

博士学位论文

信息物理系统中的实时调度问题研究



研究生：张天宇

导师：邓庆绪 教授

副导师：Prof. Xiaobo Sharon Hu

东北大学

二〇一八年一月

分类号_____

密级_____

UDC _____

学 位 论 文

信息物理系统中的实时调度问题研究

作 者 姓 名：张天宇

指 导 教 师：邓庆绪 教授

Prof. Xiaobo Sharon Hu

东北大学计算机科学与工程学院

申请学位级别：博 士 学 科 类 别：工 学

学科专业名称：计算机系统结构

论文提交日期：2018年1月 论文答辩日期：2018年04月

学位授予日期： 答辩委员会主席：

评 阅 人：

东 北 大 学

2018年1月

A Dissertation in Computer software and theory

Real-Time Task and Packet Scheduling in Cyber-Physical Systems

by

Zhang Tianyu

Supervisor: Prof. Deng Qingxu

Prof. Xiaobo Sharon Hu

Northeastern University

Jan, 2018

独创性声明

本人声明，所提交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者：

日 期：

学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人同意东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

作者和导师同意网上交流的时间为作者获得学位后：

半年 一年 一年半 两年

学位论文作者签名：

导师签名：

签字日期：

签字日期：

信息物理系统中的实时调度问题研究

摘要

近年来,信息物理系统(Cyber-Physical System, CPS)的快速发展同时吸引了来自工业界和学术界的广泛关注,一个信息物理系统是由物理组件以及计算组件紧密结合而构成。许多信息物理系统都需要保证系统中的任务能够严格满足运行的正确性,该正确性不仅包括功能上的正确,同时还包括时间上的正确性,即保证任务能够在规定的时间内正确地执行所有功能的执行。为此,实时调度技术提供了一种方法来决定任务在共享资源上的执行顺序,以此来保证信息物理系统中尽可能多的任务能够满足它们的截止期。本文根据信息物理系统的应用场景不同,分别研究了混合关键性系统、严格周期任务系统以及实时无线传感网络中的任务以及网络包的调度问题,提出了一系列调度技术来提高处理器的实时性能以及实时无线网络中动态应对突发事件的响应和处理能力。具体的,本文的主要贡献点概括如下:

(1) 针对混合关键性系统,我们设计了一种新的应用于EDF-VD调度策略的可调度性分析方法。不同于之前的分析方法分别单独分析每个不同关键性级别下的系统可调度性,我们的方法创新性地将系统的运行时行为作整合性研究,以此获取更加精确的分析结果。大量的实验结果表明,我们的分析方法能够显著的提升EDF-VD算法调度下的混合关键性系统的可调度性,尤其是对拥有大于两个关键性级别的多级关键性系统,提升的效果更加明显。这种显著的可调度性的提升是建立在更高的分析复杂度的基础之上,针对这一问题,我们通过结合本文所提出的分析方法以及之前的研究工作,进一步提出了一种平衡分析精确度和分析复杂度的启发式算法,以此,为系统设计者提供更灵活的调度算法选择空间。

(2) 针对严格周期任务系统,本文首创性地提出一种有效的方法来为每个严格周期任务分配开始执行时间,使得系统中所有的任务可调度且不会发生冲突。本文首先给出一个充分的可调度性判定条件来检查系统是否可调度。为了提高分析的有效性,我们通过研究任务周期之间的关系,提出一种合理的任务选择策略,以提高为每个任务成功分配开始执行时间的可能性。通过大量的随机实验,验证了本文方法的可调度性显著高于现有其他相关工作,并且接近于最优的精确分析策略,并且本文算法的运行效率显著高于最优策略。因此,本文提出的

任务开始执行时间分配策略在精确性和有效性方面取得了优异的平衡。

(3) 针对实时无线传感网络, 本文提出一种新颖的分布式动态包调度框架 D^2 -PaS。 D^2 -PaS支持在线地处理网络中随机出现的紧急事件(本文定义为干涉), 保证当干涉出现时, 所有重要的网络包都能够在截止期内抵达目的节点, 而同时使得其他一些非紧急的包被丢掉的数量最小。作为一种分布式的调度策略, D^2 -PaS让每个网络中的节点在本地分布式地生成调度表, 当干涉出现时, 这种方式能够大大地减少网络中的控制节点(如网关)向每个节点广播的动态调度表相关的信息。作为一种动态策略, D^2 -PaS在网络中的控制节点上部署一个轻量级的丢包算法, 能够在线动态地响应网络中出现的干涉。我们将设计的 D^2 -PaS框架实现在一个真实的多跳实时无线网络测试平台上来验证其可用性。大量的模拟实验进一步确认了 D^2 -PaS的有效性。


(4) 同样针对实时无线传感网络, 本文提出一种完全分布式的包调度框架FD-PaS。FD-PaS能够保证网络中随机出现的每个干涉都能够得到快速的响应, 同时在确保关键包能够准时传输的同时, 其他非紧急的网络包被丢掉的数量最小。为了使得FD-PaS能够被应用于较大的实时无线传感网络中, 本文提出相应的算法以及数据链路层设计来确保网络中每个节点在没有集中式控制节点的帮助下, 依然都能够在线地对出现的干涉作出响应和处理。通过大量的随机模拟和真实测试平台实验, 我们验证了本文提出的FD-PaS调度框架的正确性和有效性。

综上, 本文研究了信息物理系统中针对不同应用场景下的具体任务模型的实时调度问题, 为CPS系统的设计与分析提供了可参考的理论依据与技术方法。

关键词: 信息物理系统; 实时调度; 混合关键性; 严格周期; 无线传感网络

Real-Time Task and Packet Scheduling in Cyber-Physical Systems

Abstract

In recent years, the rapid growth of cyber-physical systems (CPS) attracts the research interests from both industrial and academic communities. A Cyber-Physical System is a system where physical components and computational components are tightly integrated. Tasks in a CPS generally need to be accomplished correctly in terms of not only functionality but also punctuality. Real-time scheduling provides the methodology of determining the task execution order on a shared resource in order to make as many tasks in a CPS as possible to meet their deadlines. In this paper, depending on different applications, we study real-time task and packet scheduling problems in mixed-criticality systems, strictly periodic task systems and real-time wireless networks, respectively. The contributions of this work are as follows. 

(1) For mixed-criticality real-time systems, we develop new schedulability analysis methods for EDF-VD. Different from previous analysis methods that separate the analysis on each individual criticality level, our new analysis looks into system behavior crossing multiple criticality levels to obtain more precisely analysis results. Experiments show that our new analysis method can significantly improve guaranteed schedulability of EDF-VD, especially for systems with more criticality levels. The price paid for improved schedulability is higher analysis complexity, but a combination of our new techniques and previous methods can obtain a good balance between the analysis precision and efficiency.

(2) For strictly periodic task systems, we propose an efficient method to select time slots for strictly periodic tasks to make them be schedulable. Our method provides a sufficient test condition to check the feasibility of a given task set, and returns the start time assignment if it is feasible. By exploring the relations among task periods, we further present a reasonable task selection method to improve the possibility of finding feasible start time configurations. Finally, we conduct experiments with randomly generated workload to evaluate the performance of the proposed method.

(3) For real-time wireless networks (RTWNs), we introduce a novel distributed dynamic packet scheduling framework, referred to D²-PaS. D²-PaS aims to minimize the number of

dropped packets while ensuring that all critical events due to disturbances are handled by their deadlines. D^2 -PaS builds on a number of observations that help reduce the scheduling overhead, and thus is efficient and scalable. Besides extensive simulation, D^2 -PaS has been implemented on an RTWN testbed to validate its applicability on real hardware. Both testbed measurements and simulation results confirm the effectiveness of D^2 -PaS.

(4) For real-time wireless networks (RTWNs), we present a fully distributed packet scheduling framework called FD-PaS. FD-PaS aims to provide guaranteed fast response to unexpected disturbances while dropping a minimum number of packets for meeting the deadlines of all critical tasks. To combat the scalability challenge, FD-PaS incorporates several key advances in both algorithm design and data link layer protocol design to enable individual nodes to make on-line decisions locally without any centralized control. Our extensive simulation and testbed results have validated the correctness of the FD-PaS design and demonstrated its effectiveness in providing fast response for handling disturbances.

In summary, this dissertation studied various real-time task and packet scheduling problems in cyber-physical systems including mixed-criticality systems, strictly periodic task systems and real-time wireless networks. The results of the thesis serve as theoretical foundations as well as provide practical insights for the CPS design.

Keywords: CPS; real-time scheduling; mixed-criticality; strictly periodic; wireless sensor networks

目 录

独创性声明	I
摘 要	II
ABSTRACT	IV
附表清单	IX
插图清单	X
第 1 章 绪 论	1
1.1 研究背景及意义	1
1.2 国内外研究现状	4
1.2.1 混合关键性系统	5
1.2.2 严格周期任务系统	6
1.2.3 实时无线传感网络	7
1.3 本文研究内容与贡献	9
1.4 本文组织结构	11
第 2 章 信息物理系统研究背景	15
2.1 信息物理系统	15
2.1.1 基本概念	15
2.1.2 应用领域	16
2.1.3 CPS研究需求	17
2.1.4 解决方法	18
2.2 实时调度	19
2.2.1 实时调度策略	20
2.2.2 实时分析	21
第 3 章 EDF-VD调度下的混合关键性系统可调度性分析	27
3.1 预备知识	28
3.1.1 混合关键性任务系统	28
3.1.2 EDF-VD算法	29
3.1.3 现有的EDF-VD分析方法	30
3.2 新的可调度性分析方法	32
3.2.1 两级关键性系统	32

3.2.2 多级关键性系统	37
3.2.3 复杂度分析	40
3.3 调整虚拟截止期	42
3.4 实验	44
3.5 小结	47
第 4 章 严格周期任务系统的开始时间配置策略	49
4.1 预备知识	50
4.1.1 系统模型	50
4.1.2 现有分析方法	51
4.2 开始执行时间配置方法	53
4.2.1 可调度性测试算法	53
4.2.2 任务选择策略	57
4.2.3 复杂度分析	60
4.3 实验	60
4.4 小结	62
第 5 章 处理实时无线传感网络中干涉的分布式动态包调度策略	63
5.1 预备知识	65
5.1.1 系统模型	65
5.1.2 D ² -PaS 框架总览	67
5.2 本地调度表的生成	71
5.3 节奏模式下的调度	75
5.3.1 问题定义	75
5.3.2 系统节奏模式的结束时刻选择	76
5.3.3 丢包算法	79
5.4 处理并发干涉	81
5.5 测试平台实现	85
5.6 模拟实验	86
5.6.1 实验配置	86
5.6.2 处理单个干涉的实验	88
5.6.3 处理并发干涉的实验	89
5.7 小结	89

第 6 章 处理实时无线传感网络中干涉的完全分布式包调度策略	91
6.1 系统模型	92
6.2 FD-PaS 框架概述	94
6.2.1 集中式方法的缺陷	94
6.2.2 FD-PaS 框架总览	96
6.3 传播干涉信息	97
6.4 避免传输冲突	98
6.5 本地动态调度表生成	99
6.5.1 问题定义	99
6.5.2 结束时刻选择	100
6.5.3 动态调度表生成	102
6.6 性能评估	104
6.6.1 模拟实验	105
6.6.2 测试平台实验	107
6.7 小结	109
第 7 章 结 论	111
7.1 本文主要贡献与结论	111
7.2 进一步的工作	112
参考文献	115
致 谢	129
攻博期间发表的论文	131
攻博期间参与的项目	133

附表清单

表 2.1 CPS中信息和物理实体特性比较	15
表 2.2 CPS特性及其应用场景	17
表 2.3 实时调度基本概念	23
表 3.1 混合关键性任务集	29
表 5.1 实时无线传感网络实例中的任务参数	66
表 5.2 重要符号总结	68
表 5.3 V_3 在 $t = 0$ 时刻存储的调度信息表	74
表 6.1 任务参数	95

插图清单

图 1.1 信息物理系统应用示例	1
图 2.1 信息物理系统结构图示	16
图 3.1 EDF调度下的混合关键性任务集	29
图 3.2 EDF-VD调度下的混合关键性任务集	29
图 3.3 EY分析方法示例	31
图 3.4 引理3.1的证明示例	34
图 3.5 引理3.2的证明示例	35
图 3.6 多级关键性系统符号说明	38
图 3.7 需要检查的 (y,z) 的范围说明	41
图 3.8 针对三级关键性系统不同 P 和 O 设置下的实验结果	45
图 3.9 随关键性级别数量变化的分析精确度和效率	46
图 3.10 混合分析方法性能评估	46
图 4.1 严格周期任务示例	50
图 4.2 定理4.1示例	51
图 4.3 τ_3 的开始时间分配	57
图 4.4 任务选择顺序	58
图 4.5 不同 P^n 设置下生成随机严格周期任务集所对应的实验结果	61
图 4.6 运行时间开销对比	62
图 5.1 一个实时无线传感网络实例	66
图 5.2 节奏模型示例	66
图 5.3 静态调度和集中式调度示例	69
图 5.4 网关节点收到干涉报告后的网络执行示例。 t' 表示 V_g 收到节奏事件请求的时刻， t'' 表示 V_g 发送第一跳广播包的时刻。	71
图 5.5 节点 V_3 的调度表片段。上(下)图为系统(V_3)的全局(本地)调度表。	72
图 5.6 SS_j 的最长长度例子	74
图 5.7 三种并发干涉相对位置情况	83
图 5.8 实时无线传感网络测试平台	85
图 5.9 模拟实验结果。(a) 正常利用率 $U^* = 0.5$ 的接受率对比；(b) 不同 R_0 设置下的平均丢包率对比；(c) 正常利用率 $U^* = 0.9$ 的平均丢包率对比。	87
图 5.10 不同参数设置下处理并发干涉的实验结果	90

图 6.1 节奏任务模型示例。上下子图分别表示 τ_0 在正常状态和节奏状态下的
 释放时间和截止期。 94

图 6.2 一个包含7个节点的实时无线传感网络例子 95

图 6.3 实时无线传感网络例子的本地EDF调度表 96

图 6.4 FD-PaS框架的运行模型 96

图 6.5 系统利用率 $U^* = 0.5$ 下系统成功率对比 106

图 6.6 平均丢包率对比 108

图 6.7 在实验平台平台上的时间片信息和频段状态 108

第 1 章 绪 论

1.1 研究背景及意义

一个信息物理系统(CPS)^[1]是由物理组件以及计算组件紧密结合而构成。物理组件通常通过部署在环境中的传感器进行实时的监测，然后监测的信息再经由通讯设施传输给计算组件。在收到环境信息后，计算单元会作进一步的信息处理以及决策，之后这些信息又会经由通讯设施传输给物理组件中的执行器。通过这样一种方式，计算组件可以协调和控制整个物理环境来满足所需的各种监测要求。大部分信息物理系统中的计算组件都是由一个控制系统所构成的，能够很好地管理和控制整个CPS系统中的物理组件部分。近年来，信息物理系统被广泛应用于各个领域，如交通运输^[2]、健康护理^[3]、电网系统^[4,5]、建筑和环境控制^[6]以及工厂自动化^[7]。图1.1展示了一个信息物理系统应用的示例，其通过部署一个无线传感网络，将物理世界与计算单元整合为一个整体。在这个例子中，每个物理设备的状态信息都被一个或多个相应的传感器实时地监测和采集，传感器通过无线传感网络中一系列的中间结点将收集的信息发送给控制器。当控制器收到传感器发来的数据之后，将会生成包含决策信息等的执行指令，这些执行指令再通过无线传感网络中的中间结点发送给执行器。之后执行器会根据收到的执行指令对相应的物理设备进行操作，以提高设备的性能。

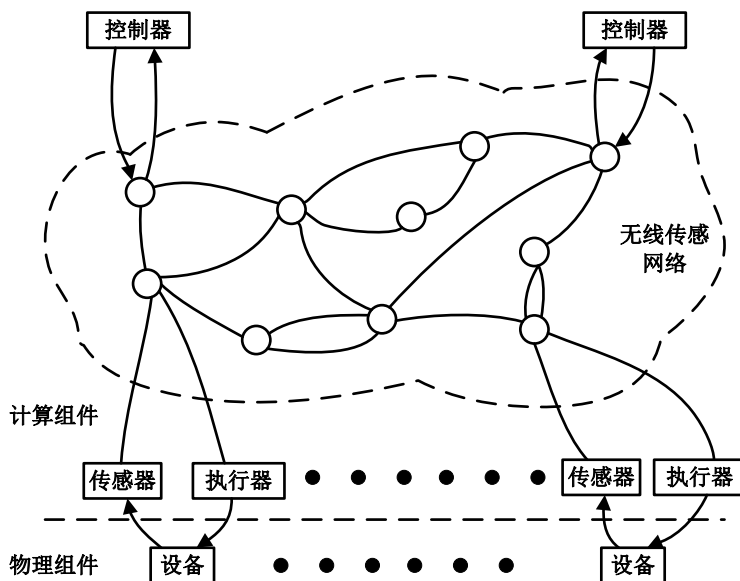


图 1.1 信息物理系统应用示例

Fig. 1.1 A CPS application example

绝大多数的物理信息系统都有一个共同的特性，即时间约束特性。具体来

说, 每个计算任务不仅需要保证功能执行的正确性, 更重要的是需要保证它的执行在有限的时间内完成。任务执行的延迟不仅会导致用户满意度的降低(如视频播放系统), 更有可能带来灾难性的后果(如电子刹车系统)。信息物理系统的这种实时性需求通常表现为系统中的每个任务都要满足一定的截止期约束。CSP中的任务通常需要被反复的重复执行, 在实时调度领域, 一个任务的每一次执行通常被称为任务的一个执行实例。由于信息物理系统中的计算组件(即处理器)需要处理若干不同的实时任务, 因此任务的执行顺序将会对整个系统的性能起到决定性的影响。一般来说, 信息物理系统在设计阶段需要满足两个特性, 即可预测性与可靠性, 具体来说就是必须满足系统中所有任务的时间特性, 而不是快速地执行一个或几个个别的任务。通过在系统中部署某个特定的实时调度算法, CPS的这两种特性就可以得到满足。实时调度提供了一种方法来决定任务在共享资源上的执行顺序, 以此来使得尽可能多的任务满足它们的时间特性。实时调度技术被广泛的应用与信息物理系统的各种不同应用中, 如飞行控制系统^[8]、无线网络控制系统^[9]以及电池管理系统^[10]等。根据部署的应用场景不同, 信息物理系统可以有多种不同的实现形式, 抽象成具体的系统模型包括混合关键性系统(mixed-criticality systems, MCSs)、严格周期任务系统(strictly periodic task systems, SPTSs)以及实时无线传感网络(real-time wireless networks, RTWNs)等。

混合关键性系统的出现得益于现代实时嵌入式系统的一个重要的发展趋势, 即在同一硬件平台中集成具有不同关键性级别的多个应用, 以此来提升资源的有效利用率。这里的关键性定义为系统中的一个应用在发生不同程度的错误情景时所能保证的正确性级别。一个混合关键性系统通常包含两个及多个不同的关键性级别^[1]。例如, 美国联邦航空部门(Federal Aviation Administration, FAA)采用DO-178B标准^[11]来验证航空系统中软件运行的可靠性。DO-178B标准包含五个不同的关键性级别(A-E):

- 灾难级(A): 软件运行错误可能会导致航空灾难。
- 危险级(B): 运行错误会对航空安全带来非常负面的影响。
- 严重级(C): 运行错误比较严重, 但不会产生特别负面的影响。
- 轻微级(D): 运行错误值得留意。
- 无影响(E): 运行错误对飞行几乎无影响。

其他一些相似的标准, 如IEC61508、DO-254以及ISO 26262, 也包含了多个不同的系统关键性级别。相应的, 区别于传统的实时任务只有一个最差执行时间(worst-case execution time, WCET), 在混合关键性系统中, 每个实时任务可能包含

了若干个不同的WCET来对应不同的关键性级别。由于拥有多个关键性级别的混合关键性任务在共享处理器平台上竞争CPU资源，如果不能合理地设计调度算法来为每个任务分配优先级，那么系统中高关键性任务的时间特性很可能不能够得到保证，进而给系统带来灾难性的后果。总的来说，为混合关键性系统设计合理的调度算法需要解决的根本问题是如何处理以下两个相互矛盾的需求。(1) 在共享处理器平台上有效的利用CPU资源；(2) 安全的将不同关键性级别的任务进行隔离以免产生相互影响。近年来，研究者们已经证实在传统实时系统中性能优异的调度算法（如EDF^[12]、RM^[13]）不能直接应用于混合关键性系统。因此，面向混合关键性系统的调度算法设计成为了近年来一个研究热点。

信息物理系统的另一个实现场景是航空电子控制系统。如今的航空电子控制系统需要整合大量的实时传感器、执行器、计算和控制单元，同时还需要具有非常高的安全性及可靠性。随着现代航空飞行器中软硬件复杂度的上升，航空电子工业界提出采用新的航空电子系统结构设计，即整合模块化航空电子系统（Integrated Modular Avionics, IMA）。传统的航空电子系统一直是采用一种称为联合结构（federated architecture）的设计思路，整个系统由大量的设备箱组成，每个设备箱对应一个子系统，每个子系统包含了若干航空电子功能。这种系统结构导致了很低的共享资源利用率以及高昂的设备开销。为此，航空电子工业界近年来已经从联合结构向整合模块化结构IMA转变^[14, 15]。这两种系统设计结构根本上的不同来源于管理计算、通讯和I/O资源方法的区别^[16]。传统的联合结构为每一个航空功能分配独立的专有计算资源，相反，基于IMA的航空电子系统为多个应用分配一个共享的计算、通讯和I/O资源，这样，之前所有独立的子系统都可以部署在一个共享的处理器平台上，以此更有效地利用计算以及通讯资源，同时还能减少能耗以及设备开销。另外一个IMA结构的创新之处体现在硬件平台标准化上，航空应用的开发者们不再需要考虑特定的计算资源以及体系结构，可以专注于功能性软件的开发，这大大缩减了开发成本以及设计周期^[16]。虽然IMA的这种共享处理器资源的设计大大提高了资源利用率，但是由于在航空电子系统中，一个关键程序微小的运行错误可能会导致严重的航空灾难，因此IMA对航空电子系统中的运行控制也有相应的更加严苛的要求。具体来说，一般的实时嵌入式系统通常采用传统的周期性任务模型，但是由于周期性任务模型在系统设计时需要考虑最坏运行情况，在系统运行时任务的实际执行时刻会有较大的抖动，这会大大降低系统的控制性能。相反的，IMA采用的则是严格周期任务模型，即任务的两次连续执行之间的间隔时间是严格等于任务周期的，并且任务的执行是不可被抢占的。

对于传统周期性任务模型，研究者们只需要为系统设计特定的调度算法以及相应的可调度性分析方法，以保证当系统实际运行时不会发生任务错失截止期的情况出现。但是，对于严格周期任务系统，更大的难点在于，系统设计者除了要考虑使用什么样的调度算法和分析方法之外，还要为系统中的每个任务分配一个开始执行时间，来保证在系统运行过程中，所有任务满足截止期的同时，任何两个任务之间不会发生执行时间的重叠。因此，为严格周期实时任务系统设计任务开始时间配置方法也是实时调度领域的一大研究热点。

最后，实时无线传感网络是很多信息物理系统应用的实现基础，例如军事、公民基础设施和工业控制等^[17-19]。一个实时无线网络通常由分布在空间中的若干传感器、执行器、中间结点和一个或多个控制节点，通过无线通讯的方式来共同完成一系列功能性的任务。实时无线传感网络中的应用都必须满足实时性，以此来保证（1）环境信息的实时采集，（2）采集数据的时效性^[20-22]以及（3）对监测到的事件实时的响应。实时无线传感网络的服务质量（Quality of Service, QoS）通常是衡量网络中的实时任务时间约束的满足程度，因此，包的调度问题在实时无线传感网络中扮演了非常重要的角色。虽然这一问题已经在学术界以及工业界研究了较长一段时间，但是随着物联网等应用的大规模发展，尤其是网络规模以及复杂度发生了显著的增长，这大大的增加了网络包调度问题解决的难度。正如一篇近期发表在Proceedings of IEEE上的文章所指出的^[23]，“当前工业无线传感网络的一个重要瓶颈是现有技术不能应对网络规模的快速增长，尤其是当网络采用集中式的控制方法。虽然这种集中式的结构已经被证实对于小规模的网络非常有效，但它无法有效地应用于大规模的网络结构（例如油田中部署的成千上万个传感器设备）”。另一方面，在几乎所有的实时无线传感网络中，都需要应对一些始料不及的突发事件（本文统一称之为干涉），这更加加剧了为实时无线传感网络设计有效包调度算法的难度。

本文将针对以上讨论的信息物理系统中出现的各种亟需解决的问题，结合现有研究成功和基础，围绕实时调度算法的设计与分析开展研究工作，分别解决混合关键性系统、严格周期任务系统和实时无线传感网络中的任务和包调度问题。

1.2 国内外研究现状

2006年，美国发布报告将信息物理系统列为重要的研究项目，并把CPS列为首要的关键信息技术。之后，美国国家科学基金会（National Science Foundation, NSF）授予大量有关于信息物理系统的研究项目以资金支持。许多大学和研究机

构（如加州伯克利大学，范德比尔特大学，孟菲斯大学，密歇根大学，圣母大学，马里兰大学和通用研究发展中心等）都加入到CPS这一研究项目当中^[24, 25]。除此之外，其他国家的研究者们也都意识到信息物理系统研究的重大意义。总之，虽然研究者们已经在信息物理系统的建模、控制、节能、安全和系统设计等方面作出了一些进展，但至今对于CPS的研究仍然处在相当初级的阶段。

1.2.1 混合关键性系统

混合关键性系统是在2007年由Vestal在[26]首次提出并且给出了形式化的定义，文章中他给出了一种固定优先级算法来调度混合关键性系统中的任务以及相应的响应时间分析方法来分析系统的可调度性。这篇文章同时还验证了在传统实时调度中最优的固定优先级分配算法，如速率单调算法（Rate Monotonic, RM）和截止期单调算法（Deadline Monotonic, DM），在混合关键性系统中不再是最优的。Dorin等人在文章[27]中证明了Vestal在[26]中提出的算法在固定优先级可抢占混合关键性系统中是最优的，他们还将算法扩展至任务释放时间可变的模型中并介绍了如何进行敏感性分析。然而，正如Baruah和Vestal在[28]中指出的，如果我们不限制系统使用固定优先级可抢占算法的话，[26]中的算法就不是最优的，而且事实上它的性能跟EDF是无法比拟的。

