next slot.

$$B' = d_k - (t_r + \Delta) - (\mathcal{D}_k - \Delta) = B_{k,l}(t_r) \geq r_{k,t_r} \tag{5.22}$$

Case 2: $\tau_{k,l}$ is chosen to run in the next slot.

$$B'' = d_k - (t_r + \Delta) - \mathcal{D}_k = B_{k,l}(t_r) - \Delta \geq r_{k,t_r} - \Delta \tag{5.23}$$

Case 3: A lower-priority job is chosen to run in the next slot.

$$B''' = d_k - (t_r + \Delta) - \mathcal{D}_k \geq r_{k,t_r+\Delta} = r_{k,t_r} \tag{5.24}$$

No matter which case is chosen by *RWS*, if $B_{k,l}(t_r) \geq r_{k,t_r}$ holds at time t_r, it will always holds when $\tau_{k,l}$ is in the ready queue. Therefore, t_r should be the first time instant that $B_{k,l}(t_r) < r_{k,t_x}$, where

$$\begin{aligned} t_x &= t_r; \\ B_{k,l}(t_r) &< C_i \end{aligned} \tag{5.25}$$

There is no priority inversion during $(t_r, d_k]$. Additionally, since *RWS* is a work-conserving scheduling algorithm, the idle and busy period in *RWS* is the same as the one in the EDF scheduling algorithm.

Let t_{-1} be the last idle instant, from t_{-1} to t_r, there is no job in $lp(\tau_{k,l})$ executing. Otherwise

$$\forall \tau_{i,q} \in hep(\tau_{k,l}), B_{i,q}(t_0) > r_{k,t_0} \tag{5.26}$$

Which indicates that idle slots exist after time instant t_{-1} if no job in $lp(\tau_{k,l})$ executed. It contradicts with the concept of busy period. Hence, no priority inversion is allowed in time period $(t_{-1}, d_k]$; only $hep(\tau_{k,l})$ jobs can be executed in this period.

Thus, the processor demand in $(t_{-1}, d_k]$ must have exceeded the supply, i.e.,

$$\sum_{j=1}^{N} \lfloor \frac{d_k - t_{-1}}{T_j} \rfloor \cdot C_j > d_k - t_{-1} \tag{5.27}$$

$$\Rightarrow \sum_{j=1}^{N} \frac{d_k - t_{-1}}{T_j} \cdot C_j > d_k - t_{-1}$$

cancelling $d_k - t_{-1}$ on both sides gives us $\sum_{j=1}^{n} C_i/T_i > 1$, which violates Equation (5.1).

□

5.2.4. Empty task. Online RWS is a work-conserving scheduling algorithm, which only improves the randomness in the busy period. For the idle period, the execution probabilities of jobs are 0, so the slot entropy is 0. In order to increase the *randomness* of the execution sequence under online RWS and distribute the task more evenly in one hyper-period, the idle period would be viewed as an empty task τ_{emp}, and its parameters (T_{emp}, C_{emp}) are calculated by:

$$\begin{aligned} C_{emp} &\leq L \times (1 - U); \\ T &= L \end{aligned} \tag{5.28}$$

We should guarantee the schedulability after adding the empty task. According to Theorem 5.2.1, the total utilization of the new task set should less than 1. Therefore, C_{emp} should follow Equation (5.28). Besides, this empty task has the lowest priority; so, it cannot preempt a task in a protected or a urgent state. We provide an example below.

Example 5.2.5. *We continue to consider the task set in Example 5.1.1 and add an empty task* $\tau_{emp} = (2, 6)$ *to it. The possibility assignment with the empty task (PROC_R) for the first six slots is listed at Table 5.5 while the one without the empty task (PROC) is between the parentheses. Thus, according to Equation (5.4), in the first hyper-period, the schedule entropy with the empty task* $H_{emp}(t, \mathcal{S})$ *in Table 5.5 is 5.53 while the one without empty task is 0.97. However, the maximum schedule entropy is* $\log_2 54 \approx 5.7$ *with Equation (5.9). Though we add a empty task to the taskset, we design online RWS with the greedy strategy, the schedule entropy of online RWS will still fall into a local optima. In future work, we will address this issue and optimize the probability assignment rules so that the schedule entropy will be closer to the maximum value.*

Table 5.5. PROC_R Slot entropy of a task set.

Task	s_1	s_2	s_3	s_4	s_5	s_6
τ_1	2/7(2/5)	15/37(0)	1(0)	1/3(1)	1/2(0)	0(0)
τ_2	3/7(3/5)	10/37(1)	0(1)	0(0)	0(0)	0(0)
τ_{emp}	2/7	12/37	0	2/3	1/2	1
$H_{emp}(t, \mathcal{S})$	1.56	2.06	0	0.91	1	0
H(t, $\mathcal{S}$)	0.97	0	0	0	0	0

5.3. EVALUATION

In this section, we first introduce the evaluation setup, then we show the scheduling entropy in different settings, and conclude with an analysis of the experiment results.

5.3.1. Evaluation Setup. To evaluate the schedule entropy under varying settings, we randomly generated the task set as follows:

- The number of tasks n are chosen from $\{5, 7, 9, 11, 13, 15\}$.
- The utilization U of the task set is $\{0.5, 0.55, 0.6, ..., 0.9\}$.
- The task's utilization is generated by UUnifastDiscard Emberson *et al.* (2010).
- The task's period and WCET are integers.
- The task's period is 2^i, where $i \in [8, 12]$.
- For each pair of (U, n), we generate 20 different task sets.
- To ensure the schedulability, we set $\Delta = 1$.

Under our settings, we have 1200 random task sets which are in 60 groups to evaluate the scheduling entropy.

5.3.2. Results. We set the task released without jitter, and implemented online RWS and Yoon *et al.*'s TaskShuffler Yoon *et al.* (2016) which is the base case shown in Figure 5.12. Online RWS experiment results are shown in Figure 5.9-5.11 for the different number of tasks, where the x-axis is the utilization and y-axis is the schedule entropy. In online RWS, the average schedule entropy increases with the increase of the task's number, Moreover, online RWS performs well when the workload is heavy, the average value of schedule entropy may large than the schedule entropy with small utilization.

We compare our experimental result with the base case in Figure 5.13. The x-axis is the utilization, while the y-axis is the ratio of the value of the schedule entropy between online RWS and the base case where we combined the group by the utilization. As Figure 5.13 shows that our work has better randomness than Yoon *et al.*'s. The average ratio is 527 and the best one is 3606. Additionally, when the utilization increases, our scheduling entropy increases while TaskShuffler's decreases. There are three main reasons:

1. **The inversion budget.** When they calculate the inversion budget V_i[6], they use the worst-case maximum inversion budget which is pessimistic, especially for lower-priority tasks. Thus, when TaskShuffler selects the candidate job set, it will miss some lower-priority tasks. On the contrary, for our online RWS, we consider a precise semi-inversion budget so that no potential candidate job will be missed when conducting the selection.

2. **The probability assignment**. In TaskShuffler, they assign equal probabilities to the candidate jobs so as to gain the maximum scheduling entropy. But during the experiment, we find that assigning equal probabilities to candidate jobs will cause an unnecessary pessimism, which shrinks the candidate job set in the later scheduling slot. Hence, when we design online RWS, we avoid the side-effect brought by previous slots through assigning the probabilities unevenly.

3. **The scheduling point**. The scheduling entropy depends on the probability assignment and the number of scheduling points. In the TaskShuffler, the scheduling point is decided by the inversion budget. Thus, if a task utilization is small, i.e., V_i can

[6] $V_i = d_i - (e_i + \sum_{\tau_j \in hp(\tau_i)}(\lceil \frac{d_i}{T_j} \rceil + 1)e_j)$, where $hp(\tau_i)$ is the tasks with higher priority than τ_i under fixed priority.

be large, and the length of the time slot, which is decided by V_i, might be long. Considering a hyper-period, if the system has only few scheduling points, even with the perfect probability assignment, the scheduling entropy cannot be high.