Baruah等人随后基于一个简化的混合关键性系统模型（系统仅包含有限个释放时间固定的任务实例）提出了一系列根本性的工作，他们证明了即使对于这样一个简化的系统并且所有任务实例都在同一时刻释放，验证系统的可调度性仍然是一个强NP难的问题^[29]。Baruah等人在文献[30]中提出了针对混合关键性系统的启发式算法OCBP（Own Criticality Based Priority），该算法主要适用于仅包含有限数量任务实例的混合关键性系统，它基于Audsley算法^[31]给每个任务实例赋优先级，由于OCBP是一种固定任务实例优先级调度算法，它在具体应用时有很大的局限性。Li和Baruah随后又将OCBP算法扩展到混合关键性偶发任务系统中，并提出一种在线调整优先级的伪多项式时间复杂度算法，虽然该算法的实时性能不错，但由于它高昂的时间开销，限制了它在实际系统中的应用。随后更加高效的多项式时间以及线性时间的算法在文章[32, 33]中被相继提出，但这些算法在对任务优先级计算时仍然有很多悲观的估计，浪费系统资源和降低了系统的利用率。

一种基于EDF（Earliest Deadline First）的调度算法EDF-VD（EDF with Virtual Deadlines）在文章[34]中被Baruah等人首次提出，用于调度混合关键性偶发任务集。EDF-VD算法的主要思想是为每个任务在不同关键级别下赋予不同的虚拟截止期，

这样，一个高关键性的任务可以在系统处于低关键性级别时通过设置一个更早的虚拟截止期，来使得它的执行更早的完成，以便预留出足够多的时间以防系统突然转换到高关键性级别。Baruah等人随后又对EDF-VD算法的可调度性分析作了进一步的改进^[35]。最近，Ekberg和Yi提出了一种针对于EDF-VD进行可调度性分析的方法（本文称为EY算法）^[36, 37]，通过分析每个任务在不同系统关键级别下的需求界限函数（Demand Bound Function, DBF），量化缩小高关键性任务在低关键性级别下虚拟截止期的优缺点，然后设计一种启发式的算法为每个任务找到合适的虚拟截止期配置。EY算法随后又被扩展到了更加一般化的包含多于两个关键性级别的混合关键性系统中。之后，Easwaran针对EY算法进一步提升了可调度分析的精度^[38]，但是该方法仅适用于简化的仅包含高低两种关键性级别的混合关键性系统。除此之外，EDF-VD算法还被应用到多核处理器平台中，一系列全局的以及划分的多核调度算法被相继提出^[39-41]。

尽管距离Vestal在2007年首次提出混合关键性系统调度问题仅仅十年的时间，但却出乎意料地吸引了大量的研究者们关注到这个领域的研究中来，本文仅仅列举了一部分关于混合关键性实时调度的相关工作，更加详尽综合的文献调查读者可以参考这篇关于混合关键性系统的综述^[42]。

1.2.2 严格周期任务系统

当前，针对传统周期性任务模型系统的调度性研究已经非常成熟^[12, 43]，然而针对严格周期不可抢占任务系统的调度问题还没有得到很好的解决，并且也只是从过去的十几年才开始吸引了一部分研究者的关注。这不仅是因为不可抢占这一方面给这一问题所带来的计算复杂度的增长^[44]，更主要的是任务严格周期的这种特性大大增加了问题的难度。

Baruah和Chakraborty分析了不可抢占周期性任务模型的可调度性，并且证实存在伪多项式时间复杂度的近似算法来调度不可抢占系统^[44]。Buttazzo和Cervin使用不可抢占任务模型来降低任务执行的抖动^[45]，在文章^[46]中，George等人针对不可抢占系统提出一种综合的可调度性分析方法。但是这些工作仅适用不可抢占的传统周期性模型，并没有考虑任务的严格周期这种特性，在传统周期性模型中任务的两次实例的开始执行时间是可以改变的，但是在严格周期任务模型中，任务的开始执行时间是一个常数，一旦确定了就不能再发生改变。

在文章^[47, 48]中，Korst等人首先证明了判断不可抢占严格周期任务系统是否可调度是一个NP完全问题。此外，文章还给出了判断一对严格周期任务是否

可调度的充分必要条件。之后, Kermia和Sorel在[49]中给出了判定一个严格周期任务系统可调度性的必要条件, 但这个判定条件随后被证明是非常悲观的^[50, 51]。Eisenbrand等人提出一种针对严格周期任务模型的可调度性判定条件, 但要求系统中的任务都必须是谐波任务, 即所有任务的周期之间都是可以整除的倍数关系^[52]。在文章[50, 51], Marouf和Sorel利用类似的思想提出一种启发式调度算法, 这种算法的限制在于每个新调度的任务的周期必须是所有已被调度任务周期的整数倍。在最近的一篇文章中, Kermia给出了一个严格周期任务模型的充分可调度性判定条件, 但是并没有给出严格的证明, 此外, 这个判定条件仅仅能够判断是否存在一个具体的调度算法能够调度当前的任务集, 但是并不能给出具体的调度策略, 以及如何为每个任务分配一个可调度的开始执行时间。

1.2.3 实时无线传感网络

无线传感网络近年来在工业界受到了越来越多的关注, 得益于它能够大幅的降低部署成本, 尤其是在恶劣的工业控制环境中。工业控制标准, 例如ISA^[53]、HART^[54]、WINA^[55]以及ZigBee^[56], 积极地推动了无线技术在工业自动化以及工业制造领域的应用。然而, 无线控制技术在工业控制上的应用也面临着独特的挑战, 那就是对实时性的要求。不同于有线控制网络(如控制区域网络^[57])中实时调度技术的成熟, 现在针对无线传感网的实时算法还非常有限, 因此实时无线传感网络近年来收到了来自工业界和学术界的广泛关注^[58]。

由于WirelessHART的一系列优秀的特性, 例如采用集中式的网络管理方式、一系列保证可靠传输的机制以及支持多种路由模式等, 使得WirelessHART在工业过程管理和控制得到了全球范围内的广泛关注和应用^[59, 60]。为了研究和测试实时无线传感网络协议, Sha等人^[61]首先搭建了一个WSAN的实验测试平台, 系统包含一个运行在服务器上的网络控制节点和一个实现在TinyOS^[62]上的网络协议栈。作者们在此实验平台上进行了一系列的实验测试, 例如比较了应用于WirelessHART上的两种路由协议, 源路由和图路由, 研究了可靠性、网络时延和能耗在不同路由协议下的关系。文章[61]还得出了一个结论, 就是图路由算法对于最优化网络时延和能耗非常重要。这一问题在随后的文章[63, 64]中也被研究过, 并且提出了一系列应用于WirelessHART的实时路由算法。

无线网络中的实时通讯技术已经在很多近期的工作中研究过^[65-82], 综述[83]对这些工作给出了非常全面的总结。但是, 所有这些工作并不适用于通常要求多信道通讯、多路径路由以及实时性能分析的工业应用中。

在文章[84-89]中, 学者们研究了针对单跳的实时无线传感网络, 对于多跳的无线网络, 文章[90]提出一种数学上的框架来建模和分析WirelessHART网络的调度问题, 随后很多针对WirelessHART的调度问题研究工作也相继被提出^[91-97], 这些提出的调度策略大体上可以分为两类: 固定优先级调度和动态优先级调度。在固定优先级调度研究中, [94]利用可调度性分析来为每个网络中的任务分配优先级使得所有任务都能够满足截止期约束, 并且给出了最优以及近似最优的优先级分配策略。[92, 93]提出了一系列对网络中实时任务的时延分析, 这些分析能够给出任务端到端(从传感器到执行器)通讯时延的上限以及可调度性的充分判定条件。在动态优先级调度研究中, [95, 97]具有树型结构的WirelessHART网络里任务的动态优先级调度, [91, 96]取消了树型结构的限制, 将研究扩展到一般的WirelessHART网络中。[91]的工作已经证明了WirelessHART网络中的实时传输调度问题是NP难的, 并且通过发现传输冲突在通讯时延以及可调度性分析中扮演的重要作用, 作者提出了一种最优的本地搜索调度算法以及更高效的启发式算法C-LLF (conflict-aware least laxity first) 来进行动态优先级调度。[96]研究了EDF算法在实时无线传感网络中的可调度性分析问题。

在实时无线传感网络中另外一个非常值得关注的问题就是不时出现的突发事件, 本文统称为干涉, 这更加加剧了无线传感网络实时调度问题研究的难度。以上总结的工作都可以总结为针对实时无线传感网络的静态包调度策略, 这种静态调度方式能够支持确定性的实时通讯, 但并不能够用来处理由干涉所引发的动态网络变化, 或者只能线下提前做非常悲观的估计, 这样会大大降低网络带宽利用率。网络中的动态事件可以是由网络提供的资源变化引起的(如节点和链路损坏或无线电等引起的干扰等), 也可以是由网络资源需求改变所引起的(如检测到网络入侵者、突然的温度或者压力变化等), 我们称前者为网络内部干涉, 后者为网络外部干涉。针对处理网络内部干涉, 很多采用集中式控制的方法在近几年被提出^[98-100], 但是针对处理网络外部干涉的研究却相对来说比较少。文章[101]在系统初始化阶段预先存储一定数量的数据链路层调度表, 在网络运行过程中通过选择合适的调度表来应对出现的外部干涉, 但是这种方法或者不能处理任意情况的外部干涉, 或者必须做出非常悲观的近似估计, 并且, 文章中并没有具体讨论这些预存的调度表是怎样计算出来的。文章[102, 103]提供控制决策来允许动态的增加或者删除网络中的某个任务以此来应对外部干涉, 但是它们都没有考虑网络并不能满足所有任务截止期的情况。文章[104]提出一种协议为不时出现的外部干涉任务分配预留的时间片, 并且允许在没有干涉出现的时候, 其他普通任务可以

使用这部分预留时间片进行传输，但是文章并没有讨论当干涉出现时，如何来满足其他普通任务的截止期需求。最近的一项研究实时无线传感网络中外部干涉的工作是Hong在[105]中提出的OLS算法，OLS是一种集中式的框架并且已经被证明在小规模的网络中应对外部干涉非常有效。但是对于大规模的实时无线传感网络（比如超过50个节点的网络），由于OLS是一种基于动态规划的调度算法，它的运行时开销非常大以至于并不能及时地对干涉进行响应。此外，OLS还会由于广播包大小限制的问题，不必要地丢掉很多额外的普通网络包，这大大降低了系统的性能。所有的这些缺点及不足促使本文去研究如何更好地解决实时无线传感网络中的外部干涉处理问题。

1.3 本文研究内容与贡献

本文针对信息物理系统研究领域尚存的不足，围绕实时调度算法的设计与分析，在混合关键性系统、严格周期任务系统以及实时无线传感网络中开展研究工作。本文的研究内容主要有以下几个部分：

(1) EDF-VD调度下的混合关键性系统的可调度性分析研究。分析EDF-VD可调度性的主要挑战来源于系统运行时的关键性级别变化这一行为，通常来说，系统什么时间发生关键性级别转换以及在转换发生时系统处在何种状态（如每个任务当前释放的实例的执行情况）都是不可预测的，所以事先枚举所有的可能性这种方法显然因为高昂的时间复杂度开销而不能实际采用。现在针对EDF-VD算法进行可调度性分析最先进的技术是[36, 37]提出的EY算法。EY的主要思想是通过近似化的计算系统由低关键性级别向高关键性级别转换时所带入的工作量，对系统处在不同很关键性级别时的情况进行分析。但是这种近似的计算是非常不精确的，会导致本来可调度的系统被EY分析为不可调度。针对这一问题，本文提出一种针对EDF-VD算法的新的可调度性分析方法，不同于EY将系统不同关键性级别时的情况单独进行分析，我们的方法整合性地考虑系统从低关键性转换至高关键性级别的整个过程，以此来更加精确的得到系统工作量的上限，从而产生更加精确的分析结果。由于本文的分析方法是以提高时间复杂度为代价的，为了解决算法实用性的问题，我们进一步提出一种混合的分析方法，结合EY和本文的方法，来平衡分析精确度和算法执行效率。

(2) 为严格周期不可抢占任务系统配置每个任务的开始执行时间。针对严格周期任务系统，本文首创性地提出一种有效的方法来为每个严格周期任务分配开始执行时间，使得系统中所有的任务可调度且不会发生冲突。本文首先给出一个

充分的可调度性判定条件来检查系统是否可调度。为了进一步提高分析的有效性，我们通过研究任务周期之间的关系，提出一种合理的任务选择策略，以提高为每个任务成功分配开始执行时间的可能性。通过大量的随机实验，本文方法的可调度性显著高于现有其他相关工作，并且接近于最优的精确分析策略，然而本文算法的运行速度显著高于最优策略。因此，本文提出的任务开始执行时间分配策略在精确性和有效性方面取得了优异的平衡。

(3) 实时无线传感网络中应对干涉的包调度问题研究。针对实时无线传感网络中始料不及出现的干涉，本文提出一种新颖的分布式的动态包调度框架D²-PaS (distributed dynamic packet scheduling framework)。作为一种分布式的方法，D²-PaS让网络中的每一个节点在本地计算自己的调度表，这样可以大幅减少当网络中出现干涉时，控制节点向网络中所有节点广播动态调度表相关的信息。作为一种动态调度方法，当网络中出现干涉事件时，D²-PaS在控制节点上在线动态地生成一段临时的调度表，并将最少的信息（包括干涉的信息以及需要丢掉的普通网络包信息）通过广播包的方式发送给网络中的所有节点。为了保证这样一种分布式动态包调度框架的有效性和高效率，我们设计了一个轻量级（内存使用以及计算能力）的调度器使之能够实际部署在网络中的每个节点之上。此外，为了响应出现的干涉，我们设计一个低时间复杂度的丢包算法来决定在一段临时的动态调度表中，哪些普通包需要被丢掉以及最小化丢包的数量。除了理论方向的成果，我们还在一个真实的实时无线传感网络测试平台上部署了本文提出的D²-PaS包调度框架。

(4) 实时无线传感网络中应对干涉的完全分布式包调度框架。同样针对实时无线传感网络中出现的干涉，本文在前一步工作的基础上，进一步提出一种完全的分布式包调度框架FD-PaS (fully distributed packet scheduling framework)。跟之前设计的D²-PaS框架一样，FD-PaS也利用相同的技术让网络中的每一个节点在本地生成自己的调度表。但是，不同的是，当网络中出现干涉时，FD-PaS不需要依赖网络中的控制节点（例如网关节点），可以以一种完全分布式的方式在本地让节点自己来处理干涉。这种完全分布式的框架是通过利用任务的路由，仅把出现的干涉信息发送给网络中一部分相关的节点来实现的。在这样一种方式中，FD-PaS不再需要依赖控制节点发送广播包来把干涉信息发送给网络中所有的节点，这样可以大大地缩短对干涉事件处理的响应时间。为了保证这种发送干涉信息给部分节点的机制能够正确地运行，我们需要处理一系列的问题。例如，为了避免由于信息不对称，不同节点间生成的调度表可能发生的冲突，我们提出一种多优先级无线包抢占机制MP-MAC来部署在数据链路层，以保证当出现调度表冲

突时，具有高优先级的包能够得到成功的传输。另外，为了保证网络的QoS而丢掉尽可能少的普通网络包，我们形式化了一个包调度问题来为每个节点决定一段临时的动态调度表以响应出现的干涉。我们首先证明了这个包调度问题是强NP难的，然后设计了一种最优的整数线性规划算法（integer linear programming, ILP）以及高效的启发式算法来部署在每个网络节点中。我们还将设计的MP-MAC以及动态调度表计算算法（两者共同构成了FD-PaS）实现了一个真实的实时无线传感网络实验平台上。

1.4 本文组织结构

本文共分为7章，各章具体内容组织如下：

第1章为绪论，主要介绍了本文的研究背景及意义，包括信息物理系统研究背景，各种CPS不同应用场景下的具体任务模型的主要内容和意义，从混合关键性系统、严格周期不可抢占任务系统以及实时无线传感网络三个方面分析了国内外研究现状和尚存在的问题，介绍了本文的主要内容，以及篇章结构。

第2章主要介绍信息物理系统的背景知识，包括基本框架、概念以及理论。然后介绍了实时调度算法的基础知识，包括任务模型，基本概念和技术等。

第3章研究了EDF-VD算法调度下的混合关键性系统的可调度性分析问题，提出一种新的可调度性分析方法，通过整合系统从低关键性转换到高关键性级别整个过程的运行行为，使得分析的结果更加精确。

第4章研究了严格周期不可抢占任务模型的调度问题，通过发现任务周期之间的关系，提出一种启发式算法为每个任务分配开始执行时间，使得系统设计者不仅能够判断系统的可调度性，还能得到具体的调度表。

第5章研究了实时无线传感网络中的包调度问题，主要针对网络中意外出现的干涉事件，提出一种分布式的动态包调度框架D²-PaS，使得当网络中出现干涉时，控制节点能够在线生成一段临时的调度表，并通过广播包发送给所有节点来响应干涉。我们还将D²-PaS实现了一个真实的实验测试平台上以测试其实用性和有效性。

第6章同样研究了实时无线传感网络中处理干涉的包调度问题，提出了一种完全分布式的包调度框架FD-PaS。FD-PaS不必依赖于网络中任何的控制节点，能够让本地的节点自己处理出现的干涉，这不仅使得FD-PaS能够被应用到大规模的实时无线传感网络中，还能够大幅提高对出现干涉的响应时间。同样，我们也在真实的实验测试平台上对FD-PaS进行了具体的实现。

第7章总结全文，并提出了未来研究方向以及可以继续深入研究的内容。

第2章 信息物理系统研究背景

本章首先介绍信息物理系统的背景知识，包括基本概念、应用场景和研究挑战等。之后着重介绍本文研究的实时调度技术的一些基本概念、模型和分类等。

2.1 信息物理系统

信息物理系统（CPS）紧密整合了计算资源（如计算机软硬件）和物理实体（如人、车等）之间的联系。嵌入式系统和网络监测和控制物理世界中的实体，并且通常是一种闭环的控制，物理实体的状态影响计算，反之亦然。在物理世界中，时间是不可逆的并且存在复杂的并行，这些都是现如今的计算机和网络技术中考虑比较少的。在表格2.1中，我们列举了物理实体和计算单元之间一些主要的差异。

2.1.1 基本概念

信息物理系统研究的主要目的是深度整合物理实体和计算实体设计，图2.1展示了信息物理系统的结构图示。显然，信息物理系统并不是传统的嵌入式系统、实时系统、传感网络或者桌面应用，而是具有某些特定特征的复杂系统：

- 紧密整合性。信息物理系统是计算过程和物理过程的紧密整合。
- 物理组件的计算性。物理实体一般都有相应的嵌入式系统运行一些嵌入式软件。
- 计算资源有限性。系统资源，例如处理器计算能力、网络带宽等，通常都是有限的。
- 网络连接性。信息物理系统中的网络通常都是分布式网络，包括有线和无线网络、WLAN、蓝牙、GSM等。此外，系统规模以及设备种类通常都非常大。

表 2.1 CPS中信息和物理实体特性比较
Table 2.1 Comparison of Cyber and Physical properties of CPS

	信息实体	物理实体
保证正确顺序的方式	顺序执行	实时
事件同步方式	同步	异步
时间特性	离散	连续
结构	计算抽象	物理规则

- 时间和空间复杂性。在信息物理系统中，不同的组件可能具有不同的时间和空间颗粒型，并且系统具有严格的空间和实时特性。
- 动态重配置性或重组织性。信息物理系统都是非常复杂的系统，必须具有一定的自适应能力。
- 高度的自动化特性。信息物理系统支持便捷的人机交互，并且应用了一些高级的反馈控制技术，控制都是采用闭环的方式。
- 执行可信赖性。作为非常大规模且复杂的系统，信息物理系统必须具备很高的可靠性及安全性，有些情况下甚至必须经过一定的认证过程。

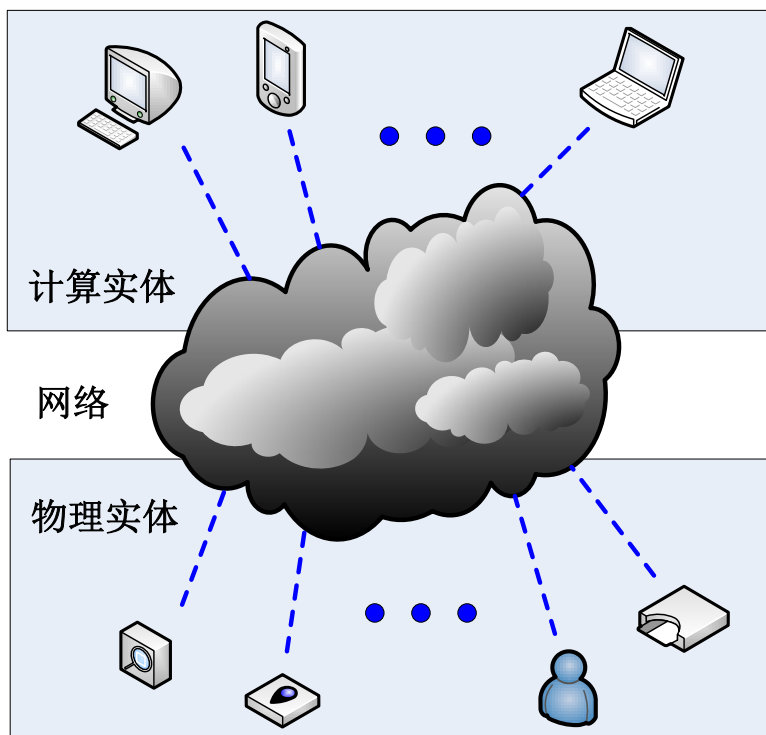


图 2.1 信息物理系统结构图示
Fig. 2.1 Diagrammatic layout for CPSs

2.1.2 应用领域

信息物理系统具备一系列的优点：CPSs是高效且安全的系统，允许不同的实体工作在一起以组成一个复杂的系统，从而具备新的功能性。信息物理技术可被应用的领域非常广泛，包括基础设施控制、高效的安全运输、可替代能源、环境控制、远程监控、医疗设备整合系统、远程医疗、辅助生活照护、社交网络和决策、制造业以及农业控制等。对于其中任意一种应用场景，又会由于具体功能不同产生很多不同的特性，比如像关键性的基础设施控制，包含的设备种类非常之多：用于水资源供给的（存储、处理、运输和分销以及污水处理），电力生产

的，交通运输的，油气田生产的等等。因此，信息物理系统的应用场景非常广泛，表2.2概括了信息物理系统的一些具体应用场景以及相应的对系统的需求。

2.1.3 CPS研究需求

信息物理系统的研究当前还处在一个非常初始的阶段，大都被分解为各个具体子领域独立的研究，如传感器、通讯、网络、控制理论、数学运算、软件工程和计算机科学。例如，系统设计和分析通常利用各种各样的模型形式化方法和工具，但每一种都只能着重关注于某一种具体的特性上而忽略其他特性，以此提高分析可行性。通常，一个具体的模型形式化定义只能较好的定义信息实体或者物理实体，并不能两者兼顾。尽管这种建模和形式化的方法可能能够“分治”地应用于信息物理系统，但也会在验证系统整体正确性、安全性时带来一系列的问题^[106]。接下来，本文简要讨论一下信息物理系统研究的一些需求。

(1) 模型抽象和结构化。为了应对信息物理系统快速的发展和将来大范围的实际部署，必须设计和开发创新型的模型抽象和结构化的方法，使其能够无缝的整合控制、通讯以及计算。例如，在通讯网络中，不同层的接口必须实行统一的标准。一旦这些标准建立，模块化的特性使得可以专注于每一层的开发和设计，各种不同类型不同功能的系统只要具备符合标准的接口类型，就可以实现即插即用，极大的方便了系统的更新换代以应对网络的快速发展。然而，信息物理系统领域现存的科学和工程技术并不支持这类高效的、鲁棒性的模块化设计开发。因此，CPS的研究亟需标准化的模型抽象和结构化方法以实现物理实体和计算实体的充分整合和交互。

(2) 分布式计算和网络控制。网络控制系统的设计与实现在很多方面提出了许多新的挑战，包括给予时间或事件驱动的计算、软件、时间延迟、运行错误、重配置以及支持分布式决策的系统。针对支持实时服务质量的网络协议设计所面

表 2.2 CPS特性及其应用场景
Table 2.2 CPSs Characteristics and their Application Domains

应用场景	系统需求
交通控制、汽车自动化	强大的计算处理能力，例如用来在线决定最佳交通线路的复杂交通控制算法
环境控制	长时间的无人工干预运行能力
航空，国防	精确控制、高安全性以及超强的计算能力
关键基础设施	精确的可靠的控制
健康护理	新一代的分析方法、整合技术

面临的挑战包括平衡控制逻辑设计和实时部署复杂度之间的关系、缩小离散和连续时间系统之间的差距以及提升大规模系统的鲁棒性。此外，还需要创新型的框架、算法、方法和工具来满足信息物理系统中各种异构组件的高可靠性和安全性需求。

(3) 系统验证和核实。超越现有技术的软硬件组件、中间件和系统是信息物理系统的另一大需求，软硬件都必须具备很高的可靠性、可重置性以及需要从模块化的组件到整合系统的验证。CPS这类高复杂性的系统对可靠性的需求是当前的传统信息系统所不具备的，例如在航空工业中，相比于开发新的安全关键性系统，对系统可靠性验证大于需要花费掉超过百分之五十的资源，这种情景在其他类似领域，如医疗、自动化、能耗系统中也是很常见的。当前针对安全和可靠性验证的唯一方法就是给予悲观估计的系统设计方法。然而这对复杂的信息物理系统几乎是不切实际的，因此在系统设计阶段亟需新的建模、算法、方法论和工具来将系统验证和核实予以整合。

2.1.4 解决方法

本章针对信息物理系统中亟需解决的问题给出一些可能的解决方法，但是以下任何某一个具体的方法都无法完整的解决好CPS系统设计问题。

(1) 系统验证和鉴定。当前，由于在任何编程语言中都无法表达时间特性，大部分的嵌入式软件开发都是通过原型机测试的方法对系统进行可行性验证。考虑到这种原型机测试的方法已经成功应用于非常多的嵌入式系统中，下一步针对信息物理系统设计一个合理的方式就是进一步提升当前这种测试机制。例如，由于硬件大都具有独特的特性，并且用嵌入式软件效仿实际物理环境也不太容易实现，因此自动回归测试(automated regression test)是很难实现的。针对这种情况下的一种可能的解决方式是探寻更好的测试方法。伯克利的BEE2项目就是其中之一^[107]，他们利用一个基于FPGA的系统来对无线组件进行系统仿真。

另外一种解决方法是利用模拟技术对信息物理系统进行验证。同时对软硬件进行精确地模拟是非常困难的，其中一个主要原因是在现代处理器上执行精确的软件执行模拟所需要的高昂开销。在最近的一次EMSOFT/CASES-ISSS/CODES上，Namsung Woo (三星的执行VP)介绍说三星的虚拟平台ViP (Virtual Platform) 仅能够提供近似精确的软硬件模拟，就是因为想要得到完全精确的模拟结果，开销太大。即使能够得到精确的结果，这种模拟也只能代表在特定环境下的特定软件运行在特定处理器上的结果，任一参数(软件、硬件、环境)稍作改变，模拟结果的

正确性都不再能得到保证。

第三种解决方法是利用形式化验证技术。近年来这一领域也取得了一定的进展，而且对于某些任务模型是非常有效的，例如用来确认可能的死锁条件。然而，某些特性由于不能被形式化的表征从而不能被验证，例如软件的时间特性必须单独形式化定义。这一问题在信息物理系统中变得更为严峻，尽管自动化模型抽象技术已经取得了一定的进展^[108]，在实际系统中的可用性仍然是一个问题。

(2) 软件工程。首先，更好的软件工程设计流程对设计可靠的信息物理系统至关重要。一个有代表性的例子是在2000年左右开始研发的Ptolemy项目^[109]，Ptolemy系统上线后得到了广泛的应用，一直没有出现任何问题，直到2004年，系统运行过程中却突然出现了一段代码死锁。虽然在系统上线之前已经经过了严格的软件工程测试，但依然出现了严重的死锁这种bug，而且会不会再出现其他的问题？再过多久才会出现？所以，到底需要多久的测试才能真正保证系统百分百正确。当然，有很多简单的编程规则可以避免死锁的发生，例如总是在相同的顺序内使用锁^[110]。但是这一规则说起来简单，在实际编程中遵守起来却很难，尤其对于越来越复杂的程序设计需求。

另一种方法是在并行计算中使用审核设计模式^[110, 111]，例如MapReduce这类更高层的模式为程序语言设计者提出了很多新的机遇与挑战，这些模式是在协调语言（coordination language）层起作用而非传统的程序语言（programming languages）层。在信息物理系统软件工程中，在已有程序语言上（如Java、C++）设计兼容的协调语言，比直接设计新的程序语言更加能够适应CPS的设计需求。

(3) 新技术。虽然以上这些解决方案能够对信息物理系统设计起到帮助，但是这种影响始终是有限的，要想充分发挥CPS的潜能，仍然需要一系列根本上的新技术支持。例如，针对并行编程，Split-C^[112]和Cilk^[113]都是支持多线程的类C语言且具能够更好更容易的理解和控制程序行为，但是这些语言仍然无法避免发生死锁的风险。解决这一问题的最好方法是实现完全的可预测性，这在技术上是可行的，但需要要有全新的技术来改变原有的解决思路。比如说改变计算机的体系结构来使得能够表达精确的时间特性^[114]，这样就能够在编程时为物理世界中的各种并行行为建立可预测性的程序设计。

2.2 实时调度

在一个实时系统中，实时调度器决定了一系列共享资源（CPU 或网络带宽）的实时任务的执行顺序，实时调度的目的是保证系统中每个实时任务的时间

特性都能够得到满足。通常来说，实时调度可以分为线下（offline）调度和在线（online）调度。线下调度是在系统执行之前就已经做好所有的调度决策，系统运行之后就按照调度表给出的顺序执行所有的实时任务。在线调度则是在系统运行过程中根据系统的时间特性在线的做出调度决策。由于在实时调度领域有各种各样的实时调度模型及算法，本章仅介绍与本文关系紧密的一些术语和概念。

2.2.1 实时调度策略

基于时间的调度策略。时间驱动的调度器工作方式如下：调度器创建一个调度表（scheduler或者table），一般来说这个调度表是在系统开始运行前就已经生成的（线下调度），但有的调度器也可以在系统运行过程中生成调度表（在线调度）。在系统运行过程中，调度程序（dispatcher）依照调度表来执行每一个实时任务，每个任务只能在调度表中确定的时间点内执行。

通过线下方式来创建调度表，最大的优点就是可以承受相对高昂的时间复杂度开销（调度表生成以及可调度性分析），这对于在线调度方式来说是很困难的。并且，采用线下调度的实时系统具有非常好的可预测性，因为系统中的任务都会严格按照预先生成的调度表来执行。也正因此，线下调度的方式是很多有高安全性需求的应用（如航空控制）所最常采用的调度方式。但是另一方面，线下调度的缺陷也是很明显的，由于调度表是在系统运行前生成的，调度的灵活性就非常有限，一旦系统运行过程中发生任何的改变（例如增加一些功能性，或者改变硬件环境），先下调度并不能提供很好的支持，这也促进了基于优先级调度策略的应用。

基于优先级的调度策略。凡是在系统运行过程中决定任务执行顺序的调度策略都被归类为在线调度。这类调度算法会根据系统运行构成中的时间约束，例如任务优先级，来调度系统中的任务。根据任务优先级来做调度决策的策略统称为基于优先级的调度策略。

相比基于时间的调度策略，基于优先级的调度可以根据任务最新当前的特性来决定哪个任务得到处理器资源来执行，从而大大提高了调度的灵活性。因此，优先级调度可以应对动态的系统工作量变化，比如只要能够保证可调度性，系统可以在线添加或删除任务已实现功能上的灵活变化。基于优先级的调度策略又可进一步分为固定优先级调度（Fixed Priority Schedule, FPS）和动态优先级调度（Dynamic Priority Schedule, DPS），两者的区别在于任务的优先级是否在系统运行中是固定不变的。

(1) 固定优先级调度。当系统采用固定优先级调度时,任务的优先级一旦被分配就不再被改变,系统运行中任意时刻,在所有已就绪的任务中调度器选取具有最高优先级的任务执行。根据具体系统时间特性的不同,可以有很多种不同的固定优先级分配策略。例如,速率单调(Rate Monotonic, RM)算法已经被证明是单处理器上最优的固定优先级分配算法^[12],RM更具任务的周期为其分配相应的优先级,周期越短,优先级越高。

(2) 动态优先级调度。最著名的动态优先级调度策略是最早截止期优先算法(Earliest Deadline First, EDF),在EDF调度算法下,任意时刻所有就绪任务中距离截止期最近的任务被赋予最高优先级,并得到处理器资源执行,因此任务的优先级并不是固定的,而是随着系统运行动态改变的。EDF也已经被证明是单处理器上最优的动态优先级调度算法^[13],并且相比于固定优先级调度策略,EDF具有更高的可调度性。其他的动态优先级调度策略还有最少松弛度优先算法(Least Laxity First, LLF)^[115, 116],松弛时间是一个任务在当前时刻距离其截止期的剩余时间^[117],LLF在任意时刻选择松弛时间最短的任务执行。

分层调度。在某些时候,系统的一部分功能适合于被某种算法调度执行,而系统其余的功能适合被另外一种算法所调度,这种情况下系统一般采用分层调度器对系统进行分层调度。在固定优先级系统中典型的分层调度策略包括基于偶发服务器的(Sporadic Server, SS)^[118]、基于可延期服务器的(Deferrable Server, DS)^[119]以及基于轮询服务器的^[120]。对于动态优先级调度,也存在一些基于EDF的分层调度器,例如文章^[121, 122]中提出的两层调度器,其使用EDF调度若干总带宽服务器(Total Bandwidth Server, TBS)。^[123]中的两层调度器使用固定带宽服务器(Constant Bandwidth Server, CBS)以及^[124]中使用的带宽共享服务器(Bandwidth Sharing Server, BSS)。

2.2.2 实时分析

对于采用线下调度策略的实时系统,一旦调度表在系统运行前生成之后,系统设计者可以很容易的验证该调度表是否可以满足系统中所有任务的时间特性。然而,对于基于优先级的这类在线调度策略,由于调度决策是系统运行过程中在线生成的,系统的执行具有动态性。因此,必须在系统运行之前,使用实时分析(或称可调度性测试)技术来确定系统中的任务集能否在给定的调度策略下满足其所有时间特性。如果通过了可调度性测试,则称系统是可调度的(schedulable或feasible)。

通常有三类不同的可调度性分析技术：基于系统利用率的可调度性测试、基于系统需求量的可调度性测试和基于响应时间的可调度性测试。顾名思义，第一种是基于任务集的利用率来分析是否能够被调度算法调度，第二种则是基于任务集在一段时间区间内的处理器资源需求做分析，第三种则是根据为任务集中的每个任务计算的最坏情况下的响应时间来做分析。基于系统利用率的分析方法通常较另外两种具有更小的时间复杂度以及更快的执行性，但不能不能够适用于一些较为复杂的任务模型^[125]。

2.2.2.1 任务模型和定义

表2.3给出了一些常见的实时任务模型中的基本概念和对应的符号表示。一个实时任务集 τ 通常由若干实时任务（task）组成，每个任务 τ_i 根据一定的时间间隔被循环释放，每次释放称为 τ_i 的一个任务实例，表示为 $\tau_{i,j}$ ，并且需要在一定的时间截止期之前完成执行。这样的任务称为周期性任务，一个周期性任务 τ_i 通常可以用一个三元组 $\langle C_i, D_i, T_i \rangle$ 来表示。其中， C_i 为任务的最差执行时间，即每次释放的任务实例所需要的最长执行时间； D_i 为任务的相对截止期，即每次释放的任务实例必须在释放后的 D_i 时间内完成执行； T_i 为任务的最短释放间隔（也称为周期），即任务的两次连续释放所需要的最短时间间隔。如果任务集中的所有任务都在同一时刻释放其首个任务实例（通常认为是0时刻），这样的任务集称为同步周期性任务集，否则称为非同步任务集^[126]。

定义 2.1 (资源利用率)：一个任务 τ_i 的资源利用率为

$$U_i = \frac{C_i}{T_i}$$

资源利用率表示一个任务给系统带来的工作负载，即在一段时间区间内系统执行该任务所需处理器资源的最大比例。

定义 2.2 (系统资源利用率)：一个系统（或任务集 τ ）的资源利用率为

$$U(\tau) = \sum_{\tau_i \in \tau} \frac{C_i}{T_i}$$

2.2.2.2 基于系统利用率的实时分析

针对于固定优先级调度和动态优先级调度，Liu和Layland都提出了相应的基于系统利用率的可调度性分析方法^[12]，具有重大的意义。

在文章^[127, 128]中，Fineberg和Serlin给出了一种针对于速率单调算法调度下的同步周期任务集的基于系统利用率的可调度性测试方法，之后Liu和Layland给出了形式化的证明。根据其研究结果，一个周期性任务集只要能够满足公式(2.1)的可调度判定条件，系统就能够被速率单调算法所调度。

表 2.3 实时调度基本概念
Table 2.3 Definitions in Real-Time Scheduling

概念	符号表示
任务集	τ
任务 <i>i</i>	τ_i
任务 <i>i</i> 释放的第 <i>j</i> 个实例	$\tau_{i,j}$
任务集中任务个数	N
任务 <i>i</i> 的最差执行时间 (Worst-Case Execution Time, WCET)	C_i
任务 <i>i</i> 的周期	T_i
任务 <i>i</i> 的相对截止期	D_i
任务实例 $\tau_{i,j}$ 的释放时刻	$r_{i,j}$
任务实例 $\tau_{i,j}$ 的绝对截止期	$d_{i,j}$
任务 <i>i</i> 的响应时间	R_i

$$\sum_{\tau_i \in \tau} \frac{C_i}{T_i} \leq N \times (2^{1/N} - 1) \tag{2.1}$$

该可调度性测试只能保证如果一个系统满足(2.1)的条件，就不会出现任务执行过程中错失截止期的情况。当系统中的任务个数*N*趋近于无穷大时，该可调度性判定条件能够提供的系统资源利用率下限大约为69%，但是，(2.1)是一个充分非必要判定条件，也就是说，可能会出现系统并不能通过该可调度性测试但实际系统可调度。之后，Lehoczky通过使用大量的随机周期性任务集进行实验，得出被速率单调算法调度的系统的“真实”系统利用率约为88%^[129]。并且，Lehoczky还提出一种精确的分析方法^[129]，但该方法的分析复杂度非常高。另外，当任务集中的任务周期互为整数倍时，称谐波任务集，此时任务集在速率单调算法下的利用率界限可达100%^[130]。针对固定优先级算法的基于系统利用率的可调度性分析方法，读者可以参考文章[131]以获得更全面详细的了解。

Liu和Layland同样针对动态优先级分配策略EDF给出了相应的基于系统利用率的可调度性分析方法：

$$\sum_{\tau_i \in \tau} \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

对于每个任务的截止期等于周期这样的完全可抢占任务集，公式(2.2)是一个充分必要的判定条件，Coffman随后证明了即使对于非同步的实时任务集，该判定条件仍然是充要的^[132]。

2.2.2.3 基于系统需求量的实时分析

处理器需求量 (processor demand) 是实时系统分析中另外一个常用的度量指标, 它描述了一个实时任务集在一段时间区间内若想满足其时间需求所需要的处理器资源量。在时间段 $[t_1, t_2)$ 内的处理器需求量, 表示为 $DB_{[t_1, t_2)}$, 能够通过以下公式求得:

$$DB_{[t_1, t_2)} = \sum_{t_1 < r_k, d_k \leq t_2} C_k \quad (2.3)$$

其中, r_k 和 d_k 分别是任务 τ_k 的释放时刻和绝对截止期。也就是说, 处理器需求量表示的是系统在任意给定的一段时间区间内所一共释放的工作量总和。

由于处理器需求量描述的是在一段确定的时间区间内 ($[t_1, t_2)$) 的工作量, 在实时性分析时很难遍历的检查所有的时间区间, 更加实际的方法是分析出在一段固定时间长度内处理器需求量的上限, 即需求界限函数 DBF_t (demand bound function)。公式(2.4)给出了对于一般周期性任务集需求界限函数的计算方法:

$$DBF_t = \sum_{D_i \leq t} ([1 + \lfloor \frac{t - D_i}{T_i} \rfloor] \times C_i) \quad (2.4)$$

其中, D_i 和 T_i 分别是任务 τ_i 的相对截止期和周期。随后, 任务集是可调度的当且仅当满足以下不等式:

$$\forall t, DBF_t \leq t \quad (2.5)$$

Baruah等人对公式(2.4)进行扩展, 支持包含截止期大于周期的实时系统^[13]以及截止期小于周期的实时系统^[133]可调度性分析。另外, George将该分析方法扩展到对不可抢占EDF的系统分析上^[46]。其他的一些扩展, 例如偶发任务集的可调度性分析也同样被研究者们相继提出^[134]。

2.2.2.4 基于响应时间的实时分析

基于响应时间的实时分析是通过计算任务集在某个特定的调度算法下每个任务在最坏情况下的最长响应时间, 来判断是否能够满足所有任务的时间特性。一个任务的响应时间是从其释放到完成执行所经历的时间长度, 而最长响应时间则是其释放的所有任务实例中最长的响应时间。通过比较一个任务的最长响应时间和其相对截止期的大小关系, 就可以判定该任务是否可调度。对于很多调度算法, 很难高效的计算出每个任务的响应时间, 因此类似于基于处理器需求量的分析方法, 可以通过计算出一个任务响应时间的上限值来进行充分而非必要的可调度行

判定。

第一篇研究实时系统基于响应时间的可调度性分析方法的文章是Joseph等人提出的^[135]，他们针对可抢占固定优先级系统，提出了计算最坏情况响应时间上限的方法：

$$R_i = \sum_{j \in hp(i)} \left(\lceil \frac{R_i}{T_j} \rceil \times C_j \right) + C_i \quad (2.6)$$

其中， $hp(i)$ 包含了所有优先级比任务 τ_i 高的任务。

Audsley等人随后将该分析方法扩展到了不可抢占固定优先级系统中^[136]，对于不可抢占任务集，当一个低优先级的任务已经开始执行后，高优先级的任务必须等待该低优先级任务执行完成后才能得到处理器资源，因此高优先级任务的响应时间中包含一段被低优先级任务所阻塞的时间，在最坏情况下即任务集中比任务 τ_i 优先级低的任务中最长的最坏执行时间。

第3章 EDF-VD调度下的混合关键性系统可调度性分析

近年来，嵌入式系统设计的一个趋势是将各种不同关键性的功能集成在一个共享资源平台上，以获得更好的性能及更少的能耗。在这种混合关键性系统中，对于高关键性的系统功能，为了达到更高标准的正确性保证，我们需要在系统分析时对高关键性级别的应用给予较悲观的假设（例如通过静态分析得到的最差执行时间上限）；而对于低关键性的系统功能，我们则需要保证系统中所有关键性级别的应用都能够满足其时间要求。

混合关键性系统给实时系统的设计带来了巨大的挑战。传统的实时系统调度算法，如EDF和RM，当被直接应用于混合关键性系统中时会带来巨大的资源浪费。近年来，研究者们通过系统的分析混合关键性系统中不同关键级别之间的特性，提出了一些新的调度算法来提升系统资源使用率。其中，EDF-VD（Earliest Deadline First-Virtual Deadlines）已经被证明相较其他算法具有更好的可调度性以及运行时效率。顾名思义，EDF-VD的主要思想是以最早截止期优先算法（EDF）作为基本调度策略，当系统运行于不同于关键性级别时，通过给任务设置不同的虚拟截止期来平衡不同关键性级别的可调度性。

虽然EDF-VD算法已经展现了较其他针对混合关键性系统的调度算法更为出色的性能，但是其特点优势还并未被完全开发出来。当前针对EDF-VD算法的可调度性分析技术仍然非常悲观，这可能会导致很多实际能被EDF-VD算法调度的系统被判定为不可调度，本章的主要目的就是设计新的可调度性分析方法来更精确的分析EDF-VD调度下的混合关键性系统。

EDF-VD算法的可调度性分析最大的难点在于系统运行时的关键性切换行为，系统关键性级别的转换通常是被某些任务的过载运行所触发的，而问题是，这种系统关键性级别转换何时发生，以及发生时系统处在什么样的运行状态（例如哪些任务还未执行完），这些都是系统设计者无法预先知道的。如果简单的在系统设计时遍历所有的运行时状态可能性，这种方式所带来的计算复杂度是无法估量的，显然不切实际，因此研究者们转而去设计一些近似的分析策略以平衡分析精度和分析复杂度。在文章[36, 37]中，Ekberg和Yi提出了最新的针对EDF-VD算法的可调度性分析技术，本文称为EY算法，其主要思想是单独的分析系统处于不同关

键性级别的状态,近似的计算系统从低关键性级别向高关键性级别转换时所带入的工作量。然而,这种近似的结果非常不精确,并且这种偏差会随着系统关键性级别数量增加而不断累计。因此EY算法对EDF-VD分析的结果非常悲观,尤其对于多关键性级别的混合关键性系统。

针对这一问题,在本章中,我们设计新的针对EDF-VD的可调度性分析方法。不同于EY算法对系统不同关键性级别的行为单独分析,我们的分析方法考虑系统关键级转换时的行为,将不同关键级别下的系统工作量作为整体来分析,以最终获得更为精确的分析结果。通过大量实验表明,我们的分析方法相较于EY算法能够获得更好的分析性能,具有更高的系统可调度性。随着系统关键性级别数量的增加,我们算法相比于EY的优势变得更加明显,当然,这种精确性的提升是以分析复杂度的增加为代价的,然而,通过结合我们的分析方法和EY算法,我们能够获得分析性能和分析效率的有效平衡。

3.1 预备知识

3.1.1 混合关键性任务系统

作为传统周期性任务集的扩展,混合关键性任务系统 τ 由若干相互独立的任务组成, τ 中的每个混合关键性任务 τ_i 可以由一个元组 (T_i, D_i, l_i, C_i) 来表示:

- T_i 是任务 τ_i 的最小释放间隔时间,也称周期。
- D_i 是任务 τ_i 的相对截止期。
- $l_i \in \{1, \dots, L\}$ 是任务 τ_i 的关键性级别,其中 L 是系统总的关键性级别个数,数字越小代表的关键性级别越低。
- $C_i = (C_i^1, \dots, C_i^L)$ 是任务 τ_i 的最差执行时间向量,其中 C_i^l 代表 τ_i 在系统处于关键性级别 l 时的最差执行时间,并且我们假设如果 $x < y$ 则 $C_i^x < C_i^y$ 。

每个任务 τ_i 会周期性的释放无限多个任务实例,我们用 J_i^k 代表任务 τ_i 释放的第 k 个实例,如果 τ_i 在 r 时刻释放了一个实例,则该实例的绝对截止期 d 等于 $r + D_i$ 。为了简化符号,我们用 J_i 来表示 τ_i 释放的任务实例。

系统从最低的关键性级别开始运行,如果在运行过程中,任一任务 τ_i 在其最差执行时间 C_i^1 内没有完成执行,则系统会立即转换到更高的第2级别关键性运行。之后所有关键级别为1的任务的截止期都不再被要求满足,但是对于所有其他其他高关键性级别的任务,需要按照更悲观的执行时间 C_i^2 来执行。相似的,如果任一任务在此时没有在 C_i^2 时间内完成执行,系统则进一步转换到更高的关键性级别运行。在实际中,系统可能会由于处理器空闲从某个高级别的关键性运行状态中

转换回到低级别关键性运行，但从建模角度，我们假设当系统一旦进入到关键性级别 l 后，所有关键性级别 l 以下的任务都立即被丢弃。

我们用 $\tau(l)$ 表示具有关键性级别 l 的所有任务子集合，即 $\tau(l) = \{\tau_i | l_i = l\}$ ，那么系统的可调度性可定义为如下。

定义 3.1 (混合关键性系统的可调度性): 一个混合关键性系统可调度，当且仅当在系统运行于关键性级别 l 时，所有任务 $\cup_{k \geq l} \tau(k)$ 都能满足它们的相对截止期。

为了简化表示，当系统仅包含两个关键性级别时（即 $l_i \in \{1, 2\}$ ），我们用低关键级表示关键级别1，用高关键级表示关键级别2。

3.1.2 EDF-VD算法

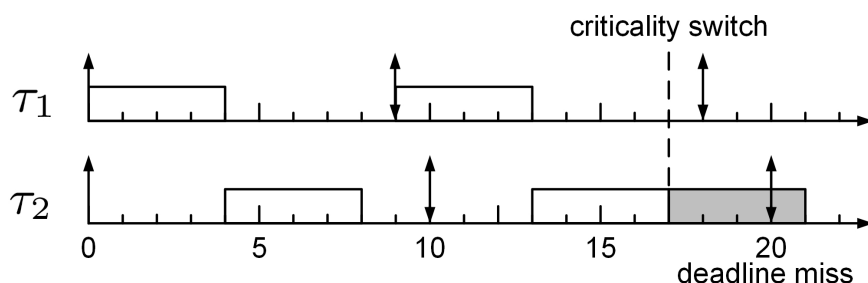


图 3.1 EDF调度下的混合关键性任务集

Fig. 3.1 Mixed-criticality task set scheduled by EDF.

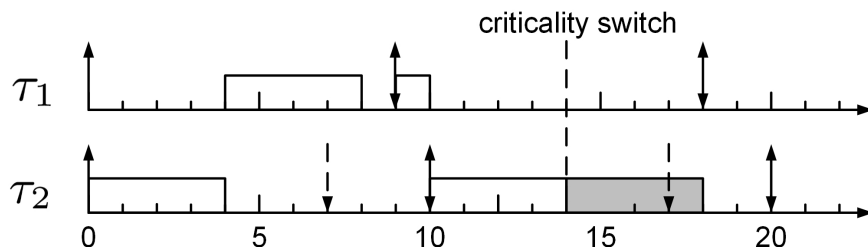


图 3.2 EDF-VD调度下的混合关键性任务集

Fig. 3.2 Mixed-criticality task set scheduled by EDF-VD.

在传统周期性实时系统中，EDF算法是最优的，然而对于以上介绍的混合关键性系统来说，EDF可能会带来较低的系统性能。例如，我们使用EDF算法调度

表 3.1 混合关键性任务集

Table 3.1 Mixed-Criticality Task Set.

任务	C_i^{LO}	C_i^{HI}	T_i	l_i	D_i^{LO}	D_i^{HI}
τ_1	4	—	9	LO	9	—
τ_2	4	8	10	HI	7	10

表3.1中具有高低两种关键性级别的混合关键性任务集，该任务集仅包含两个不同关键性级别的任务，且两个任务同时在系统0时刻释放其第一个任务实例。如图3.1所示，假设系统在低关键性级别运行过程中，高关键性任务 τ_2 释放的第二个任务实例在执行了其最差执行时间 $C_2^{LO} = 4$ 后并没有标志完成，则系统在该时刻($t = 17$)立即被触发进入到高关键性级别继续运行。之后，虽然低关键性任务 τ_1 被直接丢弃，但由于高关键性任务 τ_2 的最差执行时间 $C_2^{HI} = 8$ ，其仍然需要4个时间单位来完成执行，由于距离其绝对截止期只剩3个时间单位，显然 τ_2 将会错失截止期。该例子说明，以防系统突然发生关键级转换，高关键性任务在系统处于低关键性级别运行时，即使其截止期比较靠后，也应该为其赋予一个较高的优先级以其能够为可能发生的系统关键级转换预留足够的时间。

在EDF基础上，EDF-VD算法为每个任务在不同系统关键级别下设置不同的虚拟截止期，调度器根据这些虚拟截止期来为其释放的实例赋予相应的优先级。我们用 D_i^{HI} 和 D_i^{LO} 来表示任务 τ_i 在高低关键级别下的虚拟截止期，用 D_i^l 来表示 τ_i 在关键性级别 l 下的虚拟截止期，如果一个任务本身的关键性级别小于 l ，则无需为其设置在关键性 l 下的虚拟截止期，即 D_i^l 。此时再来看刚才的例子，如果我们给任务 τ_2 设置虚拟截止期 $D_2^{LO} = 7$ （其他所有参数不变），那么更新后的调度表则如图3.2所示， τ_2 释放的第二个实例将拥有比 τ_1 释放实例更高的优先级，即使在相同时刻系统转换到高关键性级别， τ_2 仍然有足够的时间来完成高关键性级别下的最差执行时间。

通过设置这样的虚拟截止期，我们有机会为那些在系统发生关键级转换时释放的任务实例获取额外的松弛时间，并且减少高关键性任务带入到系统高关键性级别运行时的工作量。另一方面，如果为任务设置完虚拟截止期后，系统在低关键性级别下变得不可调度的话，我们可以延长某些任务的虚拟截止期。总之，要想让系统在所有关键性级别下都可调度，需要为所有任务设置合适的虚拟截止期。

3.1.3 现有的EDF-VD分析方法

对于一个混合关键性任务系统，我们想要知道的是，是否能够找到一种为系统中所有任务配置合适虚拟截止期的方法，使得系统在所有关键性级别下都可以被EDF算法所调度而不出现任何任务错失截止期的情况。为此，我们需要解决以下两个问题。

- (1) 可调度性测试。给定一种虚拟截止期配置，如何来判断系统是否在所有

关键性级别下都可以被EDF所调度。

(2) 虚拟截止期调整。给定一种可调度性判定方法，如何来为所有任务配置虚拟截止期，使得系统可调度。

可调度性测试

Ekberg和Yi在文章[36]中研究了EDF-VD算法的可调度性测试问题，它们考虑仅包含高、低两种关键性级别的混合关键性系统，系统在低关键性级别下的可调度性可以通过标准的需求界限函数来分析^[13]。而为了分析系统处于高关键性级别下的可调度性，每个在系统发生关键性转换时释放的高关键性任务都被分割成两个部分，一半是其低关键性虚拟截止期之前的执行部分，另一半是在其低关键性虚拟截止期和原始截止期之间的执行部分。然后，EY方法仅分析系统发生关键性转换之后的时间区间。

现在我们来考虑表3.1中的混合关键性任务集，根据EY方法，当系统发生关键性级别转换后，高关键性任务 τ_2 被“拆分”成转换时刻前后的两个子任务，对于后一个子任务，其释放时间为系统发生关键性级别转换时刻，截止期为 τ_2 原始的绝对截止期，但显然，如图3.3所示，这种情况下该任务的执行时间将会大于相对截止期 ($C_2^{HI} - C_2^{LO} > D_2^{HI} - D_2^{LO}$)，无论如何都不可调度。造成这种结果的主要原因是，EY方法悲观的假设每个高关键性任务在系统处于低关键性级别下的执行部分都是在其低关键性虚拟截止期时刻刚好完成，而它额外需要执行的高关键性级别部分的执行时间 ($C_2^{HI} - C_2^{LO}$) 全部需要在其虚拟截止期和实际截止期之间 ($D_2^{HI} - D_2^{LO}$) 完成。而在系统实际执行过程中，这种情况可能并不会发生，也就是说，该任务示例的 C_2^{LO} 部分可能在虚拟截止期 D_2^{LO} 之前很久就已经完成执行了。

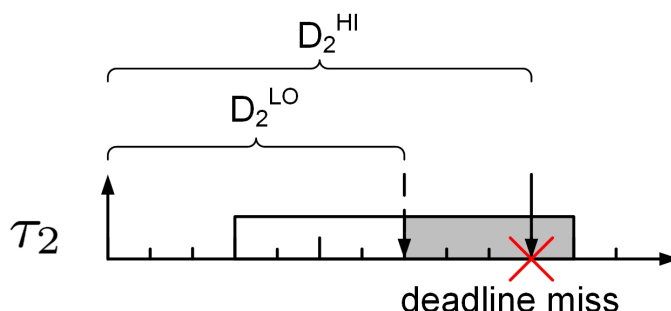


图 3.3 EY分析方法示例

Fig. 3.3 Illustration of EY analysis.