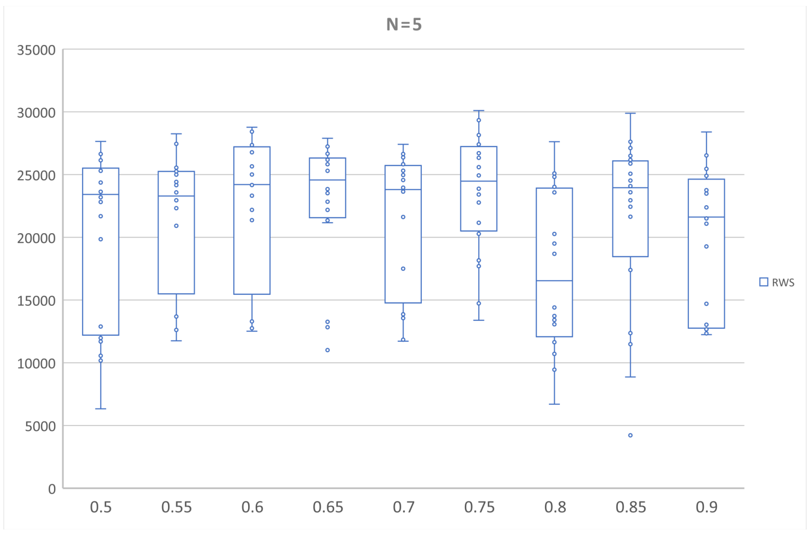


Figure 5.9. Task sets with task number n=5 are scheduled under RWS, the average value of schedule entropies are large than 10000.

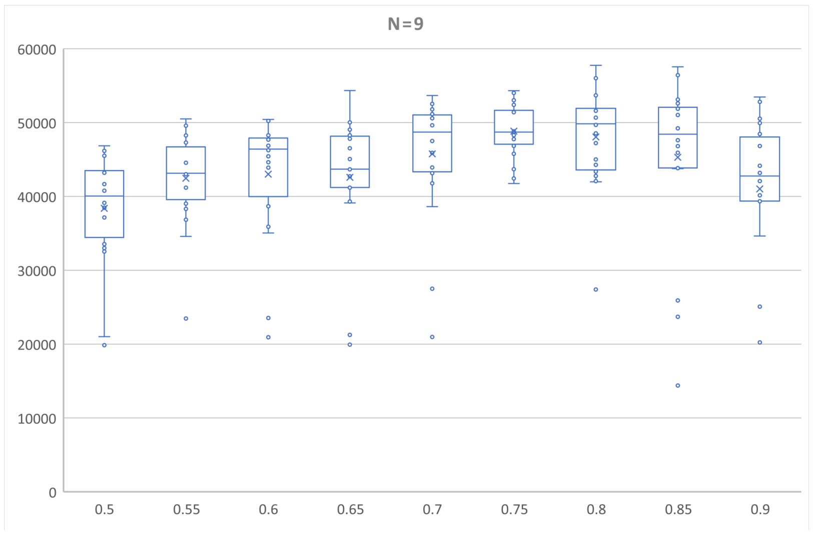


Figure 5.10. Task sets with task number n=9 are scheduled under RWS, the average value of schedule entropies are large than 40000.

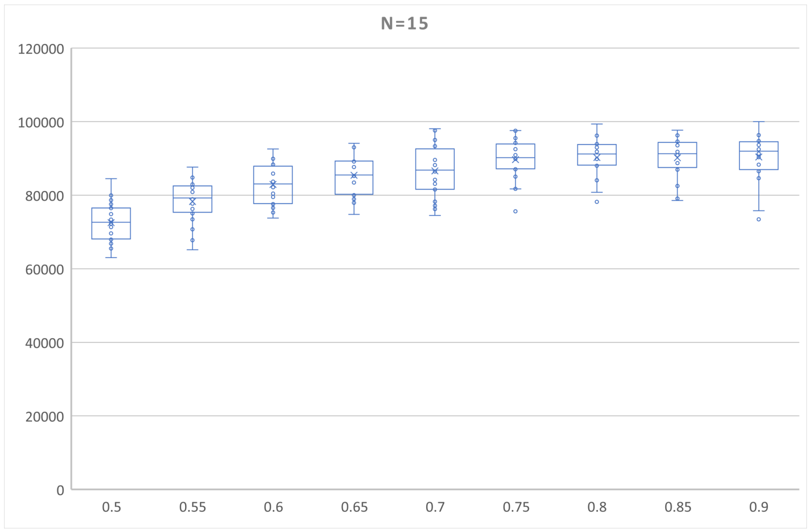


Figure 5.11. Task sets with task number n=15 are scheduled under RWS, the average value of schedule entropies are larger than 70000.

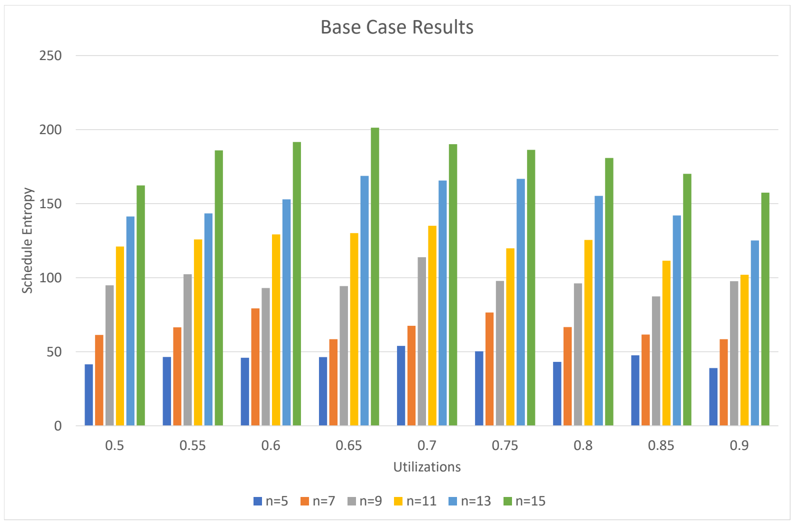


Figure 5.12. The schedule entropy for randomly generated task sets scheduled under TaskShufflerYoon *et al.* (2016). Each group represents the schedule entropy for the different number of tasks with same utilization.

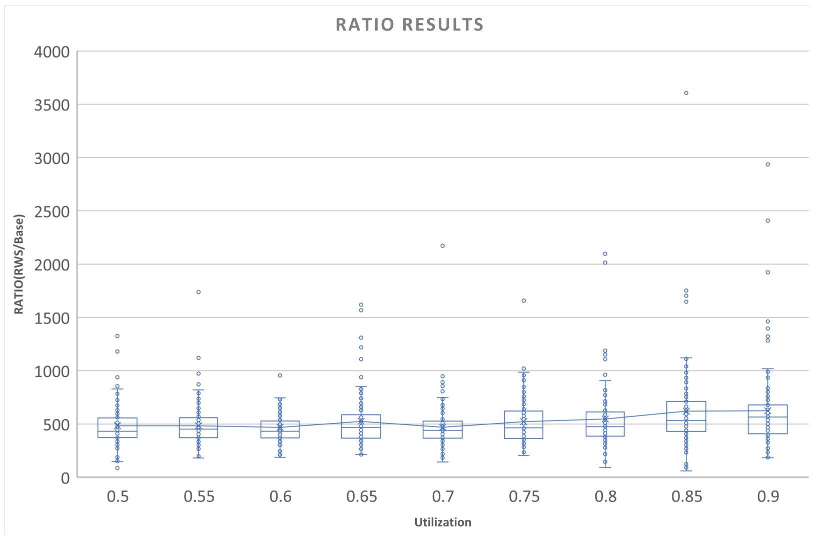


Figure 5.13. Ratio results of RWS and TaskShufflerYoon *et al.* (2016) for the randomly generated task set with different utilization. RWS outperforms TaskShuffler by about 500 times on average under all settings.

6. CONCLUSION

This thesis mainly focuses on two cache problems in real-time systems: CRPD and cache information leakage. For the CRPD part, we first integrate the CRPD into GEDF schedulability test and present different methods to bound the CRPD under GEDF. Specifically, the condensed multiset approach leverages the ECB-union multiset approach and UCB-union multiset approach so as to provide a tighter upper bound for the CRPD. Both theoretical analysis and the simulation of results demonstrate the performance of the proposed method[1].

As the predictability of a real-time system potentially leads to scheduling information leakage, which can be utilized by attackers to launch a successful cache attack, we propose the offline and online RWS randomized scheduling algorithms that guarantee schedulability. By assigning probability to each job at every scheduling point, the RWS scheduling algorithm breaks the execution pattern and increases the randomness of the execution sequence. This can be adapted to different devices and systems. Scheduling entropy is applied to measure randomness. According to our evaluation, online RWS can handle various utilization workload settings[2].