文章[34]中的可调度性测试方法（本章称为BA算法）具有跟EY方法相似的思想，但是是基于系统利用率界限而非需求界限函数，BA算法同样也具有刚刚讨论的EY算法所遭遇的缺陷，甚至BA算法的结果更加悲观（由于为了得到资源利用率

界限而作的近似工作量估计)。

虚拟截止期调整

在文章[34]中,虚拟截止期的设置是通过选择一个固定的比率,给每个任务不同关键性级别设置虚拟截止期。随后文章[36]改进了这种方法,从高到低的给每个任务在不同关键级别下设置虚拟截止期时,在保证任务在当前关键级别下可调度的情况下,尽可能的为任务在更低优先级下设置一个相对长的虚拟截止期。

3.2 新的可调度性分析方法

正如在上一小节中讨论的, EY方法对所有跨越系统关键级转换时刻释放的任务实例做了过于悲观的工作量估计。具体来说,对于每一个系统关键级, EY方法都假设每一个任务在前一关键级别下尽可能晚的执行,然而在系统实际运行过程中,这种情况并不总是发生,任务可能在系统发生关键级转换之前很久就已经完成执行了, EY这类对系统在不同关键级别下的运行行为进行单独分析的方法摆脱这种悲观的估计,因为必须假设系统从关键级转换时刻开始重新运行了。

为了解决这一问题,提高分析的精度,我们提出一种新的可调度性分析方法,该方法对系统跨越关键级转换时刻的行为一并分析,以此对系统的运行行为得到一个更加全面的了解,来获得更加精确的分析结果。接下来,我们首先假设系统仅包含高、低两种关键性级别(简称为两级关键性系统),来介绍我们提出的新的可调度性方法,之后我们再将该方法扩展于更加一般化的包含若干关键性级别的混合关键性系统(简称为多级关键性系统)。

3.2.1 两级关键性系统

为了验证一个两级关键性系统的可调度性,我们需要证明以下两个条件成立。

- (1) 当系统运行于低关键性级别时,所有高、低关键性的任务都要满足它们的截止期约束。
- (2) 当系统运行于高关键性级别时,所有高关键性任务 ($\tau_i \in \tau(HI)$) 都要满足它们的截止期约束。

为了验证条件(1)是否成立,我们可以直接利用基于需求界限函数的可调度性测试方法来验证,即判断以下不等式是否成立,

$$\forall x, \sum_{\tau_i \in \tau} dbf_i^{LO}(x) \leq x$$

其中,

$$dbf_i^{LO}(x) = n_i(x) \times C_i^{LO} \quad (3.1)$$

是任务 τ_i 在系统处于低关键性级别下的需求界限函数, 其中 $n_i(x)$ 表示 τ_i 在长度为 x 的时间区间内所能释放最多的任务实例个数, 即释放时刻和截止期都在区间内。

$$n_i(x) = \max(\lfloor (x - D_i) / T_i \rfloor + 1, 0)$$

但是, 对于条件(2)的验证就困难得多了, 用 $n_i(x) \times C_i^{HI}$ 来任务 τ_i 在系统处于高关键性级别下释放的任务实例个数显然是不精确的, 因为系统在发生关键级转换后存在一段“转换阶段”。接下来我们就着重来分析如何验证条件(2)成立。

我们用 $[t_1, t_d]$ 来表示包含系统“转换阶段”的一段时间区间, 其中 t_d 是关键级转换时刻(表示为 t_s)后的任一时刻, t_1 是 t_d 之前最近的时刻, 并且满足在区间 $[t_1, t_d]$ 之间高关键性任务子集 $\tau(HI)$ 包含至少一个活跃实例, 即截止期在 t_d 之前并且在 t_1 时刻还未执行完成的任务实例。如果 t_1 在 t_s 之后($t_1 > t_s$), 那么也就是说, 所有在系统关键级转换时刻 t_s 之前释放的任务实例在 t_1 时刻都已经执行完成了, 则区间 $[t_1, t_d]$ 内的工作量全部都是由 t_1 时刻之后释放的任务实例所贡献的, 这种情况下就可以直接用标准的需求界限函数来分析。接下来我们来关注另外一种更有意思的情况, 即 $t_1 \leq t_s$ 。我们定义 $x = t_d - t_1$, $y = t_d - t_s$ 。

一个高关键性任务 τ_i 在长度为 x 的时间区间内所最多能包含的活跃实例的数量为 $n_i(x)$, 如果一个实例在 t_s 之前已经执行完成了, 则它最多能执行 $C_i(LO)$ 个时间单位, 因为不然的话系统在 t_s 之前就会被触发关键级转换。所以我们的目标是分析在 $n_i(x)$ 内的这些任务实例, 能够在 $[t_s, t_d]$ 内执行的做多的数量。由于这一数量不仅取决于 x 的长度, 还跟 y 的长度有关, 因此我们用 $m_i(x, y)$ 来表示这一数量。那么, 任务 τ_i 在区间 $[t_1, t_d]$ 内贡献的总的工作量可以计算为:

$$dbf_i^{HI}(x, y) = m_i(x, y) \times C_i^{HI} + (n_i(x) - m_i(x, y)) \times C_i^{LO} \quad (3.2)$$

很显然, τ_i 所有在系统关键级转换时刻 t_s 之后释放的任务实例都应当算在 $m_i(x, y)$ 内, 问题在于如何考虑那些“跨越”关键级转换时刻的任务实例, 为了方便描述, 我们首先定义跨越实例如下。

定义 3.2 (跨越实例): 如果一个任务实例在系统关键级转换时刻之前释放, 并且截止期在关键级转换时刻之后, 那么该实例称为跨越实例。

接下来我们讨论如何安全的分析 $m_i(x,y)$ 中包含的跨越实例的数量，从而获得 $m_i(x,y)$ 的上界。

引理 3.1: 如果一个高关键性任务 τ_i 在系统处于低关键性级别下能够满足它的低关键性（虚拟）截止期 D_i^{LO} ，那么一定满足 $m_i(x,y) \leq p_i(x,y)$ ：

$$p_i(x,y) = \begin{cases} n_i(y), & y \bmod T_i < S_i \\ \min(n_i(y) + 1, n_i(x)), & \text{otherwise} \end{cases} \quad (3.3)$$

证明: 用 t_a 表示跨越实例 J_i 标志完成或执行了 C_i^{LO} 的时刻，用 t_b 表示 J_i 的绝对截止期。

如果 $t_a < t_s$ ，如图3.4-(a)所示，跨越实例在 t_a 时刻一定已经标志执行完成了（否则系统关键级转换将会发生在 t_s 之前）。这种情况下，只有 τ_i 释放的普通实例能够在 $[t_s, t_d]$ 内执行，这一数量的上界可以通过 $n_i(y)$ 计算得出。

如果 $t_a \geq t_s$ ，如图3.4-(b)所示，跨越实例在 t_s 时刻可能还没有完成执行。这种情况下，跨越实例和普通实例都可以在区间 $[t_s, t_d]$ 内执行，从而其总数小于 $n_i(y) + 1$ 。并且，这一数量可以通过跟 $n_i(x)$ 取小来得到进一步的精确，以排除跨越实例在 t_1 时刻之前就已经释放的情况。总之，当 $t_a \geq t_s$ 时， $m_i(x,y) \leq \min(n_i(y) + 1, n_i(x))$ 。

接下来的证明是来说明如何判断 t_a 和 t_s 之间的大小关系。我们考虑任务 τ_i 的最坏释放模式，即其每个实例都是尽可能早的释放，并且其中一个实例的截止期跟 t_d 对齐，如图3.1所示，从而我们可以推断出

$$y \bmod T_i < t_b - t_a \Rightarrow t_a < t_s \quad (3.4)$$

由于 τ_i 在系统处于低关键性级别下是可以满足截止期的，那么我们可以得知 $t_a \leq t_r + D_i^{LO}$ ，其中 t_r 是 J_i 的释放时刻。另一方面，由于 $t_b = t_r + D_i^{HI}$ ，因此以下公式成立，

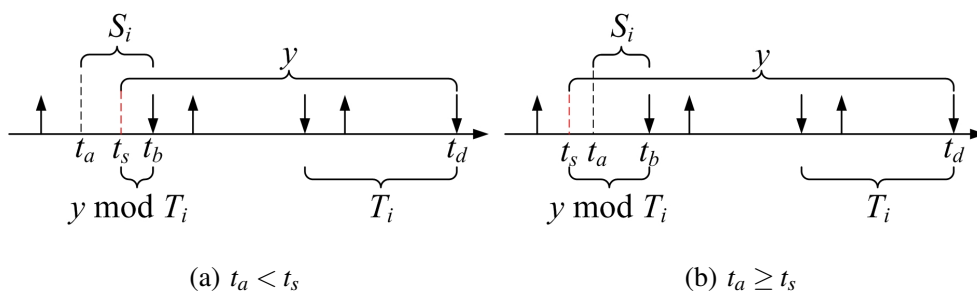


图 3.4 引理3.1的证明示例

Fig. 3.4 Illustration of the proof of Lemma 3.1.

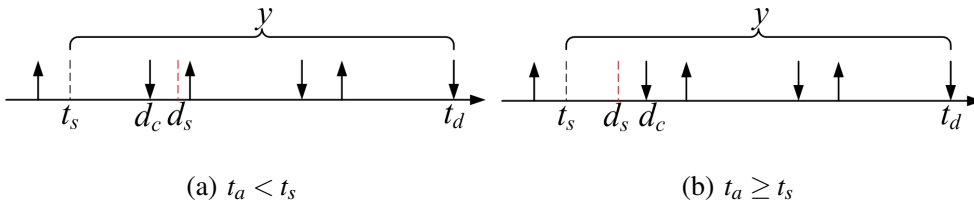


图 3.5 引理3.2的证明示例

Fig. 3.5 Illustration of the proof of Lemma 3.2.

$$t_b - t_a \geq D_i^{HI} - D_i^{LO} \quad (3.5)$$

结合公式3.4和3.5, 我们得到 $y \bmod T_i < D_i^{HI} - D_i^{LO} \Rightarrow t_a < t_s$. \square

引理 3.2: 首先定义 \mathcal{S} ,

$$\mathcal{S} = \min_{\tau_i \in \tau(HI)} \{D_i^{HI} - D_i^{LO}\}$$

那么 $q_i(x, y)$ 可以作为 $m_i(x, y)$ 的一个上界:

$$q_i(x, y) = \max \left(\min \left(\left\lceil \frac{y - \mathcal{S}}{T_i} \right\rceil, n_i(x) \right), n_i(y) \right). \quad (3.6)$$

证明: 假设任务 τ_s 在系统处于低关键性级别是触发了系统的关键级转换, 即 τ_s 的一个任务实例 J_s 执行了 C_s^{LO} 时刻之后并没有标志完成。我们用 d_s 表示 J_s 的绝对截止期。如果 $d_s > t_d$, 那么任何截止期在 $[t_1, t_d]$ 之间的任务实例都具有比 J_s 更高的优先级, 那么只有在 t_s 时刻之后释放的 (正常) 任务能够在系统处于高关键性级别时执行。这样的正常实例的个数, 即释放时间和截止期都在区间 $[t_s, t_d]$ 内的实例, 可以直接由 $n_i(y)$ 计算得到一个上界。接下来我们着重考虑 $d_s \leq t_d$ 的情况。

同样, 我们考虑任务 τ_i 的最坏释放模式, 即其每个实例都是尽可能早的释放, 并且其中一个实例的截止期跟 t_d 对齐, 我们用 J_c 表示跨越实例, d_c 表示 J_c 的绝对截止期。

如果 $d_c < d_s$, 如图3.5-(a)所示, 跨越实例具有比 J_s 更高的优先级, 并且当 J_s 在 t_s 时刻执行时已经完成执行。因此, 该跨越实例不会在系统处于高关键性级别运行时执行, 从而应当从 $m_i(x, y)$ 内排除。否则, 如图3.5-(b)所示, 该跨越实例可能在 J_s 之后执行, 从而不能从 $m_i(x, y)$ 内排除。

通过以上讨论, 我们得知当一个区间 $[t_1, t_d]$ 内的任务实例满足任一以下两个条件时, 它可以在系统处于高关键性级别运行时执行。(1) 它的截止期在 $[d_s, t_d]$ 内; (2) 它是一个正常实例。满足条件 (1) 的任务实例数量不会超过 $\lceil (t_d - d_s) / T_i \rceil$, 然后可以进一步跟 $n_i(x)$ 取小来排除该跨越实例在 t_1 时刻前已经释放的情况。总之, 满足条件 (1) 的任务实例数量不会超过 $\min(\lceil (t_d - d_s) / T_i \rceil, n_i(x))$,

满足条件(2)的任务实例数量不会超过 $n_i(y)$ 。

总结以上讨论,我们可以推出

$$m_i(x,y) \leq \max \left(\min \left(\left\lceil \frac{t_d - d_s}{T_i} \right\rceil, n_i(x) \right), n_i(y) \right). \quad (3.7)$$

证明的最后一步需要找到一个 $t_d - d_s$ 的上界。由于 J_s 在 t_s 时刻触发了系统的关键性级别转换,即 J_s 在 t_s 时刻已经执行了 $C_i(LO)$ 个时间单位,因此 t_s 和它的绝对截止期之间的距离应当不小于 $D_i^{HI} - D_i^{LO}$,即

$$\begin{aligned} d_s - t_s &\geq D_i^{HI} - D_i^{LO} \\ \Leftrightarrow t_d - d_s &\leq y - (D_i^{HI} - D_i^{LO}) \\ \Rightarrow t_d - d_s &\leq y - \mathcal{S} \end{aligned}$$

将此带入到公式3.7中,即得到我们要证明的结果。□

利用以上引理中我们得到的 $m_i(x,y)$ 的上界,我们可以建立对于两级关键性系统 τ 的可调度性判定方法。

定理 3.3: 两级关键性系统 τ 是可调度的,当且仅当以下条件满足,

$$\forall x \geq y \geq 0: \sum_{\tau_i \in \tau(LO)} dbf_i^{LO}(x-y) + \sum_{\tau_i \in \tau(HI)} dbf_i^{HI}(x,y) \leq x \quad (3.8)$$

其中, $dbf_i^{LO}(x-y)$ 由公式3.1计算得到, $dbf_i^{HI}(x,y)$ 可以通过结合

$$m_i(x,y) = \min(p_i(x,y), q_i(x,y)) \quad (3.9)$$

由公式3.2得出。

证明: 根据 $dbf_i^{HI}(x,y)$ 的定义,当 $y=0$ 时, $dbf_i^{HI}(x,y) = dbf_i^{LO}(x)$ 。从而条件(3.8)等价于

$$\forall x \geq 0: \sum_{\tau_i \in \tau} dbf_i^{LO}(x) \leq x.$$

这正是标准的需求界限函数对于系统处于低关键性级别下的可调度性判定。

接下来我们着重证明当系统处于高关键性级别运行时,所有高关键性任务都能满足它们的截止期约束。我们通过反证法来证明,假设 J_i 是第一个在 t_d 时刻错失截止期的任务实例,我们还是定义 t_d 是关键级转换时刻 t_s 后的任一时刻, t_1 是 t_d 之前最近的时刻,并且满足在区间 $[t_1, t_d]$ 之间高关键性任务子集 $\tau(HI)$ 包含至少一个活跃实例,即截止期在 t_d 之前并且在 t_1 时刻还未执行完成的任务实例。对于 t_1 和 t_s 的大小关系,有两种情况:(1) $t_1 > t_s$, (2) $t_1 \leq t_s$ 。

我们首先考虑第一种情况。由于 $t_1 > t_s$,所有低关键性任务在 t_1 时刻都已经被丢弃了,因此只有释放时间和截止期都在 $[t_1, t_d]$ 区间内的高关键性任务释放的实例

能够在 $[t_1, t_d]$ 内执行。任务 τ_j 所释放的这种实例个数的上界为

$$W_j = n_j(t_d - t_1) \times C_j^{HI}.$$

由于我们假设 J_i 错失了它的截止期，所以

$$\sum_{\tau_j \in \tau(HI)} W_j > t_d - t_1.$$

另一方面，根据 $p_i(x, y)$ 的定义，

$$n_j(t_d - t_1) \leq p_j(t_d - t_1, t_d - t_1)$$

因此 W_j 不会超过 $dbf_j^{HI}(t_d - t_1, t_d - t_1)$ 。结合以上这些，我们得到

$$\sum_{\tau_j \in \tau(HI)} dbf_j^{HI}(t_d - t_1, t_d - t_1) > t_d - t_1$$

这跟条件(3.8)是矛盾的，也就是说我们的假设不成立。

再来考虑第二种情况 $t_1 \leq t_s$ 。由于低关键性任务在 t_s 时刻被丢弃了，只有在 $[t_1, t_s]$ 内释放的实例能够在 $[t_1, t_d]$ 内执行，因此低关键性任务 τ_j 在 $[t_1, t_d]$ 内的工作量不会超过 $n_j(t_s - t_1) \times C_j^{LO}$ 。对于每个高关键性任务，它所释放的截止期在 t_d 之前的实例不会超过 $n_j(t_d - t_1)$ 。根据引理3.1和3.2，在这些实例中，执行了 C_j^{HI} 单位时间的实例数量不超过 $m_j(t_d - t_1, t_d - t_s)$ （公式(3.9)），因此释放时间和截止期都在 $[t_1, t_d]$ 内的任务实例工作量上界为 $dbf_i^{HI}(t_d - t_1, t_d - t_s)$ 。并且由于 J_i 在 t_d 时刻错失了截止期，因此整个任务集中释放时间和截止期都在 $[t_1, t_d]$ 内的总工作量上界一定严格大于 $t_d - t_1$ ，

$$\sum_{\tau_i \in \tau(Lo)} dbf_i^{LO}(t_s - t_1) + \sum_{\tau_i \in \tau(HI)} dbf_i^{HI}(t_d - t_1, t_d - t_s) > t_d - t_1$$

这同样跟条件(3.8)是矛盾的。所以两种情况都会导致矛盾发生，也就是说我们假设一个高关键性任务实例错失截止期是不成立的。□

在接下来的章节3.2.3中，我们会条件(3.8)中用于测试的 x 和 y 的取值范围。

3.2.2 多级关键性系统

在这一小节中，我们将以上介绍的可调度性分析方法应用到更为一般化的多级关键性系统中。相应的，我们将 $dbf_i^{HI}(x, y)$ 扩展为 $dbf_i^l(x_1, x_2, \dots, x_l)$ 来作为任务 τ_i 在区间 $[t_1, t_d]$ 内的工作量上限，即需求界限函数。

$$dbf_i^l(x_1, x_2, \dots, x_l) = \sum_{j=1}^l N_i^j \times C_i^j \tag{3.10}$$

其中， N_i^j 由以下公式迭代算出，

$$N_i^j = \max(\overline{N_i^j} - \overline{N_i^{j+1}}, 0)$$

$\overline{N}_i^{l+1} = 0$ 以及

$$\begin{aligned} \overline{N}_i^1 &= n_i(x_1) \\ \overline{N}_i^j &= \min(p_i^j(x_1, x_j), q_i^j(x_1, x_j)) \\ p_i^j(x_1, x_j) &= \begin{cases} n_i(x_j), & \text{if } x_j \bmod T_i < D_i^j - D_i^{j-1} \\ \min(n_i(x_j) + 1, n_i(x_1)), & \text{otherwise} \end{cases} \\ q_i^j(x_1, x_j) &= \max\left(\min\left(\left\lceil \frac{x_j - \mathcal{S}^j}{T_i} \right\rceil, n_i(x_1)\right), n_i(x_j)\right). \\ \mathcal{S}^j &= \min_{\tau_i \in \cup_{l \geq j} \tau(l)} \{D_i^j - D_i^{j-1}\} \end{aligned}$$

在介绍和证明此可调度性测试条件前，我们首先引入以下辅助性的定理。

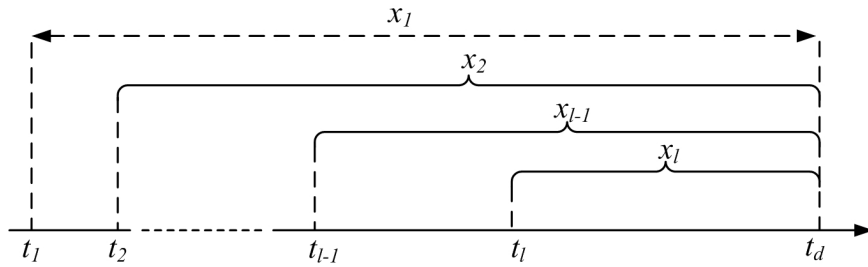


图 3.6 多级关键性系统符号说明

Fig. 3.6 Illustration of notation in multi-criticality systems.

引理 3.4: 给定三个满足以下条件的数字序列 $\{\mu_1, \mu_2, \dots, \mu_K\}$, $\{N_1, N_2, \dots, N_K\}$ 和 $\{C_1, C_2, \dots, C_K\}$,

$$(1) \forall 1 \leq k \leq K: \sum_{l=k}^K \mu_l \leq \sum_{l=k}^K N_l$$

$$(2) C_1 \leq C_2 \leq \dots \leq C_K$$

那么以下不等式成立,

$$\sum_{l=1}^K \mu_l \times C_l \leq \sum_{l=1}^K N_l \times C_l \quad (3.11)$$

证明: 我们用数学归纳法证明此引理。首先, 当 $l = K$ 时, 也就是 $\sum_{l=K}^K \mu_l \leq \sum_{l=K}^K N_l$, 即

$$\mu_K \leq N_K \Rightarrow \mu_K \times C_K \leq N_K \times C_K$$

该引理是成立的。

假设对于 $l = k+1, k+2, \dots, K$ 时引理成立, 那么我们需要证明当 $l = k$ 时, 引理依旧成立。考虑以下两种情况:

(a) $\mu_k \leq N_k$:

$$\begin{aligned} & \sum_{l=k+1}^K \mu_l \times C_l \leq \sum_{l=k+1}^K N_l \times C_l \\ \Rightarrow & \mu_k \times C_k + \sum_{l=k+1}^K \mu_l \times C_l \leq N_k \times C_k + \sum_{l=k+1}^K N_l \times C_l \\ \Leftrightarrow & \sum_{l=k}^K \mu_l \times C_l \leq \sum_{l=k}^K N_l \times C_l \end{aligned}$$

引理成立。

(b) $\mu_k > N_k$: 令 $\lambda_l = N_l - \mu_l$, 则 $\lambda_k < 0$ 。首先, $\sum_{l=k+1}^K \lambda_l > 0$ 一定是成立的, 因为若不然, 那么 $\sum_{l=k+1}^K \mu_l \geq \sum_{l=k+1}^K N_l$, 然后

$$\begin{aligned} \sum_{l=k+1}^K \mu_l \geq \sum_{l=k+1}^K N_l & \Rightarrow \mu_k + \sum_{l=k+1}^K \mu_l > N_k + \sum_{l=k+1}^K N_l \\ & \Leftrightarrow \sum_{l=k}^K \mu_l > \sum_{l=k}^K N_l \end{aligned}$$

这与引理中的条件(1)是矛盾的。

因此我们可以得到 $\sum_{l=k+1}^K \lambda_l > 0$, 并且由于 $\sum_{l=k}^K \mu_l \leq \sum_{l=k}^K N_l$, 那么

$$\sum_{l=k}^K \lambda_l = \lambda_k + \sum_{l=k+1}^K \lambda_l \geq 0$$

又因为 $C_k \leq C_{k+1} \leq \dots \leq C_K$, 我们得到

$$\begin{aligned} & \lambda_k \times C_k + \sum_{l=k+1}^K \lambda_l \times C_l \geq 0 \\ \Leftrightarrow & N_k \times C_k + \sum_{l=k+1}^K N_l \times C_l \geq \mu_k \times C_k + \sum_{l=k+1}^K \mu_l \times C_l \\ \Leftrightarrow & \sum_{l=k}^K \mu_l \times C_l \leq \sum_{l=k}^K N_l \times C_l \end{aligned}$$

满足引理。 □

根据以上引理, 我们可以建立对于拥有 L 个关键性级别的多级关键性系统 τ 的可调度性判定条件。

定理 3.5: 一个拥有 L 个关键性级别的多级关键性系统 τ 在 EDF 算法下是可调度的, 当以下条件满足:

$$\forall x_1 \geq \dots \geq x_L \geq 0: \sum_{l=1}^L \sum_{\tau_i \in \tau(l)} dbf_i^l(x_1, \dots, x_l) \leq x_1 \quad (3.12)$$