Future Direction: For integrating CRPD into GEDF, we first aim to give a more precise method to calculate the total number of preemptions so that we canobtain a tighter bound of CRPD in schedulability analysis. Second, offer a more general approach that could be applied into different cache models to bound the CRPD.

[1]This work was published on ICESS 2017 (Proceedings of the 14th IEEE International Conference On Embedded Software and Systems) Zhang *et al.* (2017)

[2]This work was published on RTAS 2018 (Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium) Zhang *et al.* (2018)

For the RWS, we will optimize the probability assignment rules that can predict and minimize side-effects that are caused by the current decision. Moreover, we plan to improve the flexibility of online RWS through relaxing the limitations of a fixed and small segment length (Δ) and take context switch delay Lunniss *et al.* (2013); Zhang *et al.* (2017) into consideration.

REFERENCES

Abdi, F., Hasan, M., Mohan, S., Agarwal, D., and Caccamo, M., ‘Resecure: A restart-based security protocol for tightly actuated hard real-time systems,’ 2016.

Altmeyer, S., Davis, R. I., and Maiza, C., ‘Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems,’ in ‘IEEE 32nd Real-Time Systems Symposium (RTSS),’ IEEE, 2011 pp. 261–271.

Altmeyer, S., Davis, R. I., and Maiza, C., ‘Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems,’ Real-Time Systems, 2012, **48**(5), pp. 499–526.

Baker, T. P., ‘Multiprocessor EDF and deadline monotonic schedulability analysis,’ in ‘IEEE 24th Real-Time Systems Symposium (RTSS),’ IEEE, 2003 pp. 120–129.

Baker, T. P., ‘An analysis of EDF schedulability on a multiprocessor,’ IEEE transactions on parallel and distributed systems, 2005, **16**(8), pp. 760–768.

Baker, T. P. and Shaw, A., ‘The cyclic executive model and ada,’ Real-Time Systems, 1989, **1**(1), pp. 7–25.

Baruah, S., ‘Techniques for multiprocessor global schedulability analysis,’ in ‘IEEE 28th International Real-Time Systems Symposium (RTSS),’ IEEE, 2007 pp. 119–128.

Baruah, S. K., Mok, A. K., and Rosier, L. E., ‘Preemptively scheduling hard-real-time sporadic tasks on one processor,’ in ‘Real-Time Systems Symposium,’ IEEE, 1990 pp. 182–190.

Bauer, A., Jaulmes, E., Lomné, V., Prouff, E., and Roche, T., ‘Side-channel attack against rsa key generation algorithms,’ in ‘International Workshop on Cryptographic Hardware and Embedded Systems,’ Springer, 2014 pp. 223–241.

Bertogna, M. and Cirinei, M., ‘Response-time analysis for globally scheduled symmetric multiprocessor platforms,’ in ‘IEEE 28th International Real-Time Systems Symposium (RTSS),’ IEEE, 2007 pp. 149–160.

Bini, E. and Buttazzo, G. C., ‘Measuring the performance of schedulability tests,’ Real-Time Systems, 2005, **30**(1), pp. 129–154.

Brucker, P., ‘Multiprocessor tasks,’ in ‘Scheduling Algorithms,’ pp. 298–320, 1998.

Buttazzo, G., *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24, 2011.

Chen, C.-Y., Ghassami, A., Nagy, S., Yoon, M.-K., Mohan, S., Kiyavash, N., Bobba, R. B., and Pellizzoni, R., 'Schedule-based side-channel attack in fixed-priority real-time systems,' Technical report, 2015.

Davis, R. I. and Burns, A., 'Resource sharing in hierarchical fixed priority pre-emptive systems,' in 'RIEEE International real-Time Systems Symposium (RTSS),' IEEE, 2006 pp. 257–270.

Emberson, P., Stafford, R., and Davis, R. I., 'Techniques for the synthesis of multiprocessor tasksets,' in 'proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010),' 2010 pp. 6–11.

Ferdinand, C. and Wilhelm, R., 'Efficient and precise cache behavior prediction for real-time systems,' Real-Time Systems, 1999, **17**(2-3), pp. 131–181.