其中 $dbf_i^l(x_1, \dots, x_l)$ 由公式(3.10)计算得出。

证明: 我们通过反证法来证明该定理, 假设一个任务实例 J_i 在系统运行

于关键级 l 时的 t_d 时刻错失截止期, 用 t_l 表示系统从关键级 $l-1$ 转换到 l 时的时刻, t_b 是 t_d 之前最近的时刻, 满足在 $[t_b, t_d]$ 内的任意时刻, $\tau(l)$ 有至少一个截止期在 t_d 之前的活跃(释放但未完成)任务实例。如果我们将区间 $[t_b, t_d]$ 分成若干小份 $[t_1, t_2), [t_2, t_3), \dots, [t_\beta, t_d]$, 并且假设 $t_b \in [t_\alpha, t_{\alpha+1})$ (此处我们忽略 $t_b \in [t_\beta, t_d]$ 的情况, 因为该区间内没有关键级转换发生, 证明比较简单)。我们令 $x_l = t_d - t_l$ 。接下来, 我们需要证明关键性 γ 的任务子集 $\tau(\gamma)$ 中的任务 τ_j 在区间 $[t_1, t_d]$ 内的活跃实例工作量上界为 $\Delta = dbf_i^\beta(x_1, \dots, x_\beta)$, 其中 $x_1 = x_2 = \dots = x_\alpha$ 并且 $x_\gamma = x_{\gamma+1} = \dots = x_\beta$ 。如果这成立, 则必须满足 $\Delta > x_1$, 这跟定理中的(3.12)矛盾。我们假设在系统运行过程中 $\tau(\gamma)$ 里某个任务在 $[t_1, t_d]$ 内的工作量是大于 Δ 的, 假设实际 τ_j 在区间 $[t_1, t_d]$ 内的执行时间大于 C_i^l 小于等于 C_i^{l+1} 的实例个数为 μ_i^{l+1} , 那么 $\tau(\gamma)$ 总的工作量上界为 $\sum_{l=\alpha}^\gamma \mu_i^l \times C_i^l$ 。另一方面, 根据 dbf_i^l 的定义, $\Delta = \sum_{l=\alpha}^\gamma N_i^l \times C_i^l$ 。因此我们得到

$$\sum_{l=\alpha}^\gamma \mu_i^l \times C_i^l > \sum_{l=\alpha}^\gamma N_i^l \times C_i^l \quad (3.13)$$

令 $\mu = \sum_{l=\alpha}^\gamma \mu_i^l$ 以及 $N = \sum_{l=\alpha}^\gamma N_i^l$, 那么以下两个条件之一必须成立以满足(3.13): 但是, 我们接下来会证明在两种情况下都会出现矛盾。

(i) $\mu > N$ 。根据 N_i^l 的定义我们得知, $N = n_i(x_\alpha)$, 即释放时间和截止期都在长度为 x_α 的时间区间内的任务 τ_i 所能拥有的实例个数最大值。很显然 $\mu > N$ 不可能成立。

(ii) $\mu \leq N$ 。令 δ 为满足 $\mu_i^\delta \neq N_i^\delta$ 的最大的关键性级别。考虑两种情况: (1) $\mu_i^\delta > N_i^\delta$: 在这种情况下, $\overline{\mu}_i^\delta = \mu_i^\delta + \overline{\mu}_i^{\delta+1}$ 并且 $\overline{N}_i^\delta = N_i^\delta + \overline{N}_i^{\delta+1}$ 。根据 δ 的定义, 对于任意的 $j > \delta$, $\mu_i^j = N_i^j$, $\overline{\mu}_i^{\delta+1} = \overline{N}_i^{\delta+1}$ 成立。因此 $\overline{\mu}_i^\delta > \overline{N}_i^\delta$ 。利用引理3.1和3.2中同样的结论, $p_i^\delta(x_\alpha, x_\delta)$ 和 $q_i^\delta(x_\alpha, x_\delta)$ 可以作为 τ_i 在关键级 δ 或更高时执行所释放实例个数的上界, 因此 $\overline{N}_i^\delta = \min(p_i^\delta(x_\alpha, x_\delta), q_i^\delta(x_\alpha, x_\delta))$ 是一个安全的上界。所以 $\overline{\mu}_i^\delta > \overline{N}_i^\delta$ 不可能成立, 进而 $\mu_i^\delta > N_i^\delta$ 也不成立。(2) $\mu_i^\delta < N_i^\delta$: 根据引理3.4,

$$\sum_{l=\alpha}^\gamma \mu_i^l \times C_i^l \leq \sum_{l=\alpha}^\gamma N_i^l \times C_i^l$$

这跟(3.13)矛盾。

所以两种情况都会导致跟(3.13)矛盾, 因此假设 J_i 在系统运行于关键级 l 时的 t_d 时刻错失截止期不成立。□

3.2.3 复杂度分析

我们首先分析本章提出的可调度性分析方法在两级关键性系统下的复杂度, 即(3.8)中的判定条件。我们首先给出以下定义:

$$\begin{aligned} \mathcal{U}_{LO}^{LO} &= \sum_{\tau_i \in \tau(LO)} U_i^{LO}, \mathcal{U}_{HI}^{LO} = \sum_{\tau_i \in \tau(HI)} U_i^{LO}, & \mathcal{U}_{HI}^{HI} &= \sum_{\tau_i \in \tau(HI)} U_i^{HI} \\ \mathcal{C}_{LO}^{LO} &= \sum_{\tau_i \in \tau(LO)} C_i^{LO}, \mathcal{C}_{HI}^{LO} = \sum_{\tau_i \in \tau(HI)} C_i^{LO}, & \mathcal{C}_{HI}^{HI} &= \sum_{\tau_i \in \tau(HI)} C_i^{HI} \end{aligned}$$

作为在偶发任务集调度中的一个常见假设，我们假设系统总的利用率不会超过一个常数，该常数严格小于系统能够提供处理器资源的比例，即 $\mathcal{U}_{HI}^{LO} + \mathcal{U}_{LO}^{LO} \leq \varepsilon < 1$ 且 $\mathcal{U}_{HI}^{HI} \leq \varepsilon < 1$ 。以下定理给出了可调度行判定条件(3.8)中 x 和 y 取值的范围。

定理 3.6: 令 $x = y + z$ ，如果没有满足以下条件并且违反(3.8)中的判定条件的 (y, z) 存在，则(3.8)成立。

$$(1 - \mathcal{U}_{HI}^{HI}) \times y + (1 - \mathcal{U}_{HI}^{LO} - \mathcal{U}_{LO}^{LO}) \times z \leq \mathcal{C}_{LO}^{LO} + \mathcal{C}_{HI}^{LO} + \mathcal{C}_{HI}^{HI} \quad (3.14)$$

证明: 一个任务实例只有在 $[t_1, t_s]$ 区间内释放才能够在系统处于低关键性级别运行时执行，因此任务 τ_i 这类实例的工作量不会超过 $(\lceil \frac{t_s - t_1}{T_i} \rceil) \times C_i^{LO}$ ，即小于等于 $z \times U_i^{LO} + C_i^{LO}$ 。类似的，一个任务实例的截止期只有在 $(t_s, t_d]$ 内，它才能够在系统处于高关键性级别运行时执行，因此 τ_i 这类实例的工作量不会超过 $(\lceil \frac{t_d - t_s}{T_i} \rceil) \times C_i^{HI}$ ，即小于等于 $y \times U_i^{HI} + C_i^{HI}$ 。因此得到

$$\begin{aligned} \sum_{\tau_i \in \tau(LO)} dbf_i^{LO}(z) + \sum_{\tau_i \in \tau(HI)} dbf_i^{HI}(y+z, y) \leq \\ y \times \mathcal{U}_{HI}^{HI} + z \times (\mathcal{U}_{HI}^{LO} + \mathcal{U}_{LO}^{LO}) + \mathcal{C}_{LO}^{LO} + \mathcal{C}_{HI}^{LO} + \mathcal{C}_{HI}^{HI} \end{aligned}$$

由于(3.8)不能成立，即存在 y, z 满足

$$\begin{aligned} \sum_{\tau_i \in \tau(LO)} dbf_i^{LO}(z) + \sum_{\tau_i \in \tau(HI)} dbf_i^{HI}(y+z, y) \geq y+z \\ \Rightarrow y \times \mathcal{U}_{HI}^{HI} + z \times (\mathcal{U}_{HI}^{LO} + \mathcal{U}_{LO}^{LO}) + \mathcal{C}_{LO}^{LO} + \mathcal{C}_{HI}^{LO} + \mathcal{C}_{HI}^{HI} \geq y+z \end{aligned}$$

定理得证。 □

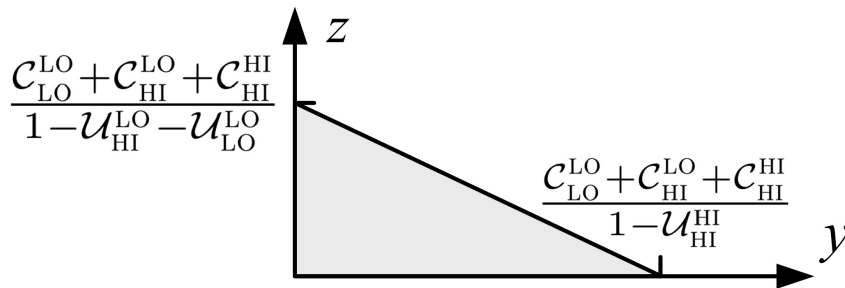


图 3.7 需要检查的 (y, z) 的范围说明

Fig. 3.7 Illustration of the region of (y, z) that needs to be checked.

图3.7所示为需要检查的 y 和 z 的范围。根据需求界限函数的定义， $dbf_i^{HI}(x, y)$ 是

一个关于 y 和 z 的非递减的阶梯状函数，因此，检查 $dbf_i^{HI}(x,y)$ 在图3.7中不可微分的点就足以用来判断(3.8)是否满足，并且该可调度性判定方法的复杂度为伪多项式时间。

同样的，我们可以通过定义以下符号得到对于多级关键性系统下相同的结论，

$$\forall j \leq l: \mathcal{U}_l^j = \sum_{\tau_i \in \tau(l)} U_i^j = \sum_{\tau_i \in \tau(l)} C_i^j / T_i^j,$$

$$\forall j \leq l: \mathcal{C}_l^j = \sum_{\tau_i \in \tau(l)} C_i^j$$

定理 3.7: 令 $z_j = x_j - x_{j+1}$ 以及 $z_L = x_L$ ，如果所有满足以下条件的 $\{z_1, \dots, z_L\}$ 并不违反(3.12)，则(3.12)一定成立：

$$\sum_{j=1}^l (1 - \sum_{i=j}^l \mathcal{U}_i^j) \times z_j \leq \sum_{j=1}^l (\sum_{i=j}^l \mathcal{C}_i^j) \tag{3.15}$$

该定理的证明同定理3.6的证明类似，所以在此省略。

对于两级关键性系统，需要检查的 y 和 z 的取值范围是一个二元区域， $\{z_1, \dots, z_L\}$ 的取值范围构成了一个 L 维空间。由于 L 是一个常数，使得计算复杂度依然维持在伪多项式时间，但在实际系统中，可能由于过大的 L 值，使得计算时间非常大。

为了解决这一问题，我们可以结合本章所提可调度性判定方法（即(3.12)）和EY方法，来平衡分析的精度和效率。具体来说，我们设定一个限制值 ζ 作为使用本章分析方法所包含的做多连续关键级的数量，例如，考虑一个包含4个关键性级别的多级关键性系统，设置 $\zeta = 2$ ，那么我们分别单独的分析系统处于前两个和后两个关键性级别的状态，当在分析后两个关键性级别时，我们再用EY方法来分析系统从第二个关键级跨越至第三关键级的工作量。

如果对于一个5级关键性系统并且设置 $\zeta = 3$ ，那么有两种组合关键性级别的方式：(1) $\{1,2\}$ 和 $\{3,4,5\}$ ；(2) $\{1,2,3\}$ 和 $\{4,5\}$ 。两种组合方式的选择及性能差别将在实验部分的3.4小结讨论。

3.3 调整虚拟截止期

在上一章节中我们介绍了本章所提出的针对EDF=VD算法调度的混合关键性系统的可调度性分析方法，EDF-VD调度算法要求为每个任务在系统处于不同关键级别下设置一个虚拟截止期，本节就提出一种新的算法来为每个任务设置虚拟截止期。

文章[36]中的一个重要发现是，缩短一个任务在关键级 l 下的虚拟截止期虽然

会降低系统在 l 关键性级别下的可调度性，但是有利于提升系统在更高关键级 $(l+1, l+2, \dots)$ 下的可调度性。因此，调整虚拟截止期的主要问题在于如何平衡系统在不同关键性级别下的可调度性。在[36, 37]中，EY方法单独的分析系统在不同关键性级别的可调度性，对于 l 级别的可调度性，EY需要考虑系统从前一关键性级别 $l-1$ 下带入的工作量。正因如此，EY的策略是从高到低地调整和设置每个关键性级别下任务的虚拟截止期，在每个关键性级别 j 下，调整 $j-1$ 关键级下任务的虚拟截止期，尽可能的减少带入的工作量，直到当前关键级 j 下系统可调度。

不同于EY方法，本章所提分析方法中计算的需求界限函数 dbf_i^j 包含了关键级 l 之前工作量，所以我们无法得知应当调整或者说缩短哪一个关键性级别下任务的虚拟截止期才能够提升当前级别的可调度性。

我们新的虚拟截止期调整方法的主要思想是设计有效的思路来选择哪一个关键性级别去调整任务的虚拟截止期。正如刚才所介绍的，缩短任务在关键性级别 j 下的虚拟截止期可能会损害系统在当前关键级别自身的可调度性，所以我们应当选取更“容易”调度（即工作量更低）的关键级别，去调整任务的虚拟截止期。我们用 $load(j)$ 来表示所有关键性级别高于 j 的任务贡献给系统关键性级别 j 的工作量：

定义 3.3 (负载): 定义负载 $load(j)$ 为

$$load(j) = \max_{\forall x > 0} \frac{\sum_{\tau_i \in \mathcal{U}_{k \geq j}} \tau_i(j) dbf_i^j(x)}{x}$$

其中 $dbf_i^j(x)$ 是用 C_i^j 作为最差执行时间计算的标准需求界限函数。

利用该负载的定义，我们设计了一种新的调整虚拟截止期的算法TuneSys(τ)，伪代码如算法1所示。TuneSys(τ)以降序的方式检测所有系统关键性级别下的可调度性，因此在调整虚拟截止期过程中的每一步，只要我们可以成功的为剩下的关键性级别设置可调度的虚拟截止期，那么就能保证已经调整过的这些关键性级别下系统是一定可调度的。一旦TuneSys(τ)发现系统 τ 在 l 关键级下是不可调度的，那么我们就从剩下的比 l 低的关键级里挑选具有最小负载的关键级 j 并且调用TuneMode(j)来调整该关键级别下任务的虚拟截止期。之后重新计算负载并重复以上过程直到系统在该关键级别下可调度。如果在某个关键性级别下，算法不能在任何更低关键级下找到能够缩短的虚拟截止期且不违反其可调度性($load(j) \geq 1$)，则算法返回失败。当且仅当任务集 τ 在所有关键级别下都可调度时，算法返回成功。

算法 1 TuneSys(τ)

```

for  $l \in \{L, \dots, 1\}$  do
    while  $\tau$  is not schedulable in mode  $l$  do
         $\forall j < l$ , calculate load( $j$ )
        if  $\forall j < l$ , load( $j$ )  $\geq 1$  then
            return FAILURE
        end if
        select  $j$  such that load( $j$ ) is minimal for  $\forall j < l$ 
        TuneMode( $j$ )
    end while
end for
return SUCCESS

```

3.4 实验

在本节中，我们设计随机任务集实验对多级关键性系统来评估本章所提可调度性分析方法的性能，即精确性及有效性。我们沿用[36, 37]中的方法来生成随机混合关键性任务集： P 是一个任务是更高关键性任务的可能性（如果 τ_i 是一个 j 关键性级别任务，则下一个随机生成的任务 τ_{i+1} 有 P 的可能性成为一个 $j+1$ 关键性任务， $1-P$ 的可能性成为一个 j 关键性任务）； O 是任务连续两个关键性级别下的最差执行时间的最大比率； L 是系统包含的总的关键性级别数量。对于每个任务 τ_i ， C_i^j 取值范围为 $[1, 10]$ ， T_i 取值范围为 $[10, 100]$ ，并且 $D_i = T_i$ 。任务集都是随机生成的，并且根据它们的平均资源利用率 \mathcal{U}_{avg} 分为若干组。

$$\mathcal{U}_{avg} = \sum_{j=1}^L \mathcal{U}^j / L, \quad \mathcal{U}^j = \sum_{\tau_i \in \tau} U_i^j$$

本节我们比较以下两种算法的性能：

- **New**。定理3.5中的可调度性判定方法，结合3.3节中的虚拟调整期调整算法。
- **EY**。文章[36]中的可调度性判定方法。

我们首先验证本章所提可调度性分析方法较EY精确度的提升。图3.8所示为不同平均利用率的三级（ $L=3$ ）关键性系统的系统接受率以及不同的 P 和 O 设置下的实验结果。对于每组平均利用率，我们都随机生成至少10000个任务集来测试之后取平均值，系统接收率的定义为被可调度性分析方法认定为可调度的任务集的

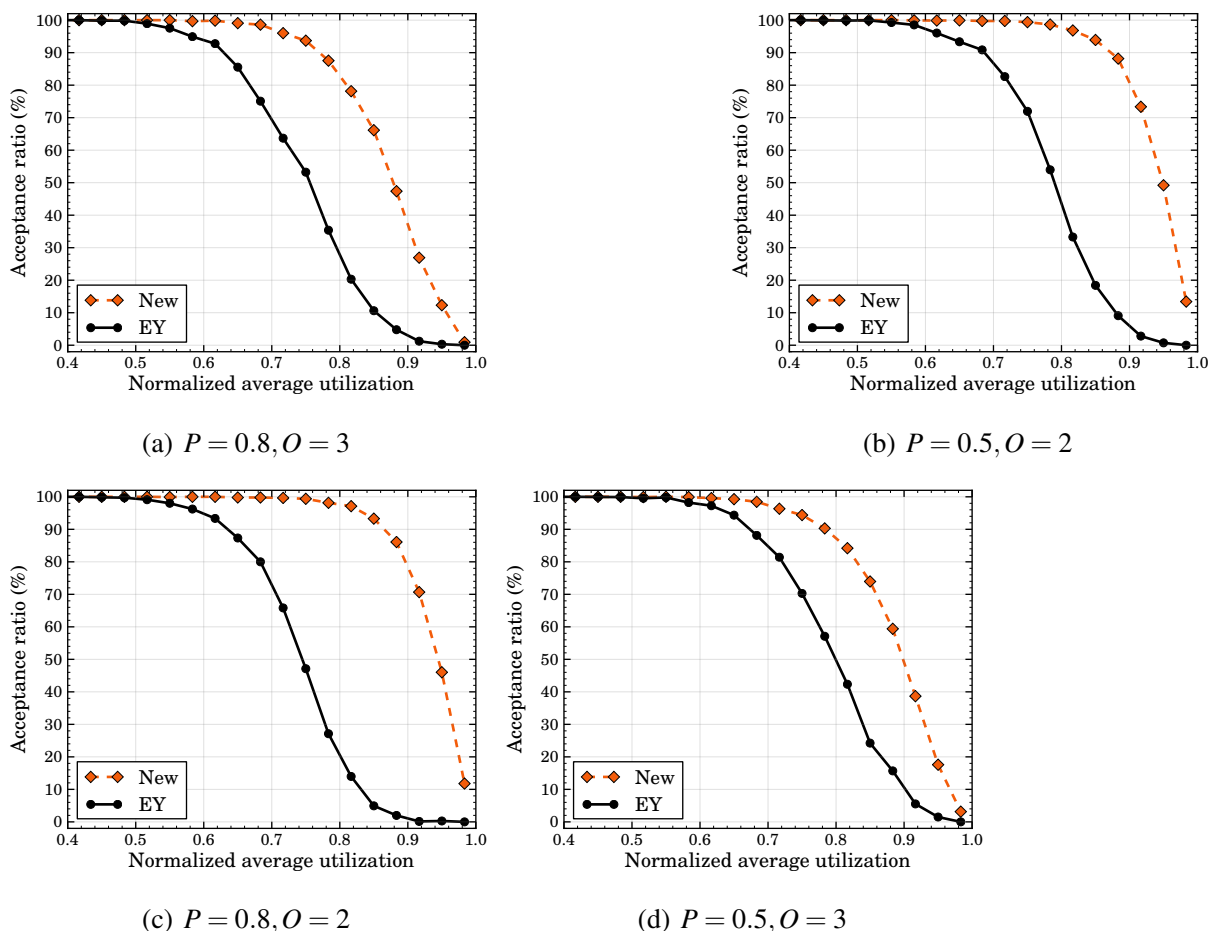


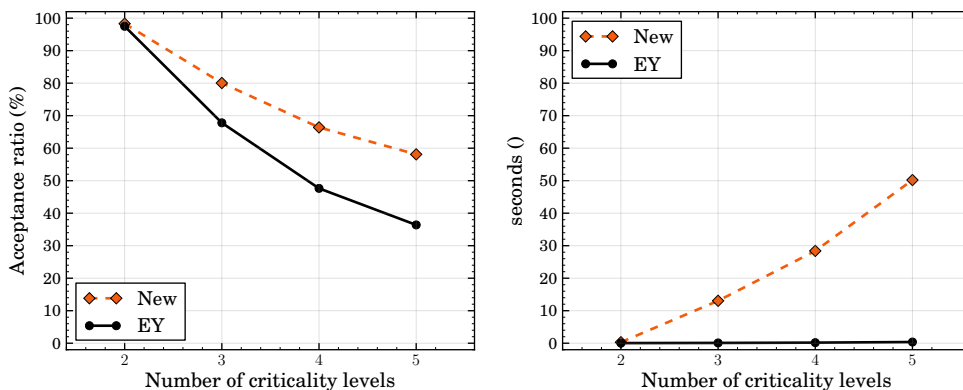
图 3.8 针对三级关键性系统不同 P 和 O 设置下的实验结果

Fig. 3.8 Experiment results with different P and O for triple-criticality task systems.

数量与总的测试任务集的数量之比。从图3.8我们可以看出，本章所提出的可调度性分析方法较EY方法在各种参数设置下都有明显的性能提升。

我们同样做了大量的实验来验证本章提出分析方法对于多级关键性系统的性能提升。图3.9-(a)所示为加权系统接受率随关键性级别数量变化（从2到5）的函数曲线，从图中我们可以看出，随着关键级数量的增加，使得系统可调度变得越来越难，但同时我们跟EY方法之间的性能提升也变得越来越明显。

对于本章所提方法来说，分析精度的提升是以更高的分析复杂度为代价的。图3.9-(b)所示为平均分析时间开销随关键性级别数量变化的函数曲线，可以看出，随着关键性级别数量增加，我们方法的时间开销变得越来越大，而EY方法的效率损失却非常低。对于拥有5级关键性的系统，分析每个任务集所花费的平均时间多达50秒，这对于大规模系统来说可能是无法接受的，尤其是关键性级别又非常多的系统。



(a) 系统接收率

(b) 时间开销

图 3.9 随关键性级别数量变化的分析精确度和效率

Fig. 3.9 Analysis precision and efficiency, with respect to the number of criticality levels.

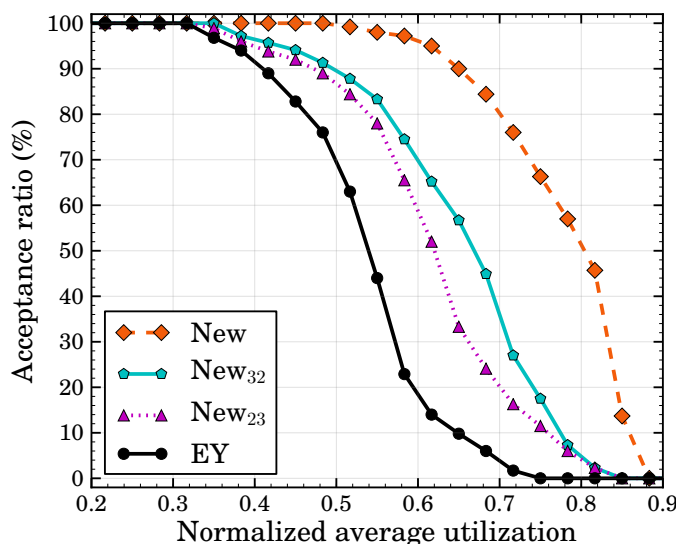


图 3.10 混合分析方法性能评估

Fig. 3.10 Evaluation of the hybrid analysis method with different criticality grouping strategies. $L = 5$, $P = 0.8$, $O = 3$.