Goossens, J., Funk, S., and Baruah, S., 'Priority-driven scheduling of periodic task systems on multiprocessors,' Real-time systems, 2003, **25**(2), pp. 187–205.

Gruss, D., Maurice, C., Wagner, K., and Mangard, S., 'Flush+flush: A fast and stealthy cache attack,' in 'Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721,' Springer-Verlag, ISBN 978-3-319-40666-4, 2016 pp. 279–299.

Gujarati, A., Nasri, M., and Brandenburg, B. B., 'Lower-bounding the mttf for systems with (m, k) constraints and iid iteration failure probabilities,' 2017.

Gülmezoğlu, B., Inci, M. S., Irazoqui, G., Eisenbarth, T., and Sunar, B., 'A faster and more realistic flush+ reload attack on aes,' in 'International Workshop on Constructive Side-Channel Analysis and Secure Design,' Springer, 2015 pp. 111–126.

Ju, L., Chakraborty, S., and Roychoudhury, A., 'Accounting for cache-related preemption delay in dynamic priority schedulability analysis,' in 'Design, Automation & Test in Europe Conference & Exhibition,' IEEE, 2007 pp. 1–6.

Kim, H., De Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R., 'Bounding memory interference delay in cots-based multi-core systems,' in 'IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS),' IEEE, 2014 pp. 145–154.

Lee, C.-G., Hahn, H., Seo, Y.-M., Min, S. L., Ha, R., Hong, S., Park, C. Y., Lee, M., and Kim, C. S., 'Analysis of cache-related preemption delay in fixed-priority preemptive scheduling,' IEEE transactions on computers, 1998, **47**(6), pp. 700–713.

Lee, E. A., 'Cyber physical systems: Design challenges,' in 'IEEE 11th International Symposium on Object Oriented Real-Time Distributed Computing (ISORC),' IEEE, 2008 pp. 363–369.

Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., and Mangard, S., 'Armageddon: Cache attacks on mobile devices,' in '25th USENIX Security Symposium (USENIX Security 16),' USENIX Association, ISBN 978-1-931971-32-4, 2016 pp. 549–564.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M., 'Meltdown,' ArXiv e-prints, 2018.

Liptay, J. S., 'Structural aspects of the system/360 model 85, ii: The cache,' IBM Systems Journal, 1968, **7**(1), pp. 15–21.

Liu, C. L. and Layland, J. W., 'Scheduling algorithms for multiprogramming in a hard-real-time environment,' Journal of the ACM (JACM), 1973a, **20**(1), pp. 46–61.

Liu, C. L. and Layland, J. W., 'Scheduling algorithms for multiprogramming in a hard-real-time environment,' Journal of the ACM (JACM), 1973b, **20**(1), pp. 46–61.

Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B., 'Last-level cache side-channel attacks are practical,' in 'IEEE Symposium on Security and Privacy (SP),' 2015 pp. 605–622.

Lunniss, W., Altmeyer, S., Maiza, C., and Davis, R. I., 'Integrating cache related pre-emption delay analysis into edf scheduling,' in 'IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS),' IEEE, 2013 pp. 75–84.

Mohan, S., Yoon, M. K., Pellizzoni, R., and Bobba, R., 'Real-time systems security through scheduler constraints,' in 'Proceeding of the 26th Euromicro Conference on Real-Time Systems (ECRTS),,' IEEE, 2014 pp. 129–140.

Negi, H. S., Mitra, T., and Roychoudhury, A., 'Accurate estimation of cache-related pre-emption delay,' in 'Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (ICHCSS),' ACM, 2003 pp. 201–206.

Neumann, P. G., 'Denial-of-service attacks,' Communications of the ACM, 2000, **43**(4), pp. 136–136.

Omara, F. A. and Arafa, M. M., 'Genetic algorithms for task scheduling problem,' J. Parallel Distrib. Comput., 2010, **70**(1), pp. 13–22, ISSN 0743-7315.

Pellizzoni, R. and Caccamo, M., 'Toward the predictable integration of real-time cots based systems,' in 'Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International,' IEEE, 2007 pp. 73–82.

Pellizzoni, R., Paryab, N., Yoon, M.-K., Bak, S., Mohan, S., and Bobba, R. B., 'A generalized model for preventing information leakage in hard real-time systems,' in 'Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE,' IEEE, 2015 pp. 271–282.