为了解决这一分析复杂度问题，我们在3.2.3一节中介绍了一种结合EY和本章方法的混合可调度性分析方法，该方法通过设置一个使用本章新方法分析的最多连续关键性级别数量 ζ ，来平衡分析的精确度和效率。图3.10所示为对于拥有5级关键性系统，在设置 $\zeta = 3$ 时该混合分析方法的分析精度。图中，曲线“New”是原始的本章所提分析方法，连续的分析所有5个关键性级别下的系统可调度性，曲线“EY”代表原始的EY方法。曲线“New₂₃”和“New₃₂”分别是设置不同关键性级别分组策略（ $\{1, 2\} + \{3, 4, 5\}$ 和 $\{1, 2, 3\} + \{4, 5\}$ ）下的混合分析方法。从实验中我们可以看出，两种混合分析方法并没有绝对的孰优孰劣（存在一些任务集可以被其一所调度而不能被另外一种判定为可调度，反之亦然），但是平均来看，使用本章所提

方法连续分析较低关键性级别数量较多的混合方法具有更高的性能。

3.5 小结

在本章中，我们针对EDF-VD调度下的混合关键性系统提出一种新的可调度性分析方法，该分析方法较当前最优的EY算法提高了分析的精度，尤其对于具有大于两个关键性级别的多级关键性系统。该方法精确性的提升是以提高分析复杂度为代价的，为了解决这一复杂度问题，我们同样提出了一种结合本章所提算法和EY算法相结合的混合分析方法，来平衡分析的精确度和效率。我们使用随机生成的任务集来评估本章所提算法的性能，实验结果也验证了其精确性。

第4章 严格周期任务系统的开始时间配置策略

略

时间驱动调度 (Time-Triggered Scheduling) 是一种非常适用于硬实时系统设计的方法。时间驱动系统较事件驱动系统来说具有很多优势, 例如更容易理解和分析系统^[137]。并且, 时间驱动范式支持确定性的系统时间行为分析。

严格周期调度^[138]就是一种关于严格周期任务的时间驱动调度, 其中严格周期任务的每两个连续释放的任务实例执行时间间隔严格等于该任务的周期。严格周期任务模型的研究是受启发于控制系统中的算法通常要求任务具有严格采样和执行周期, 因此, 反复不断释放的任务都要求以一种严格周期的模式来执行, 以提供严格的控制逻辑, 保证系统的高可靠性。相反的, 传统的周期性任务模型, 由于会在执行过程中产生较大的抖动, 会大大的降低系统的控制性能^[139], 因此通常不能被采用。在航空电子系统中, 一个关键子系统中出现的微小的运行错误就可能造成灾难性的后果。所以, 根据航空领域的ARINC 653标准^[140], 严格周期特性是整合模块化航空电子系统 (Integrated Modular Avionics, IMA) 调度器所考虑的一个主要限制条件^[141]。

为了调度一个严格周期实时任务, 我们需要为它的首次释放的任务实例设置一个开始执行时间, 随后该任务之后释放的所有实例的执行时间都是确定且已知的, 因为它们都会按照严格周期重复的执行。所以, 要想调度一个严格周期实时任务集, 我们需要为系统中的每个任务都设置一个开始执行时间使得系统是可调度的, 即任意两个任务的执行不会发生冲突。在文章^[47, 48]中, 作者已经证明了在多核处理器上为一个严格周期任务集分配开始执行时间是强NP难的, 并且通过对三划分问题的规约, 可以证明即使对于只有一个处理器的情况, 该问题仍然是强NP难的。大致上, 该问题的最优解需要遍历系统中所有任务开始时间的组合, 复杂度是非常高昂的。随后, 在最近的一篇研究严格周期任务系统的文章^[142]中, 作者提出一种不涉及任务开始执行时间配置的可调度性判定方法, 使得系统设计者能够在不知道具体的调度策略时也能测试系统的可调度性, 即是否存在一个调度表, 能够同时满足系统中所有任务的截止期和严格周期特性。然而, 在严格周期系统的设计过程中, 仅仅直到系统是否可调度是远远不够的, 更重要的是知道系统如何才能成功调度, 即为系统中的任务配置合理的开始执行时

间。因此，在本章中，我们设计有效的次优的算法来为严格周期任务系统配置开始执行时间。我们的算法通过探索任务周期之间的关系来提升为任务成功配置开始执行时间的可能性。

4.1 预备知识

4.1.1 系统模型

在本节中，我们定义本章所研究的不可抢占隐含截止期（周期等于截止期）严格周期任务系统模型。一个系统 τ 由若干不相关的反复释放执行任务组成，每个任务 $\tau_i \in \tau$ 可以表述为一个三元组 $\langle T_i, C_i, s_i^1 \rangle$ ：

- T_i 是 τ_i 的周期。
- $C_i \in \mathbb{N}^+$ 是 τ_i 的最差情况执行时间。
- $s_i^1 \in \mathbb{N}^+$ 定义为 τ_i 开始执行时间，并且 $0 \leq s_i^1 \leq T_i$ 。

在系统运行过程中，每个任务 τ_i 会释放无限多个任务实例，我们用 J_i^k 来表示 τ_i 释放的第 k 个实例， s_i^k 是实例 J_i^k 的开始执行时间。那么，

$$s_i^k = (k-1) \cdot T_i + s_i^1$$

为了描述的简洁，当上下文清晰时，我们直接用 J_i 来表示 τ_i 的一个实例，用 s_i 来表示 J_i 的开始执行时间。我们定义任务 τ_i 和任务集 τ 的利用率为：

$$U_i = C_i/T_i, U_\tau = \sum_{\tau_i \in \tau} U_i$$

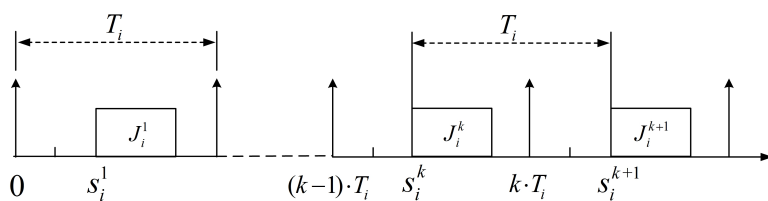


图 4.1 严格周期任务示例

Fig. 4.1 Illustration of a strictly periodic task.

不可抢占严格周期任务系统具有以下语义。如图4.1所示， τ_i 在0时刻释放的第一个实例 J_i^1 在 s_i^1 时刻开始执行，并且连续执行 C_i 时刻不被打断或抢占，之后，在每一个 τ_i 随后的周期内，在 s_i^k 时刻释放的实例 J_i^k 在 s_i^k 时刻开始执行并且在 $s_i^k + C_i$ 时刻完成执行。

在本文中，我们假设时间是离散的，当描述一个任务实例 J_i^k 在 s_i^k 时刻开始执行时，意味着它将会在时间区间 $[s_i^k, s_i^k + 1)$ 的开始时刻被处理器调度执行。并且系统

中所有任务都是不相关的，也就是说，任务之间没有任务前后依赖关系。此外，虽然本章利用隐含截止期任务模型来描述所提出的开始时间分配算法，但该工作可以轻易地更改之后应用于限制截止期以及任意截止期模型中。总的来说，本章的目的是提出一种有效的方法来为严格周期任务集 τ 找到一种调度策略，该调度表的建立是基于通过分析系统的行为所为每个任务分配的开始执行时间。

4.1.2 现有分析方法

在文章[47, 48]中，Korst等人证明了在单处理器上对于一个给定的严格周期任务集，判断是否存在一个可行的调度策略使得系统可调度的问题是强NP难的。此外，他们提出了一种判断两个任务是否可调度的充分必要判定条件，可描述为以下定理：

定理 4.1: 两个严格周期任务 τ_i 和 τ_j ，开始执行时间分别为 s_i^1 和 s_j^1 ，他们是可调度的当且仅当以下条件满足：

$$C_i \leq (s_j^1 - s_i^1) \bmod \gcd_{i,j} \leq \gcd_{i,j} - C_j \tag{4.1}$$

其中， $\gcd_{i,j} = \gcd(T_i, T_j)$ 。

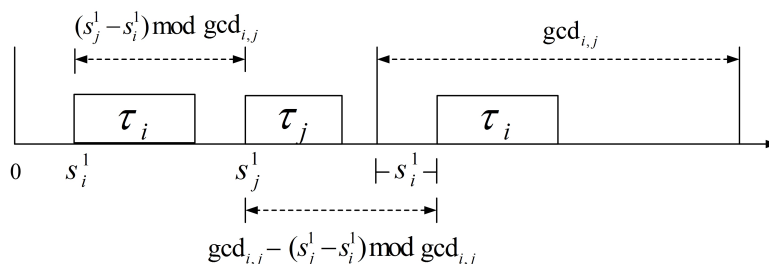


图 4.2 定理4.1示例

Fig. 4.2 Explanation of Theorem 4.1.

$(s_j^1 - s_i^1) \bmod \gcd_{i,j}$ 描述了任务 τ_i 和 τ_j 的执行之间的最小距离，如图4.2所示，该距离代表了 τ_i 不能干涉 τ_j 的执行的执行的时间区间，且其长度必须大于等于 τ_i 的执行时间 C_i 。另一方面，相似的，该不等式的右半部份保证了 τ_i 在下一个长度为 $\gcd_{i,j}$ 的区间内也不会干涉 τ_j 的执行。需要注意的是，图中的方块仅代表两个任务之间的冲突关系，而不是其实际的执行时间。具体的，假设 τ_i 的开始执行时间为 s_i^1 ，在每个 $\gcd_{i,j}$ 区间内 s_i^1 起始的长度为 C_i 的时间区间都不能用来执行 τ_j ，任务 τ_i 和 τ_j 的顺序并不影响条件(4.1)（因为 $-a \bmod b = (b - a) \bmod b$ ），定理4.1的证明读者可参考文章[48]。

条件4.1可以通过牺牲一定的复杂度，检查所有两两任务之间的可调度性而扩

展到应用于一个严格周期任务集的可调度行判定。当每个任务的开始时间没有确定的时候，我们可以遍历任务集中所有任务开始时间的组合，也就是求解以下线性不等式组：

$$\begin{cases} 0 \leq s_i^1 \leq T_i - C_i & \forall \tau_i \in \tau \\ C_i \leq (s_j^1 - s_i^1) \bmod gcd_{i,j} \leq gcd_{i,j} - C_j & \forall \tau_i, \tau_j \in \tau \end{cases} \quad (4.2)$$

条件(4.2)是针对严格周期任务集的精确的可调度性判定方法，任意以上不等式组的解都能够生成一个可行的调度表使得系统可调度，即所有任务满足它们的截止期和严格周期约束。相反的，如果不等式组无解，则不存在使得系统可调度的可行调度表。

很容易看出，每个任务开始执行时间的取值范围是 $0 \leq s_i^1 \leq T_i - C_i$ ，整个任务集就会产生

$$\prod_{\tau_i \in \tau} (T_i - C_i + 1)$$

种可能的开始执行时间组合。组合的数量是随着任务集中任务个数呈指数级增长的，因此简单地遍历所有可能的组合是非常不明智的选择。

为了解决这一问题，Kermia在[142]中提出了一种建立在非可构造性证明上的充分可调度性判定条件，该判定条件不需要确定每个任务具体的开始执行时间，就能够允许系统设计者检测是否存在调度表使得系统可调度。首先，文章提出一个称为“严格周期利用率因子”(Strict-Periodic Utilization Factor)，符号为 S_i ，表示使得任务 τ_i 可调度且不妨碍已经可调度的其他任务的执行所需要的时间比率的总和。 S_i 的计算公式为：

$$S_i = \sum_{\tau_k \in \tau_{sched}} \frac{C_k}{gcd_{i,k}} + \frac{C_i}{T_i} \quad (4.3)$$

S_i 的计算由两部分所构成，左半部分对所有已经调度的任务 τ_k 的 $\frac{C_k}{gcd_{i,k}}$ 进行累加，该部分包含了用来执行所有已经被调度任务的时间比率以及所有 τ_i 不能执行否则会跟已经调度的任务发生冲突的时间区间的比率。右半部份则是执行 τ_i 所需的时间比率，即在每个周期 T_i 内需要至少 C_i 单位的时间。

利用“严格周期利用率因子”就可以建立可调度性测试，其主要思想是在每一步从未调度的任务里挑选一个可以被调度的任务（检查 S_i 是否超过1）放进可调度任务子集中，直到原始任务集中所有任务都在可调度集合中或者在某一步不能找出任一任务可调度。该可调度性判定的一个问题是，在判定过程中的每一步，都是随机挑选任意一个能够通过可调度性测试的任务放进可调度任务子集中，该

决策一旦做出便不再更改，也就是说该过程不可回溯。就是这种随机挑选任务进行判定的方法导致了判定的结果非常的悲观，并且该判定方法并不能够为系统设计者提供具体的调度方案，也就是任务开始执行时间配置，仅能够表明是否存在一个可行的调度方案。与此不同，本章所提出的方法不仅能够为每个任务配置具体的开始执行时间，而且可调度性判定精度也大大提升。

4.2 开始执行时间配置方法

正如我们在上一小节提到的，[142]中随机挑选任务进行测试的方法可能会轻易地导致错失可行的开始执行时间配置方案。为了解决这一问题，我们提出一种新的挑选任务的方法以获得以下两方面的提升：首先，设计一种新的可调度性测试算法，该方法对每个选取的任务具有清晰的可调度性判定思路；其次，提出一种合理的任务选取方法使得我们的可调度性测试算法更加有效。

4.2.1 可调度性测试算法

在介绍我们提出的可调度性测试算法前，我们先定义冲突区间和可行区间，然后再介绍对一个严格周期任务 τ_i 的充分可调度性判定条件。

定义 4.1 (冲突区间): 任务 τ_i 的冲突区间，表示为 CI_i ，是在其第一个释放周期内不能用来执行 τ_i 的所有时间区域。否则 τ_i 的执行一定会跟已经调度的任务在某一时刻发生冲突。

定义 4.2 (可行区间): 任务 τ_i 的可行区间，表示为 AI_i ，是在其第一个释放周期内除去冲突区间之后剩余的时间区域。

令 CI_i^k 表示 τ_i 在其第一个释放周期内由已经调度的任务 τ_k 所引起的冲突区间， CI_i^k 由两部分组成，第一部分是从 s_k^1 起始的长度为 C_k 的时间区域，该区域由于是 τ_k 的执行区间，所以如果我们执行 τ_i 则会立即发生冲突。第二个部分包含所有从 $s_k^1 + m \cdot gcd_{i,k}, m = 0, 1, \dots$ 起始的长度为 C_k 的时间区域，这些区域同样也不能用来执行 τ_i ，否则， τ_i 和 τ_k 的执行一定会在将来的某一时刻发生冲突。我们接下来就证明这一性质。

证明: 假设任务 τ_i 和 τ_k 的开始执行时间分别为 s_i^1 和 s_k^1 ，它们分别在其每个释放周期的 $t_i = s_i^1 + n_i \cdot T_i$ 和 $t_k = s_k^1 + n_k \cdot T_k$ 区域内执行， $n_i, n_k \in \mathbf{Z}$ 。因为 $T_i = n_1 \cdot gcd_{i,k}$ 并且 $T_k = n_2 \cdot gcd_{i,k}$ ， $n_1, n_2 \in \mathbf{Z}$ 。我们得到 $t_i = s_i^1 + n_i \cdot (n_1 \cdot gcd_{i,k})$ 以及 $t_k = s_k^1 + n_k \cdot (n_2 \cdot gcd_{i,k})$ 。此外， $t_i = s_i^1 + n'_i \cdot gcd_{i,k}$ 并且 $t_k = s_k^1 + n'_k \cdot gcd_{i,k}$ ， $n'_i, n'_k \in \mathbf{Z}$ 。如果 $s_i^1 = s_k^1 + m \cdot gcd_{i,k}$ ，那么当 $m = n'_k - n'_i \in \mathbf{Z}$ 时， $t_i = t_k$ 。□

因此，我们可以得到

$$CI_i^k = \bigcap_{m=0, \dots, \frac{T_i}{gcd_{i,k}}} [s_k^1 + m \cdot gcd_{i,k}, s_k^1 + C_k + m \cdot gcd_{i,k}) \cap [0, T_i), \quad (4.4)$$

例 4.1: 考虑一个严格周期任务集 $\tau = \{\tau_1, \tau_2\}$, $(T_1, C_1) = (4, 1)$, $(T_2, C_2) = (6, 1)$ 。 τ_1 已经被认定为可调度且其开始执行时间 $s_1^1 = 0$ 。我们令 $s_2^1 = s_1^1 + m \cdot gcd_{1,2}$ 并且取一任意整数 $m = 3$ 。那么, $s_2^1 = 0 + 3 \cdot 2 = 6$ 。这导致两个任务的执行会在时刻 12, 也就是 τ_1 的第 3 个释放周期和 τ_2 的第 2 个释放周期, 发生冲突。

根据之前定义的冲突区间和可行区间, 我们得到

$$CI_i = \bigcap_{\tau_k \in \tau_{sched}} CI_i^k, \quad (4.5)$$

其中 τ_{sched} 代表已经被调度的任务子集, 并且

$$AI_i = [0, T_i) - CI_i \quad (4.6)$$

定理 4.2: 一个开始执行时间为 $s_i^1 \in [0, T_i - C_i]$ 的严格周期任务 τ_i 是可调度的, 当以下条件满足:

$$[s_i^1, s_i^1 + C_i) \subseteq AI_i. \quad (4.7)$$

证明: 我们用反证法来证明定理 4.2, 假设开始执行时间为 s_i^1 的任务 τ_i 满足定理中的条件 (4.7) 但是实际是不可调度的。如果 τ_i 不可调度, 那么一定是由于以下两个原因之一导致。

- (1) 面对已经被分配了开始执行时间的可调度的任务, 无法找到一个时刻能分配给 τ_i 来让其开始执行。
- (2) 存在一个时刻 s_i^1 能分配给 τ_i 来让其开始执行, 但是 s_i^1 之后并没有足够长的时间 (即大于等于 C_i) 来完成 τ_i 的执行。

我们假设在 τ_i 之前, 子集 $\{\tau_1, \dots, \tau_{i-1}\}$ 中的任务都已经被调度, 而子集 $\{\tau_{i+1}, \dots, \tau_n\}$ 将在 τ_i 之后被判定是否可调度, 即能否配置可行的开始执行时间。

对于第一种情况, 无法找到一个时刻让 τ_i 开始执行意味着其第一个周期内的所有时刻都已经被 $\{\tau_1, \dots, \tau_{i-1}\}$ 中的任务所占据。因此, 在 $[0, T_i)$ 内, $\{\tau_1, \dots, \tau_{i-1}\}$ 中的任务 τ_k 所需要的且不与 τ_i 冲突的时间区域为:

$$CI_i^k = \bigcap_{m=0, \dots, \frac{T_i}{gcd_{i,k}}} [s_k^1 + m \cdot gcd_{i,k}, s_k^1 + C_k + m \cdot gcd_{i,k}) \cap [0, T_i)$$

因此, 所有 $\{\tau_1, \dots, \tau_{i-1}\}$ 中的任务所需的总的时间区域为

$$CI_i = \bigcap_{\tau_k \in \{\tau_1, \dots, \tau_{i-1}\}} CI_i^k.$$

那么，无法为 τ_i 找到可以开始执行的时刻意味着

$$AI_i = \emptyset.$$

因为 τ_i 满足条件(4.7)并且 $[s_i^1, s_i^1 + C_i)$ 不是一个空集，那么这与我们的假设矛盾。

对于第二种情况，这意味着 τ_i 可行区间 AI_i 内的每一个时间区间的长度都小于 τ_i 的最差执行时间 C_i ，这也与条件(4.7)矛盾。所以，两种情况都会导致矛盾，那么如果 τ_i 满足定理中的条件，它一定是可调度的。□

本章针对严格周期任务系统 τ 所提出的新的可调度性判定算法，表示为STSP(τ)，如算法2中的伪代码所示。初始的，任务集 τ 中的所有任务被复制到集合 Γ_1 中，算法的输出则是一个可调度的任务集合 Γ_2 包含 Γ_1 中所有的任务且为其分配相应的开始执行时间。

算法的第一步，我们首先计算 τ 中所有任务的最大公约数 g 。显然，如果 $g = 1$ ，也就是至少存在两个任务它们的周期互质，我们可以轻易的推出该任务集无论如何不可调度，这一结论同样可以由条件(4.1)推出。当确认了系统中不包含互质的任务周期后，下一步我们循环的检查每个任务的可调度性。每一步，我们从 Γ_1 中选择一个合适的任务 τ_i 执行以下步骤（MS(Γ_1)表示我们提出的任务选择策略，具体的细节我们在下一小节中详细介绍），我们首先检查在若干任务已经被调度的情况下，是否存在任务时刻能够使得 τ_i 开始执行，即考虑所有 Γ_2 中的任务并且分析它们的执行行为，如果 τ_i 的可行区间 AI_i 不是空集，根据 AI_i 的定义，这意味着存在一个时刻 s_i^1 可以用来开始执行 τ_i 。但一个存在的 s_i^1 并不能够决定一个 τ_i 的可行的开始时间存在，因为我们需要检查 s_i^1 之后是否有足够长的空闲时间可以让 τ_i 不间断的执行其最差执行时间，为此，我们检查 AI_i 内包含的所有时间区间，看是否存在一段连续的长度大于 C_i 的区间。如果存在，那么这段连续区间可以用来执行 τ_i ，其起始时刻就是 τ_i 的开始执行时间 s_i^1 。注意，如果 AI_i 中包含多个满足条件的时间区域，我们总是选择首次出现的，即最早的一段区间。这是由于，一方面，我们无法预知还未被调度的任务的信息来找到一个最优的合法区间；另一方面，我们设计的任务选择方法能够帮助我们接近最优的结果。通过了以上所有的测试之后， τ_i 就能被安全的放进可调度任务子集 Γ_2 中。如果 τ_i 不能满足以上任意一个条件，算法返回失败。否则，算法返回可调度任务集 Γ_2 包含系统中所有的任务。

例 4.2：我们通过考虑一个严格周期任务集 $\tau = \{\tau_1, \tau_2, \tau_3\}$ ，其中 $(T_1, C_1) = (4, 1)$ ， $(T_2, C_2) = (6, 1)$ 以及 $(T_3, C_3) = (8, 1)$ ，来说明算法2执行的过程，尤其是为某一个任务

算法 2 STSP(τ)

```

1:  $\tau \leftarrow \{\tau_1, \dots, \tau_n\}$ 
2:  $\Gamma_1 \leftarrow \{\tau\}, \Gamma_2 \leftarrow \emptyset$ 
3:  $g \leftarrow gcd_{1, \dots, n}$ 
4: if  $g = 1$  then
5:   return FAILURE
6: end if
7: while  $\Gamma_1 \neq \emptyset$  do
8:    $\tau_i \leftarrow MS(\Gamma_1)$ 
9:    $AI_i \leftarrow [0, T_i)$ 
10:   $s_i^1 \leftarrow -1$ 
11:  for  $\tau_k \in \Gamma_2$  do
12:     $CI_i^k$  and  $CI_i$  are computed by equation (4.4) and (4.5) respectively
13:     $AI_i \leftarrow AI_i - CI_i$ 
14:  end for
15:  if  $AI_i = \emptyset$  then
16:    return FAILURE
17:  else
18:    for  $t \in [0, \dots, T_i - C_i]$  do
19:      if  $[t, t + C_i) \subseteq AI_i$  then
20:         $s_i^1 \leftarrow t$ 
21:      end if
22:    end for
23:  end if
24:  if  $s_i^1 = -1$  then
25:    return FAILURE
26:  else
27:     $\Gamma_1 \leftarrow \Gamma_1 \setminus \{\tau_i\}$ 
28:     $\Gamma_2 \leftarrow \Gamma_2 \cup \{\tau_i\}$ 
29:  end if
30: end while
31: return  $\Gamma_2$ 

```

赋开始执行时间的操作。我们假设任务 τ_1 和 τ_2 都已经被调度, 并且 $s_1^1 = 0$, $s_2^1 = 1$ 。当前我们需要检测任务 τ_3 能否被调度。

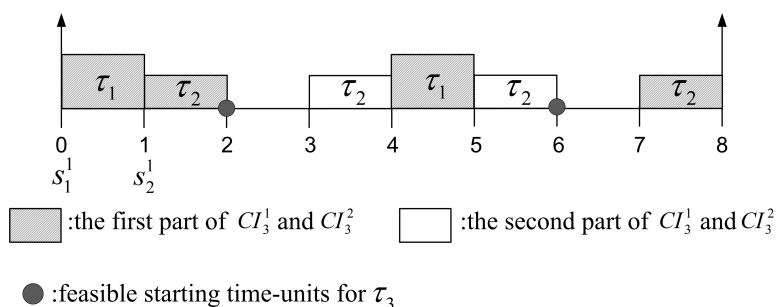


图 4.3 τ_3 的开始时间分配

Fig. 4.3 Start time instant assignment for τ_3 .

为了在 τ_3 的第一个释放周期 $[0,8)$ 内找到一个时刻使其能够开始执行, 我们首先检查所有由 τ_1 和 τ_2 引起的冲突区间, 因为这两个任务的开始执行时间都是已知的, 我们可以直接用公式(4.5)和(4.4)来检测, 并且得到 $CI_3 = CI_3^1 \cap CI_3^2 = \{[0,2), [3,6), [7,8)\}$ 。如图4.3所示, τ_3 的可行区间 $AI_3 = \{[2,3), [6,7)\}$, 那么可行的开始时刻为时刻2和时刻6, 但正如我们之前所介绍的, 我们直接选择最早出现的时刻2。最终, 我们为 τ_3 分配的开始执行时间 $s_3^1 = 2$, 并且将 τ_3 放入可调度任务子集 Γ_2 中。

本章所提出的可调度性测试算法充分而非必要的判定条件, 这主要是由于以下原因。在循环过程中的每一步针对某个任务所做出的可调度性决策都是不可更改的, 这可能会导致错失最优解。其次, 在为每个任务选择开始执行时间时, 我们直接选取首次出现的时间区间, 这也导致错失了一部分的解空间。我们提出的算法能否为整个系统找到一个可调度的开始执行时间配置, 很大一部分程度取决于我们在算法执行过程中的任务选择顺序。接下来, 我们就提出一种任务选择方法, 大大地提升算法找到可行解的可能性。

4.2.2 任务选择策略

在上一小节中, 在算法STSP(τ)中我们没有介绍的一部分内容就是任务选择方法MS(τ) (第8行), 在详细介绍MS(τ)之前, 我们首先讨论MS(τ)在可调度性测试中所扮演的角色, 以下的例子说明了对于同样的任务集, 不同的任务选择方法所导致的不同的可调度性结果。

例 4.3: 考虑一个严格周期任务集 $\tau = \{\tau_1, \tau_2, \tau_3\}$, 其中 $(T_1, C_1) = (4, 1)$, $(T_2, C_2) =$

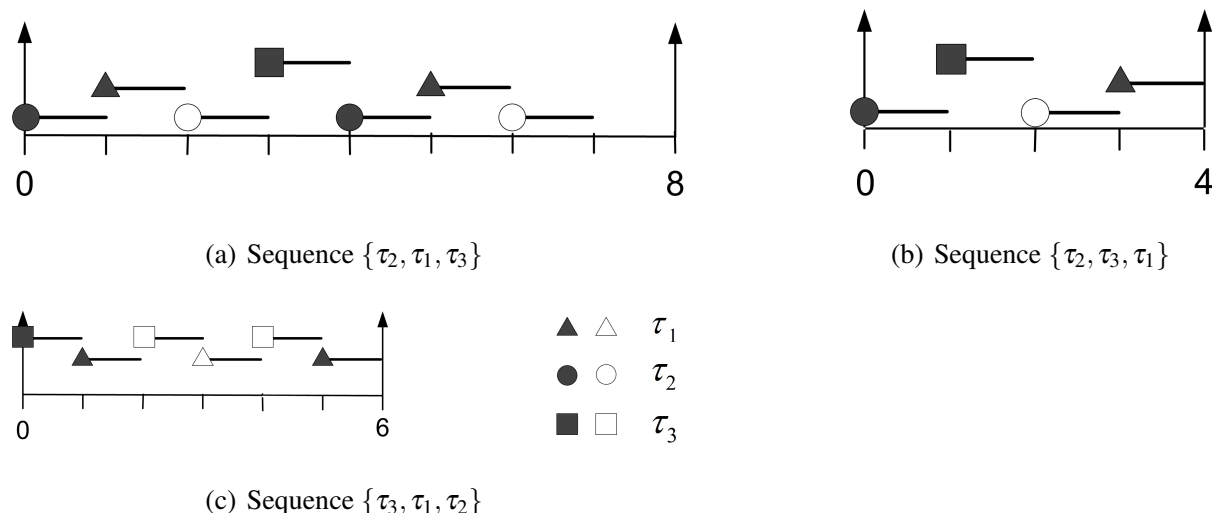


图 4.4 任务选择顺序

Fig. 4.4 Task selection sequence.

$(6, 1)$ 以及 $(T_3, C_3) = (8, 1)$ 。我们假设三种不同的任务选择顺序： $\{\tau_2, \tau_1, \tau_3\}$ ， $\{\tau_2, \tau_3, \tau_1\}$ 和 $\{\tau_3, \tau_1, \tau_2\}$ 。在其中每一种选择顺序中，前两个任务都是可调度的，而我们的任务是检查能否为最后一个任务找到一个可行的开始执行时间使其可调度。图4.4-(a)，4.4-(b)和4.4-(c)分别展示了在每种任务选择顺序中为最后一个任务设置开始执行时间的图示。根据冲突区间 CI_i 和可行区间 AI_i 的计算公式，我们得到：

$$\{\tau_2, \tau_1, \tau_3\}: CI_3 = \{[0, 3), [4, 7)\}, AI_3 = \{[3, 4), [7, 8)\};$$

$$\{\tau_2, \tau_3, \tau_1\}: CI_1 = \{[0, 3)\}, AI_1 = \{[3, 4)\};$$

$$\{\tau_3, \tau_1, \tau_2\}: CI_2 = \{[0, 6)\}, AI_2 = \emptyset.$$

在第三种情况中，任务 τ_2 的严格周期特性不能被满足，因为在其周期 $[0, 6)$ 内所有的时间区间都已经被 τ_1 和 τ_3 直接或间接的占据（因此图4.4-(c)中没有任务 τ_2 所对应的实心圆符号）。这意味着如果任务 τ_2 被最后一个选取，系统则被判定为不可调度。然而，显然系统在第一种和第二种任务选择顺序下是可调度的。

从以上例子可以看出，对于同样的严格周期任务集，不同的任务选择顺序会导致不同的可调度性判定结果，这促使我们设计一种更好的任务选择策略以提高可调度性判定精度。从图4.4-(c)中可以看出，任务 τ_2 之所以找不到一个可行的开始执行时间是由于 $gcd_{2,1}$ 和 $gcd_{2,3}$ 的值都很小（等于2），那么我们必须每个 $gcd_{2,1}$ 和 $gcd_{2,3}$ 内预留 C_1 和 C_3 个时间单位来防止冲突发生。然而，这种预留是很悲观的，因为任务 τ_1 和 τ_3 的最大公约数 $gcd_{1,3}$ 的值很大，这两个任务在 $gcd_{2,1}$ 和 $gcd_{2,3}$ 内一定会发生重合。这一观察促使我们进一步探寻 $gcd_{i,j}$ 的含义来帮助设计任务选择

策略。

定义 4.3 (谐波链 $\mathcal{H}(p)$ ^[143]): 谐波链 $\mathcal{H}(p)$ 是一个任务集, 其中所有任务都是谐波任务, 即周期都为 p 的整数倍。

谐波链 $\mathcal{H}(p)$ 形象化的描述了其中所有任务周期的信息, 即对于所有 $\tau_i \in \mathcal{H}(p)$, $T_i = m \cdot p, m \in \mathbb{N}^+$ 。如果按照周期从小到大递增的顺序 ($i < j \Leftrightarrow T_i \leq T_j$) 选择任务, 我们可以轻易地推出 $\forall \tau_i \in \mathcal{H}(p), \gcd_{1,i} = T_1 = p$, 其中 T_1 是谐波链中最小的任务周期。相应的, τ_i 和任意一个任务 τ_j ($T_j \geq T_i$) 之间的 $\gcd_{i,j}$ 随着 i 的增大而增大。此外, 从例子4.3中我们观察到, 一个数值小的 $\gcd_{i,j}$ 不利于 τ_i 在已调度任务 τ_j 之后的可调度性。因此, 任务之间 \gcd 数值比较小的应当优先被调度, 此时处理器的资源相对充足。根据这一观察以及谐波链中 \gcd 的特性, 我们希望在任务选择过程中 $\gcd_{i,j}$ 的数值逐渐增大, 这对最终任务集的可调度性是有利的。通过以上讨论, 我们得到在任务选择策略 $\text{MS}(\tau)$ 中的第一条原则。

原则1: $\text{MS}(\tau)$ 在同一条谐波链中每次都选择周期最小的任务。

这条原则规定了在一条谐波链中应当选择哪一个任务作可调度性判定。然而, 不失一般性的, 一个严格周期任务集可能包含一些任务可以被归类在不同的谐波链中, 那么问题来了, 当任务集中包含不止一条谐波链时, $\text{MS}(\tau)$ 如何选择任务。

假设两条谐波链, $\mathcal{H}(p), \mathcal{H}(q) \subseteq \tau (p \neq q)$, 同时存在于任务集中, 并且任务 $\tau_i \in \mathcal{H}(p)$ 。注意, 两条谐波链可能会有交叉, 即它们的交集不为空集。在这种情况下, 我们将两条谐波链交集集中的任务分组到包含更多任务的那一条谐波链中。例如, 考虑四个任务, 它们的周期分别为4, 6, 8和12。我们令周期为12的任务属于谐波链 $\mathcal{H}(4)$ 而不是 $\mathcal{H}(6)$, 即使12也是6的整数倍。这里我们假设 $\mathcal{H}(p)$ 比 $\mathcal{H}(q)$ 包含更多的任务, 我们可以推出 τ_i 和 $\mathcal{H}(p)$ 中的任何一个任务的 \gcd 值具有很高的可能性比 τ_i 和 $\mathcal{H}(q)$ 中的任意一个任务的 \gcd 值更大。那么, 我们得到在任务选择策略 $\text{MS}(\tau)$ 中的第二条原则。

原则2: 当系统中存在多条谐波链时, $\text{MS}(\tau)$ 优先选择包含任务个数最少的一条谐波链。

原则2是由于考虑到在一条谐波链中的任务不可避免的会遇到更另外一条谐波链中任务较小数值的 \gcd , 所以我们宁愿在任务选择的开始阶段经历这一过程, 此时系统的资源相对充足。

最终, 任务选择策略 $\text{MS}(\tau)$ 按照以上两条原则进行任务选择。首先, 它将系统中的所有任务划分为不同的谐波链, 然后每一步它选择包含任务数量最少的谐波

链进行操作，在每条谐波链内， $MS(\tau)$ 按照任务周期从小到大的顺序选择任务进行可调度性判定。现在，我们再来考虑例子4.3中的任务集，根据以上的讨论，我们得到最终可调度的任务选择顺序 $\{\tau_2, \tau_1, \tau_3\}$ 。

4.2.3 复杂度分析

本章所提出的可调度性判定算法具有伪多项式时间复杂度 $O(n^2)$ ，其中 n 是严格周期任务集中的任务个数。注意，在检查可行的开始时刻过程中所涉及的最大公约数 gcd 的计算也应当考虑在内。在文章[144]中，对于 m 位数计算 gcd 的复杂度可以提升到 $O(m(\log m)^2 \log(\log m))$ ，并且，作者基于快速整数运算^[145]提出一种准线性算法用来计算 n 个 m 位数所有的 gcd 对。总之，我们的方法相比指数级时间复杂度的最优算法能够在合理时间内得到计算结果，这也同样可以从下一小节的实验对比中看出。

4.3 实验

在这一小节中，我们通过大量随机实验来验证本章所提出的可调度性判定算法的性能。我们比较了包括基于[48]的最优精确判定算法在内的多个现有针对严格周期任务集的分析方法。

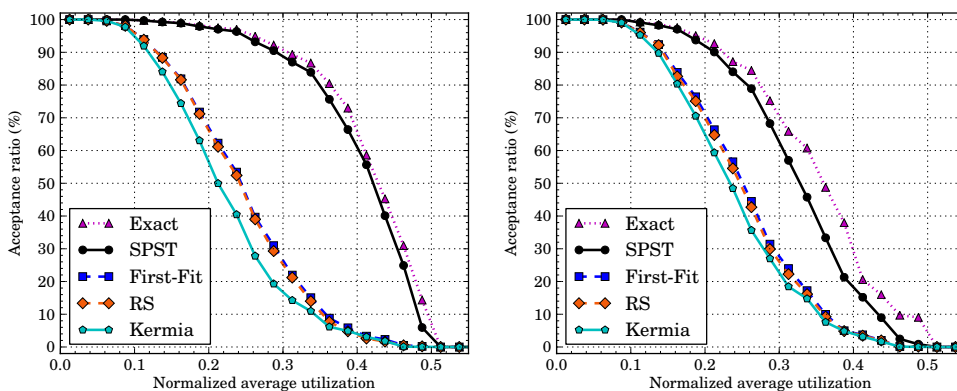
实验中的严格周期任务集都是通过以下方式随机生成的。由于想要我们的方法呈现普适性，我们考虑一般性即包含谐波任务也包含非谐波任务的严格周期任务集，任务集的生成由以下参数控制：任务是一个非谐波任务的概率 P^n ；最差执行时间最大值 C^{max} ；谐波周期集合 T^h 和非谐波周期集合 T^{nh} 。之后，每个任务 τ_i 按照以下方式随机生成： C_i 从 $\{1, 2, \dots, C^{max}\}$ 中随机取出；如果 τ_i 是一个谐波任务（ $1 - P^n$ 的概率），则其周期 T_i 随机从 $T^h = \{1500, 3000, 6000, 12000, 24000\}$ 中取值；如果 τ_i 是一个非谐波任务（ P^n 的概率），则其周期 T_i 随机从 $T^{nh} = \{2^x \cdot 3^y \cdot 50, x \in [0, 4], y \in [0, 3]\}$ 取值。一个严格周期任务集从一个空集 $\tau = \emptyset$ 开始向其中添加随机任务。具体的，每个任务集的生成都是根据一个目标利用率 U^* ，我们允许任务集的利用率范围在 U^* 左右小范围内波动，即 $[U^* - 0.005, U^* + 0.005]$ ，因为当所有参数都为整数时，我们很难生成一个利用率完全等于 U^* 的任务集。随后，我们向任务集中逐个添加随机任务，直到系统利用率在目标范围内。如果利用率超过了目标范围，我们则删除该任务集并重新生成一个新的随机任务集。

如章节4.1.2中所介绍，定理4.1中的精确可调度性判定公式由于较高的时间开销而不能应用于大规模的严格周期任务系统，我们在实验中包含5个任务的任务

集，任务生成的相关参数为 $P^n = 0.5, C^{max} = 500$ ，而结果是相应的计算时间最少的需要15分钟，最多的则达到1个小时。

为了能够在大量随机任务集中能够跟精确的可调度性判定方法进行性能比较，我们通过参数控制生成相对来说规模较小的任务集。具体的，我们令 $T^h = \{15, 30, 60, 120, 240\}$ ， $T^{nh} = \{2^x \cdot 3^y \cdot 5, x \in [0, 2], y \in [0, 2]\}$ 以及 $C^{max} = 10$ 。最后，我们总共比较以下5种现有的针对严格周期任务集的可调度性判定方法。

- **STSP**。本章所提出的可调度性判定算法STSP(τ)。
- **Exact**。基于[48][47]中定理4.1种条件(4.2)的可调度性判定方法。
- **First-Fit**。基于定理4.1的一种首次适应启发式算法，该算法以首次适应的方式为任务分配开始执行时间。具体的，在每次循环中，算法只要找到一个任务可以满足条件(4.1)的开始执行时间分配，则将其放入到可调度任务子集中。当且仅当不能再找到任意一个任务满足条件，算法返回失败。
- **RS**。一种简化的STSP(τ)算法，其主要区别就是在算法执行过程中适用随机的任务选择策略，而不是利用本章所提出的MS(τ)算法。
- **Kermia**。文章[142]中所提出的可调度性判定方法。



(a) $P^n = 0.1$

(b) $P^n = 0.5$

图 4.5 不同 P^n 设置下生成随机严格周期任务集所对应的实验结果
 Fig. 4.5 Experiment results with different P^n for strictly periodic task sets.

图4.5所示为系统接受率随不同参数设置 ($P^n = 0.1, P^n = 0.5$) 生成的系统目标利用率变化的函数。通常情况下，目标系统的平均资源利用率越高，则系统可调度的可能性就越低，这也同样能从图中看出，不同方法所对应的每条曲线都随着系统资源利用率的增加而降低。图中曲线中的每个点都是生成2000个随机严格周期任务集所产生的实验结果。如图所示，本章所提出的可调度性分析方法相比较最优的算法能够一直保持一个可接受的系统接受率。具体的，在 $P^n = 0.1$ 的情况

下，我们方法的系统接收率非常接近于最优算法所产生的结果。另一方面，相较于其他的可调度性判定方法（First-Fit, RS和Kermia），我们方法的性能优势非常明显。

此外，我们还通过改变任务集中任务个数来比较了本章所提方法和最优算法的运行时间开销。如图4.6所示，在我们把任务个数从20增加到80的过程中，最优算法的执行时间呈指数级的增长，然而我们的方法依然能够保持在一个合理的时间开销范围内。

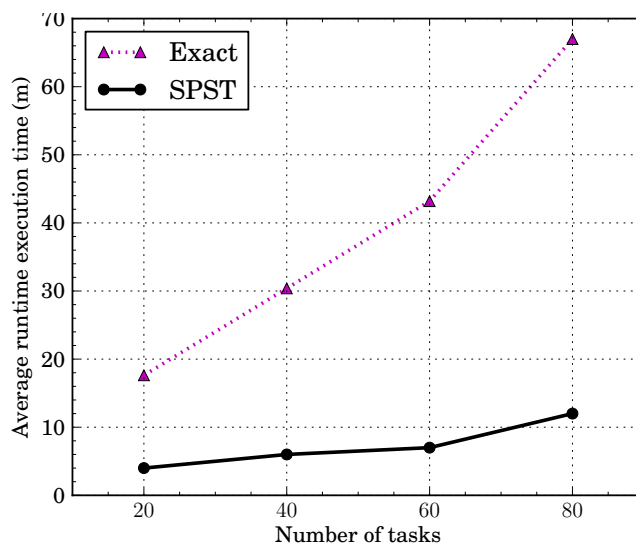


图 4.6 运行时间开销对比

Fig. 4.6 Run-time overhead comparison.

以上所有实验结果表明本章所提出的针对严格周期任务系统的可调度性判定方法在可调度性和运行效率之间取得了非常好的平衡。

4.4 小结

在本章中，我们针对严格周期任务系统提出了一种新的可调度性分析方法，该方法提供了一个充分非必要的可调度性判定条件来检查一个任务集的可调度性，并且当任务集可调度时，还为每个任务分配可行的开始执行时间。为了提高分析的精度，我们还提出了一种合理的任务选择策略。通过大量的随机任务集实验，验证了我们新的方法确实大大提高了分析的精度。

第5章 处理实时无线传感网络中干涉的分布式动态包调度策略

实时无线传感网络 (Real-time wireless networks, RTWNs) 在很多CPS应用, 例如军事、公民基础设施以及工业自动化中扮演着非常重要的角色^[17-19]。一个实时无线传感网络通常由若干分布在空间中的传感器、执行器、中间节点和一个网关组成。所有这些组件通过无线的方式连接起来以完成系统中所有实时任务的传输。实时无线传感网络所提供的服务质量通常是由系统满足任务实时性能 (从传感器经网关最终到达执行器的端到端截止期) 的好坏程度所衡量, 因此, 网络中链路层的包调度问题在提供指定服务质量的过程中扮演着重要角色, 而且随着网络规模的不断增大, 包调度的问题也变得越来越有挑战。此外, 大部分实时无线传感网络必须要应对网络中不时出现的干涉, 这一现象使得问题变得更加棘手。

在实时调度领域, 实时无线传感网络中的静态包调度问题已经得到了很好的研究^[146-148]。静态的调度方法支持决定性的实时传输, 但是并不能应对系统运行过程中发生的动态事件。动态事件可以由两种情况所引起, 一种是网络资源提供的改变 (例如节点或链路失效, 节点间干涉所引起), 另一种是由监测和控制的环境中发生的外界干涉 (例如检测到入侵者, 突然的压力变化等) 所引起的网络资源需求的改变。相应的, 很多研究者们提出了很多集中式的包调度方法, 但是这些方法大部分都是用来处理第一种网络资源提供的改变 (例如^[99, 100, 149]), 而本章的研究重点, 即研究应对网络外部干涉的工作则相对较少。文章^[101]中的方法是在系统运行前预先设计一定数量的数据链路层调度表, 当运行中出现网络外部干涉相关的动态事件后再从预先存储的调度表中选择其中最符合的。但是, 这种方法或者不能处理任意情况的干涉, 或者必须要做出非常悲观的近似, 对干涉的处理也将大打折扣。此外, 文章也并没有讨论当出现干涉时, 如何从现有的调度表中选取最合适的一个。文章^[102]和^[103]都支持接入控制, 即允许在系统运行中添加或删除任务, 以此来响应出现的干涉事件, 但是它们都不考虑当系统并不能满足所有任务的截止期约束这种情况。文章^[104]提出为偶尔出现的干涉事件预先分配预留时间片的方法, 并且允许当系统中没有干涉事件时, 原有的任务可以使用这些预留时间片。但是文章并不考虑当出现干涉事件时如何满足普通任务的截止期约束。

很多研究工作致力于对实时系统中干涉事件的建模和处理。文章[150]提出了采样周期调整的方法来提升事件驱动控制系统的性能，允许任务改变周期和截止期的速率适应模型和节奏任务模型也在文章[151]和[152]中被相继提出，[152]和[151]针对这些模型也提出相应的基于速率单调（RM）调度和最早截止期优先（EDF）算法的可调度性分析方法。虽然这些现有的任务模型可以被应用于实时无线传感网络，但是它们相应的调度方法和可调度性分析却并不能够直接应用，因为它们并没有考虑网络中端到端的包传输需求。

文章[105]中提出的OLS算法是当前最优的能够处理实时无线传感网络中干涉事件的一种动态调度方法。OLS是一种基于节奏任务模型的集中式调度框架，并且已经被证明对于小规模的网络具有较好的性能和效率。但是，对于大规模的实时无线传感网络（例如网络中的节点数量超过30个），OLS会带来非常高昂的计算开销，并且可能并不能够生成一个可行的调度表来处理干涉事件。此外，由于实时无线传感网络中广播包大小的限制，OLS可能会不必要的丢掉很多额外的网络包，这对网络的服务质量也有不好的影响。所有以上这些都会导致系统遭受干涉事件时的性能降低。

在本章中，我们提出一种新颖的分布式动态包调度框架（distributed dynamic packet scheduling framework, D²-PaS），来处理实时无线传感网络中出现的干涉事件。作为一种分布式的方法，D²-PaS允许网络中的每个设备节点在本地自行计算生成自己的调度表，这大大的减少了当干涉发生时网关节点通过广播包向每个设备节点所发送的调度表相关信息。作为一种动态方法，为了响应出现地干涉，D²-PaS在网关节点在线动态地计算哪些普通的网络包需要被丢弃，然后将最少的必要信息广播发送给网络中其它的设备节点。为了保证这样一种分布式动态方法的有效性和高效性，我们设计了一种轻量级（在内存使用和计算能力两方面）的调度器，部署在网络中的每个设备节点上。此外，我们在网关节点上设计了一种低复杂度的算法，在线地计算生成一段时间的临时调度表来处理发生的干涉事件。

我们在一个真实的实时无线传感网络测试平台上实现了D²-PaS框架，以此来验证其实用性。我们也做了大量的模拟实验以及在测试平台上的实验来评估D²-PaS的性能。当面对网络中只有一个干涉的情况时，相比于当前最优的[105]中的OLS方法，D²-PaS平均能够把丢包率从69%降低到4%，在最好情况下能从99%降低到9%。在满足关键包的截止期方面，D²-PaS能够获得100%的成功率，而OLS的平均成功率为90%而在最坏情况下仅为62%。在计算开销方面，D²-

PaS比OLS快2000倍到10000倍。当需要同时处理多个并发的干涉时，D²-PaS相较于OLS在各个方面的提升则更加明显。

5.1 预备知识

在本节中，我们首先介绍系统模型，然后给出本章所设计的D²-PaS框架总览。

5.1.1 系统模型

本章采用一种典型的实时无线传感网络系统架构，网络由许多传感器节点和执行器节点通过无限的方式直接经由一个网关节点或经若干中间节点相连。我们假设传感器节点和执行器节点都具有路由功能，且都配备一个全向的天线，在单信道半双工模式下运行。整个网络可以由一个全连接图 $G = (V, E)$ 表示，节点集合为 $V = \{\{V_0, V_1, \dots\}, V_g\}$ ，其中 V_g 代表网关节点，所有其它节点我们统称为设备节点。 $(V_i, V_j) \in E$ 表示从节点 V_i 到节点 V_j 的一条可靠链路（本章我们假设网络中所有的链路都是可靠的，处理不可靠链路可以利用很多现有研究工作）。网关节点 V_g 直接或间接地连接所有节点来控制逻辑，它还包含一个网络管理器来负责网络配置和资源分配。

为了简化表述，我们假设系统中运行着一个固定数量地任务集 $\mathcal{T} = \{\tau_0, \tau_1, \dots, \tau_n, \tau_{n+1}\}$ 。 τ_i ($0 \leq i \leq n$) 是一个单播任务，其唯一的一条路由路径包含 H_i 跳。它周期性地从传感器节点采集数据后生成一个数据包，并按照其路由路径转发给网关节点 V_g ，之后将网关节点生成地控制指令传输给相应的执行器节点。 τ_{n+1} 是一个广播任务，它在网关节点上运行，负责将调度相关信息通过特定的广播路由发送给整个网络^[146]。如图5.1所示为一个实时无线传感网络例子，其中3个单播任务和1个广播任务运行于网络中的8个节点上 (V_0, \dots, V_6 以及 V_g)，其中 V_0 、 V_1 和 V_2 是传感器节点， V_4 、 V_5 和 V_6 是执行器节点， V_3 是一个中间节点， V_g 是网关节点。相应的任务参数如表5.1所示。

为了模型化的描述当发生干涉事件是网络资源需求量的突然增长，本章采用节奏任务模型^[152]，因为它已经在处理实际事件驱动控制系统中的干涉方面得到研究者的认可^[105]（本章所提出的D²-PaS方法不仅限于节奏任务模型，而是能够处理任意给定网络资源需求量变化的任务集）。具体的，每个单播任务 τ_i 具有两种状态：正常状态和节奏状态。在正常状态下， τ_i 按照正常周期 P_i 和正常截止期 $D_i \leq P_i$ 来运行。当一个跟任务 $D_i \leq P_i$ 相关的干涉出现

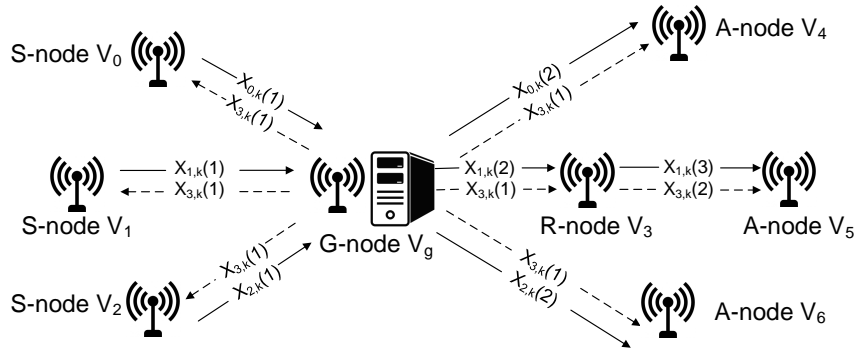


图 5.1 一个实时无线传感网络实例

Fig. 5.1 An example RTWN with 3 unicast tasks and 1 broadcast task running on 8 nodes.

表 5.1 实时无线传感网络实例中的任务参数

Table 5.1 Task parameters for the example RTWN.

Task	Routing Path	P_i	D_i
τ_0	$V_0 \rightarrow V_g \rightarrow V_4$	10	9
τ_1	$V_2 \rightarrow V_g \rightarrow V_6$	10	8
τ_2	$V_1 \rightarrow V_g \rightarrow V_3 \rightarrow V_5$	10	7
τ_3	$V_g \rightarrow *$	10	10

时， τ_i 进入到节奏状态，此时它的周期和截止期先是为了应对干涉事件而骤减，然后再逐渐的递增直到恢复至原始的正常周期和正常截止期。我们用向量 $\vec{P}_i = [P_{i,x}, x = 1, \dots, R_i]^T$ 和 $\vec{D}_i = [D_{i,x}, x = 1, \dots, R_i]^T$ 来表示 τ_i 在节奏状态下周期和截止期变化的过程。一旦 τ_i 进入到节奏状态，它的周期和截止期就按照向量中的值逐渐变化，当它的周期和截止期恢复到正常值时， τ_i 也相应的回到正常状态且继续按照正常周期和截止期运行。图5.2所示为 τ_i 在节奏状态下运行的图示。

当检测到干涉出现并报告给网关节点 V_g ，相应的任务进入到它的节奏状态，而系统也相应的转换到节奏模式下运行以处理干涉事件。只要没有其它并发的干涉出现，当处理完当前干涉后（通常是当相应的节奏任务回到它的节奏状态后的

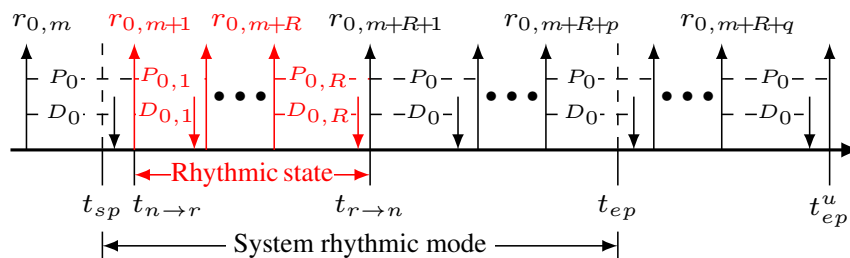


图 5.2 节奏模型示例

Fig. 5.2 Timing parameters for τ_0 and for the system in the rhythmic mode.