Prete, A., Xhani, O., Buttazzo, G., and Bertogna, M., ‘Effects of real-time scheduling on cache performance and worst case execution times,’ ????

Puaut, I., Arnaud, A., and Decotigny, D., ‘Performance analysis of static cache locking in multitasking hard real-time systems,’ Publication interne- IRISA, 2003.

Rivest, R. L., Shamir, A., and Adleman, L., ‘A method for obtaining digital signatures and public-key cryptosystems,’ Communications of the ACM, 1978, **21**(2), pp. 120–126.

Sampigethaya, K., Poovendran, R., and Bushnell, L., ‘Secure operation, control, and maintenance of future e-enabled airplanes,’ Proceedings of the IEEE, 2008, **96**(12), pp. 1992–2007, ISSN 0018-9219, doi:10.1109/JPROC.2008.2006123.

Sebek, F., ‘Measuring cache related pre-emption delay on a multiprocessor real-time system,’ Memory, 2001, **1024**, p. 66MHz.

Shannon, C. E., ‘A mathematical theory of communication,’ Bell system technical journal, 1948, **27**(3), pp. 379–423.

Shaout, A. and McGirr, K., ‘Real-time systems in automotive applications: Vehicle stability control,’ Electrical Engineering Research, 2013, **1**(4), pp. 83–95.

Son, J. *et al.*, ‘Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems,’ in ‘Information Assurance Workshop, 2006 IEEE,’ IEEE, 2006 pp. 361–368.

Stankovic, J. A. and Ramamritham, K., ‘What is predictability for real-time systems?’ Real-Time Systems, 1990, **2**(4), pp. 247–254.

Staschulat, J., Schliecker, S., and Ernst, R., ‘Scheduling analysis of real-time systems with precise modeling of cache related preemption delay,’ in ‘Euromicro 17th Conference on Real-Time Systems (ECRTS),’ IEEE, 2005 pp. 41–48.

Sun, Y. and Lipari, G., ‘Response time analysis with limited carry-in for global earliest deadline first scheduling,’ in ‘IEEE Real-Time Systems Symposium (RTSS),’ IEEE, 2015 pp. 130–140.

Tao, F., LaiLi, Y., Xu, L., and Zhang, L., ‘Fc-paco-rm: A parallel method for service composition optimal-selection in cloud manufacturing system,’ IEEE Transactions on Industrial Informatics, 2013, **9**(4), pp. 2023–2033, ISSN 1551-3203.

Yarom, Y. and Falkner, K., ‘Flush+reload: A high resolution, low noise, l3 cache side-channel attack,’ in ‘23rd USENIX Security Symposium (USENIX Security 14),’ USENIX Association, ISBN 978-1-931971-15-7, 2014 pp. 719–732.

Yoon, M. K., Mohan, S., Chen, C. Y., and Sha, L., ‘Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems,’ in ‘2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS),’ 2016 pp. 1–12, doi:10.1109/RTAS.2016.7461362.

Zhang, Y., Guo, Z., Wang, L., Xiong, H., and Zhang, Z., 'Integrating cache-related preemption delay into gedf analysis for multiprocessor scheduling with on-chip cache,' in 'Trustcom/BigDataSE/ICESS, 2017 IEEE,' IEEE, 2017 pp. 815–822.

Zhang, Y., Wang, L., Jiang, W., and Guo, Z., 'Work-in-progress: Rws-a roulette wheel scheduler for preventing execution pattern leakage,' in '2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS),' IEEE, 2018 pp. 93–96.

VITA

The author, Ying Zhang, was born in China. She developed a passion for technology early in life and received her bachelors degree in software engneerning from China, in May 2016. After her bachelor's she enrolled in Missouri University of Science and Technology for graduate studies. During her time at Missouri S&T, the author worked as a Graduate Research Assistant under Dr. Zhishan Guo and Dr. George Markowsky, from August 2016 to December 2018.

In partial fulfillment of the requirements for the Master of Science in Computer Science degree from Missouri University of Science and Technology, this thesis is the culmination of that degree. The author obtained her Master of Science in Computer Science from Missouri University of Science and Technology in December 2018.