某一时刻), 系统就回到正常模式下继续运行(见5.2)。为了简化符号, 在系统运行于节奏模式下时, 我们称相应的进入到节奏状态的任务为节奏任务, 所有其它保持在正常状态运行的任务称为周期任务(由于广播任务只有一种运行状态, 并且它也遵从固定的周期, 如果没有特别说明的时候, 本章所说的周期任务也包含广播任务)。因为干涉事件可能会对系统运行产生严重后果, 当系统运行于节奏模式时, 节奏任务的截止期为硬截止期, 即必须满足, 而其它周期任务可以允许截止期偶尔的错失。接下来, 我们首先假设在系统运行过程中的任意时刻, 至多只有一个任务处在它的节奏状态(表示为 τ_0), 我们会在随后的章节5.4将系统模型扩展到包含并发干涉, 并介绍D²-PaS如何处理。

每个任务 τ_i 都会释放任意多个任务实例, τ_i 的第 k 个实例我们称为一个包 $\chi_{i,k}$, 它的释放时间为 $r_{i,k}$, 绝对截止期为 $d_{i,k}$, 以及完成时间(即包到达执行器的时刻)为 $f_{i,k}$ 。不是一般性的, 我们假设节奏任务 τ_0 在 $r_{0,m+1}$ 时刻(表示为 $t_{n \rightarrow r}$)进入它的节奏状态, 在 $r_{0,m+R_0+1}$ 时刻(表示为 $t_{r \rightarrow n}$)回到它的正常状态。因此, τ_0 在 $[t_{n \rightarrow r}, t_{r \rightarrow n})$ 区间内在它的节奏状态下执行, $t_{n \rightarrow r}$ 和 $t_{r \rightarrow n}$ 满足 $t_{r \rightarrow n} = t_{n \rightarrow r} + \sum_{x=1}^{R_0} P_{0,x}$ 。任意节奏任务 τ_0 在其节奏状态下释放的包我们都成为节奏包, 相应的表示为 $\chi_{0,k}$, 而任意其它周期任务 τ_i ($1 \leq i \leq n+1$)释放的包称为周期包。包 $\chi_{i,k}$ 在其第 h 跳上的发送接收称为一个传输, 表示为 $\chi_{i,k}(h)$ ($1 \leq h \leq H_i$)。根据实际工业信息物理系统, 我们的模型采用时分复用(Time Division Multiple Access, TDMA)的方式进行数据链路层传输, 每个节点按照给定的调度表发送和接收包, 每个传输 $\chi_{i,k}(h)$ 必须在一个单位时间片内完成。表5.2总结了一些本章常用的符号表示。

基于以上描述的系统模型, 总的来说, 我们想要解决以下问题。给定一个实时无线传感网络, 设计一个包调度框架满足 (i) 当系统中没有干涉出现时, 所有包都能够在它们的截止期前传输到达相应的执行器节点, (ii) 当系统出现干涉时, 所有节奏包都能够在它们的截止期前传输到达相应的执行器节点并且被丢掉的周期包的个数最少, (iii) 当干涉被处理之后, 系统能够安全的回到正常模式运行, 且所有包都能够满足截止期。

5.1.2 D²-PaS框架总览

在本节中, 我们首先用一个例子来说明静态调度和集中式调度在处理节奏任务时的缺陷, 之后我们介绍本章所提出的分布式动态包调度策略D²-PaS。

考虑图5.1中的实时无线传感网络, τ_0 是节奏任务, τ_1 和 τ_2 是周期任务, τ_3 是广播任务, 它们的路由路径、周期、截止期, 包括节奏任务周期和截止期信息都在

表 5.2 重要符号总结

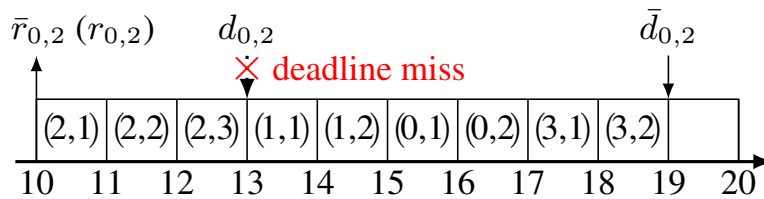
Table 5.2 Summary of important notations and definitions.

符号	定义
$V_j, j = 0, 1, \dots$	设备节点
V_g	网关节点
$\tau_i (0 \leq i \leq n)$ 和 τ_{n+1}	单播任务和广播任务
τ_0 and $\tau_i (1 \leq i \leq n+1)$	节奏任务和周期任务(单个干涉事件模型)
H_i	τ_i 路由路径中包含的跳数
$P_i (D_i)$	τ_i 的正常周期(截止期)
$\vec{P}_i (\vec{D}_i)$	τ_i 的节奏周期(截止期)向量
$\chi_{i,k}$	任务 τ_i 释放的第 k 个包
$\chi_{i,k}(h)$	包 $\chi_{i,k}$ 的第 h 跳传输
$r_{i,k}, d_{i,k}, f_{i,k}$	$\chi_{i,k}$ 的释放时刻、截止期和完成时刻
$t_{n \rightarrow r}, t_{r \rightarrow n}$	节奏任务 τ_0 离开其正常状态和节奏状态的时刻
$t_{sp}, t_{ep}, t_{ep}^c, t_{ep}^u$	开始时刻、结束时刻、备选结束时刻以及结束时刻上限
t_{np}	无带入包时刻
$\Gamma(t_{ep})$	备选结束时刻集合
$\Psi(t)$	$[t_{sp}, t)$ 区间内的活跃包集合
R_i	$\vec{P}_i (\vec{D}_i)$ 中的元素数量
Δ^d	最大允许丢包的数量
$\rho[t_{sp}, t)$	$[t_{sp}, t)$ 区间内丢掉的周期包集合
SS_j	节点 V_j 的调度表段

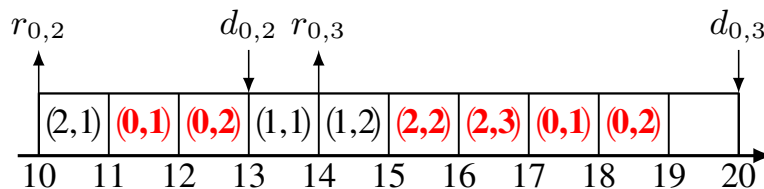
表5.1中所示。

所有周期任务和节奏任务的第一个包 $\chi_{0,1}$ 、 $\chi_{1,1}$ 和 $\chi_{2,1}$ 都在0时刻一起释放。当系统开始运行时，所有节点都按照长度为10的静态调度表周期性的发送和接收包(如图5.3-(a))。图中，每个时间片内的 (i, h) 表示当前时间片分配给任务 τ_i 的第 h 跳传输。假设在时刻10， τ_0 进入到它的节奏状态(即 $t_{n \rightarrow r} = 10$)，根据表5.1中的 \vec{P}_0 和 \vec{D}_0 ， τ_0 在 $t_{r \rightarrow n} = t_{n \rightarrow r} + P_{0,1} + P_{0,2} = 20$ 时刻回到它的正常状态。如果系统在 $t_{n \rightarrow r}$ 时刻之后依然使用静态调度表，在10时刻释放的节奏包 $\chi_{0,2}$ 将会在13时刻错失它的截止期。因此，静态调度并不能让系统正确的响应 τ_0 周期和截止期的变化。

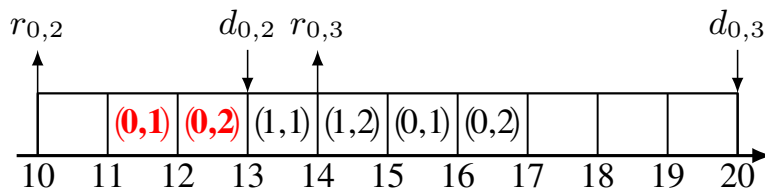
在集中式动态调度方法中，当干涉出现时，网关节点会为网络在节奏模式下生成一个临时的动态调度表，然后将动态表和静态表的区别信息广播给每个设备节点。当系统回到正常模式后再重新使用原始的静态调度表。图5.3-(b)所示为一个可能的动态调度表，该调度表可以调度所有的节奏包和周期包(仅丢掉一个广播包)，但是需要在静态调度表的基础上更新6个时间片(11, 12以及15-18)，这些更新的时间片信息需要打包在一个广播包中，再传输给网络中的每个设备节



(a) 静态调度表例子



(b) 需要更新6个时间片的动态调度表



(c) OLS生成的动态调度表

图 5.3 静态调度和集中式调度示例

Fig. 5.3 Motivational example.

点。由于实时无线传感网络中广播包的大小通常都是有限制的，因此相应的最多可以更新的时间片数量也将受限。假设在该例子中，最多只能够更新4个时间片，那么图5.3-(b)中的动态调度表就不能被放入一个广播包中。[105]中提出的在线调度框架OLS就考虑了多可以更新的时间片数量，尝试丢掉最少数量的周期包来满足这一限制条件。图5.3-(c)所示就是OLS所生成的动态调度表，它只需要更新2个时间片，但是会丢掉1个周期包 $\chi_{2,2}$ 。

虽然在处理干涉事件时，像OLS这样的集中式动态调度方法比静态调度策略更具优势，但它主要有两个缺陷。首先就是最多可以更新的时间片数量的限制，如果生成的调度表需要更新时间片的数量超过了限制条件，那么就必须额外丢弃更多的周期包。其次，集中式动态调度在生成动态调度表时会引入较高的时间延迟，并且当网络规模增大时，该问题更加明显。如果网关节点不能在下一个广播包到来之前生成动态调度表，当前的干涉事件则不能够被及时正确的处理。本章就提出一种新的方法来解决以上问题。

通过观察我们知道，集中式动态调度框架之所以受最多可以更新的时间片数量限制，主要原因是在处理发生的干涉时，网关节点包揽了所有的工作，而其它

设备节点只需要接收并更新从网关节点收到的动态调度表。这样的选择实际上相当于假定所有设备节点并不具有任何的计算能力，然而对于当今的实时无线传感网络来说这并不是事实。例如，我们运行OpenWSN协议栈以及实现D²-PaS框架的CC2538片上系统，是有可能执行更多的本地计算能力的。虽然它们的计算能力无法跟网关节点相比拟，但是像计算本地调度表这样的基本操作还是可以完成的。因此，本章的思路就是利用设备节点的本地计算能力，设计一种分布式的动态包调度框架，以期获得比集中式方法更好的性能，如减少丢包数量和更高的系统接受率。

算法 3 Main function of D²-PaS

```

1: while true do
2:   Every node generates and follows its local schedule.
3:   if a disturbance is detected and reported to the gateway then
4:      $V_g$  checks the schedulability of the system after  $t_{n \rightarrow r}$  and calculates the time duration of the system rhythmic mode.
5:     if the system is overloaded then
6:        $V_g$  determines the periodic packets to be dropped.
7:     end if
8:      $V_g$  propagates the rhythmic task information and dropped packet set to all nodes.
9:   end if
10: end while

```

算法3描述了分布式动态包调度框架D²-PaS的概况。关于D²-PaS的详细介绍将在之后几节中给出，其大致执行过程如下。在系统初始化之后，当每个节点从网关节点收到一个广播包后，它们就在本地根据指定的调度算法生成一段时间的调度表然后按照其执行。这里我们采用的是EDF算法。为了生成这样一段调度表，每个节点根据任务和路由信息决定在每个时间片内是否发送或接收某个包，或者保持空闲。这一过程可以通过简单的执行EDF模拟程序得到。由于网络中每个节点维护的是相同的任务和路由信息，因此在不同节点生成的调度表也是一致的。

当某个传感器节点检测到一个干涉发生时，它根据当前调度表通过任务 τ_0 发送一个节奏事件请求给网关节点。当在 t' 时刻（见图5.4）收到该请求后， V_g 首先检查如果 τ_0 进入到它的节奏状态后系统的可调度性。如果此时系统是过载的， V_g 则决定需要丢掉哪些周期包来保证所有节奏包都不错失截止期。然后 V_g 打包这些丢包信息以及将要进入节奏状态的任务信息（即 \vec{R}_i 和 \vec{D}_i ）到一个广播包中，然后在 t'' 时刻将该广播包发送给网络中的节点。如果系统不会过载的话，则网关节点

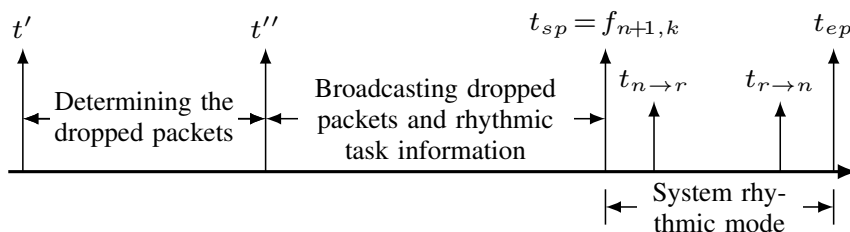


图 5.4 网关节点收到干涉报告后的网络执行示例。 t' 表示 V_g 收到节奏事件请求的时刻， t'' 表示 V_g 发送第一跳广播包的时刻。

Fig. 5.4 Network operations after disturbance is reported to the gateway V_g . t' denotes the time slot at which V_g receives the rhythmic event request. t'' denotes the time slot at which V_g sends the first hop of the broadcast packet.

只需要将节奏任务的相关信息广播给网络。因此，不同于 OLS 将整个动态调度表打包广播， D^2 -PaS 仅需要将那些需要被丢掉的包的索引编号打包广播。当在 t_{sp} 时刻或之前某一时刻收到该广播包后，每个节点根据收到的更新信息在本地生成相应的调度表，之后系统就可以转换到节奏模式下运行了。

为了确保 D^2 -PaS 能够正确的运行，我们需要解决以下几个问题。首先，在理想情况下，在系统运行于正常模式下时，每个设备节点可以在本地生成超周期长度的一整段调度表然后保存。但是，由于设备节点有限的计算能力和存储空间，这种方法在实际中是不可行的。其次，因为生成一段调度表是需要时间的，所以计算调度表的过程不应当发生在节点需要接受或者发送包的时间片内。第三，为了能够及时快速的响应发生的干涉，需要在网关节点上设计一个高效的方法来决定哪些周期包需要被丢掉。接下来，我们详细的介绍 D^2 -PaS 如何解决以上问题。

5.2 本地调度表的生成

本节主要讨论如何在设备节点本地生成调度表。我们的主要设计思路是，在系统运行过程中每个设备节点逐段地去计算生成和保存调度表。具体的，每个节点在生成本地调度表时，遵从以下两个原则：(i) 每次只生成和保存整个 EDF 调度表的一段，以此避免一次性的计算整段的调度表。(ii) 每个节点都要在其自身的空闲时间片内运行本地调度表生成算法。那么为了满足这两条原则，我们在设计时需要回答以下三个问题：(1) 如何来确定一段调度表？(2) 这些调度表片段的长度上限是多少？(3) 每次更新调度表片段时需要维护哪些信息？下面我们首先引入一些必要的定义，然后在给出对这些问题的解答。

定义 5.1 (本地忙碌时间片): 如果节点 V_j 在一个时间片内发送或者接收任意一个单播任务的包传输，那么该时间片是 V_j 的一个本地忙碌时间片，

定义 5.2 (本地空闲时间片): 如果一个时间片不是节点 V_j 的本地忙碌时间片, 那么它是 V_j 的一个本地空闲时间片。

定义 5.3 (调度表片段): 一个调度表片段, 表示为 SS_j , 是节点 V_j 每次计算生成的一段本地调度表。 SS_j 或者起始于一个 V_j 接收到一个广播包的时间片, 或者起始于 V_j 结束一段连续的本地忙碌时间片后的首个本地空闲时间片。

我们设计的 D^2 -PaS框架根据调度表片段 SS_j 的定义, 令每个节点逐段的在本地生成它的调度表。本地调度表的计算是根据EDF策略来决定在 SS_j 中的每个时间片应当接收或发送哪个包, 还是保持空闲。一个调度表片段的生成必须在一个本地空闲时间片内完成以保证 D^2 -PaS能够正确运行。因为根据IEEE 802.15.4e 协议, 在一个接收或发送广播包的时间片内是不需要确认接收过程的, 数据链路层操作不会超过8ms, 因此时间片内剩余的时间足够设备节点完成本地调度表的计算。利用广播时间片来计算本地调度表的另一个好处是, 它保证了调度表的生成一定是基于系统当前最新的运行状态(是否有干涉发生)。这样, 先前所说的设计原则ii也自然地满足了, 因为节点都是在本地空闲时间片内计算生成调度表(根据定义5.2, V_j 的一个广播时间片也是它的一个本地空闲时间片)。考虑5.1.1节中的实时无线传感网络例子, 图5.5所示为节点 V_3 的前三个调度表片段。

节点 V_j 生成调度表片段的时间取决于 SS_j 的长度。如图5.5所示, 每个 SS_j 由一个或多个连续的本地空闲时间片及之后的一段本地忙碌时间片组成。因为只有本地忙碌时间片的长度决定了计算生成 SS_j 的时间, 我们只需要找到一段连续本地忙碌时间片长度的可能的最大值。由于网关节点的计算能力突出, 可以负担复杂的计算, 我们只需考虑在设备节点上 SS_j 的长度。以下的引理5.1和定理5.2说明了任意一段 SS_j 长度的上限作为经过节点 V_j 任务个数的函数。

引理 5.1: 如果系统是可调度的, 即每个包的所有传输都能够在截止期前到达

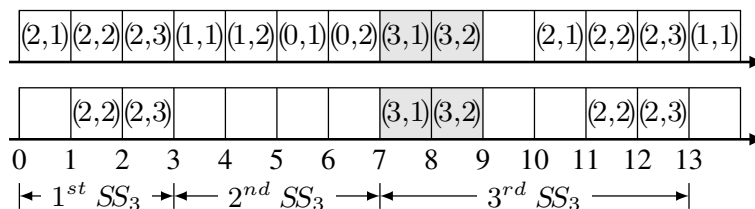


图 5.5 节点 V_3 的调度表片段。上(下)图为系统(V_3)的全局(本地)调度表。

Fig. 5.5 Schedule segments of node V_3 . The top (bottom) is the global (local) schedule of the system (V_3).

Blank slots are global and local idle slots, respectively. Gray slots are assigned for transmissions of broadcast packets.

执行器节点, 对于任意一个设备节点 V_j 和路径经过 V_j 的任务 τ_i , 在 V_j 的调度表中任意三个连续的 τ_i 的传输中间一定存在至少一个本地空闲时间片。

证明: 路径经过节点 V_j 的任务 τ_i 的任意三个连续传输只能是 (i) 来自 τ_i 的三个不同的包, 或者 (ii) 来自 τ_i 的两个不同的包。在情况 (i) 中, 假设 V_j 完成了传输 $\chi_{i,k}$, 如果在 V_j 执行下一个包的传输 $\chi_{i,k+1}$ 之前, V_j 的调度表中没有本地空闲时间片, 则任务 τ_i 的路径只会包含一跳, 即从传感器节点 (V_j) 到后继节点或者从前驱节点到执行器节点 (V_j)。否则的话, V_j 至少会有至少一个本地空闲时间片是其他的某个节点执行 $\chi_{i,k}$ 的另一个传输。这说明或者 τ_i 的路由不经过网关节点, 或者 τ_i 的路径不包含传感器节点或执行器节点。这跟系统模型相矛盾。

对于情况 (ii), 假设三个中的两个传输分别是包 $\chi_{i,k}$ 在节点 V_j 上的一个接收传输和发送传输, 第三个则是 $\chi_{i,k+1}$ (或 $\chi_{i,k-1}$) 在 V_j 上的一个接收传输(或发送传输)。如果 V_j 的调度表中在这三个连续的传输中不包含本地空闲时间片, 这代表任务 τ_i 在它的路由路径上只有两跳。否则的话, V_j 的调度表中一定存在一个本地空闲时间片来执行包 $\chi_{i,k}$ 的第三跳不经过 V_j 传输。在这种情况下, V_j 一定是网关节点, 因为根据系统模型, 每个任务都必须经过网关节点。然而, 这跟我们的假设 V_j 是一个设备节点相矛盾。 \square

定理 5.2: 如果系统是可调度的, 节点 V_j 的调度表片段 SS_j 的最大长度为 $2 \times n_j$, 其中 n_j 是经过路径经过节点 V_j 的任务数量。

证明: 我们首先考虑 SS_j 的长度等于 $2 \times n_j$ 的情况。我们用 $\tau_1, \dots, \tau_{n_j}$ 表示路径经过节点 V_j 的所有任务, 假设 V_j 是所有这些任务的路径上的第二个节点, 并且任务 τ_1 到 τ_{n_j} 的截止期递减, 即 $d_1 > \dots > d_{n_j}$ 。假设这些任务的包按照图5.6所示的模式释放, 根据EDF调度策略, 每个包在节点 V_j 上的传输需要两个时间片(接收和发送), 并且之后立即被一个更高优先级(截止期更短)的任务抢占, 这种情况下, SS_j 的长度等于 $2 \times n_j$ 。

接下来我们通过反证法来证明 SS_j 的长度不会超过 $2 \times n_j$ 。假设节点 V_j 会连续忙碌超过 $2 \times n_j$ 个时间片。路径经过 V_j 的任务数量为 n_j , 这意味着至少有一个任务在 SS_j 内需要在 V_j 上超过两个时间片的传输。那么根据引理5.1, 这超过两个的传输中必然包含至少一个本地空闲时间片, 这跟我们的假设相矛盾。 \square

现在我们讨论在每个节点需要存储哪些数据来完成本地调度表的生成。首先, 生成的调度表一定需要存储, 为了节省内存, 只有节点本身需要执行传输的时间片相关信息需要保存(例如图5.5中的时间片1,2,11和12)。其次, 为了生成本

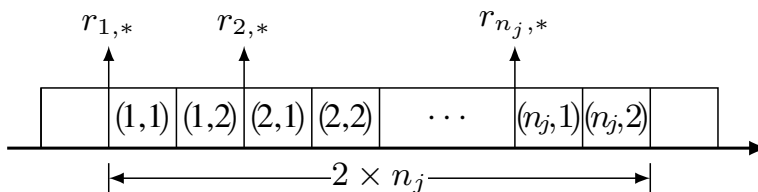


图 5.6 SS_j 的最长长度例子

Fig. 5.6 An example showing the worst-case length of SS_j . $r_{i,*}$ denotes the release time of an arbitrary packet of task τ_i .

地调度表，节点 V_j 必须知道整个任务集的参数信息。此外，虽然看起来 V_j 需要知道所有任务的路由信息，但实际上这是不必要的，因为 V_j 只需生成跟它自身相关的调度表，其它不经过 V_j 的任务的路由信息不需要保存。

为了能够逐段的计算本地调度表，我们需要在每个节点维护一个调度信息表来辅助节点进行调度表的计算。具体的，节点 V_j 在系统初始化阶段创建它自身的调度信息表，一个包含所有任务集信息的广播包会发送给实时无线传感网络，当 V_j 收到该广播包后，它会从中提取和保存必要的信息到自己的调度信息表中。表5.3是章节5.1.1中的实时无线传感网络例子中节点 V_3 在0时刻维护的调度信息表。三元组 $\langle I_i, s_i, r_i \rangle$ 任务 τ_i 跟节点 V_j 相关的路由信息， I_i 表示 V_j 是 τ_i 第 I_i 跳的接收节点（因此 V_j 也是 τ_i 第 $I_i + 1$ 跳的发送节点）。注意， $I_i = 0$ 表示 V_j 是 τ_i 的源节点，即传感器节点，而 $I_i = H_i$ 表示 V_j 是 τ_i 的目的节点，即执行器节点。 s_i 和 r_i 分别表示传输 χ_{i,I_i} 和 χ_{i,I_i+1} 的发送节点和接收节点。表中最后一列存储的向量 $\langle c_i, p_i \rangle$ 表示任务 τ_i 在当前正在运行的调度表结束时刻时的执行状态信息。在节点 V_j 每次计算本地调度表时，该调度信息表都需要动态的更新。具体的， c_i 表示举例目的节点的剩余跳数， p_i 表示在当前 SS_j 中 τ_i 的最后一个活跃包是第 p_i 个包。

基于以上介绍的调度表片段的概念和每个节点维护的调度信息表， D^2 -PaS可以在每个节点有效的生成和存储本地调度表。具体的，当 V_j 在系统初始化阶段收到广播包并创建调度信息表后，它就根据EDF策略计算自己的第一个调度表片

表 5.3 V_3 在 $t = 0$ 时刻存储的调度信息表

Table 5.3 The schedule table stored in node V_3 at $t = 0$.

Task	H_i	P_i	D_i	$\langle I_i, s_i, r_i \rangle$	$\langle c_i, p_i \rangle$
τ_0	2	10	9	N/A	$\langle 2, 1 \rangle$
τ_1	2	10	8	N/A	$\langle 2, 1 \rangle$
τ_2	3	10	7	$\langle 2, V_g, V_5 \rangle$	$\langle 3, 1 \rangle$
τ_3	2	10	10	N/A	$\langle 2, 1 \rangle$

段，之后就根据调度表片段的定义逐段的生成调度表并按照其发送和接收包。算法4详细介绍了 V_j 生成调度表片段 SS_j 的过程。

5.3 节奏模式下的调度

上一节我们讨论的本地调度表生成方法能够保证系统在没有干涉发生情况下正确运行，但是当系统进入节奏模式后，节奏任务会采用新的周期和截止期，而一些周期包可能必须要丢弃来保证节奏包的及时传输。此时，节点中保存的调度信息表就不足以应对这种情况。虽然像所有可能的节奏任务参数这类信息可以存储在本地，每个节点仍然需要知道哪个任务进入了它的节奏状态以及哪些周期包需要被丢弃。在本节中，我们详细介绍 D^2 -PaS如何处理系统运行过程中出现的干涉。我们首先在5.3.1节中给出形式化的问题定义，之后在5.3.2节中讨论如何决定系统节奏模式的结束时刻，在5.3.3节中我们介绍如何决定丢弃哪些周期包以应对节奏任务。

5.3.1 问题定义

当一个干涉事件通过任务 τ_0 报告给网关节点， D^2 -PaS需要生成一段更新的调度表让系统能够在节奏模式下使用，该动态调度表需要保证 (i) 所有节奏包都能满足截止期约束，(ii) 被丢弃的周期包的数量最小。该段更新调度表的运行区间为系统处在节奏模式下的时间段，即 $[t_{sp}, t_{ep})$ (见图5.4)。假设广播包 $\chi_{n+1,k}$ 到达网络中所有节点的时刻为 $f_{n+1,k}(H_{n+1})$ ，我们可以直接用 $f_{n+1,k}(H_{n+1})$ 作为 t_{sp} 。但是对于 t_{ep} 的选择就不那么简单了，它必须保证所有在其之后释放的包都能够满足它们的正常截止期。因此，在干涉出现后，网关节点主要需要解决的问题包括 (i) 决定系统节奏模式的结束时间 t_{ep} ，(ii) 在更新调度表内需要丢掉哪些周期包。令 $\rho[t_{sp}, t_{ep})$ 为在区间 $[t_{sp}, t_{ep})$ 丢掉的周期包的集合。那么，我们的目标就是

$$\min |\rho[t_{sp}, t_{ep})| \tag{5.1}$$

满足以下条件。

限制 5.1 每个节奏包 $\chi_{0,k}$ 必须满足其截止期 $d_{0,k}$ ，即 $f_{0,k} \leq d_{0,k}$ 。

限制 5.2 $f_{0,m+R_0} \leq t_{ep} \leq t_{ep}^u$ 。

t_{ep}^u 是一个用户定义的参数，表示处理当前节奏事件所允许的最大时间延迟。 $f_{0,m+R_0} \leq t_{ep}$ 保证了在系统回到正常模式运行之前节奏事件能够被完全处理。由于广播包所能携带的信息数量有限，我们设置一个 $[t_{sp}, t_{ep})$ 内的最大丢包数量，表示

算法 4 Schedule Segment Computation at Node V_j

Input: Schedule table stored in V_j , local idle slot t

Output: SS_j

```

1:  $SS_j \leftarrow \emptyset$ ;
2: while true do
3:   Compute the assignment of slot  $t$  according to a designated scheduling policy (EDF in this paper);
4:   if (slot  $t$  is to receive a broadcast packet) || ( $t$  is a local idle slot and  $t - 1$  is a local busy slot) then
5:     return  $SS_j$ ;
6:   else
7:     if slot  $t$  is assigned to a transmission involving  $V_j$  then
8:        $SS_j \leftarrow SS_j \cup \{t, i, h\}$ ; //  $i$  is the task ID and  $h$  is the hop index
9:     else
10:       $SS_j \leftarrow SS_j \cup \{t, -1, -1\}$ ;
11:    end if
12:  end if
13:   $t \leftarrow t + 1$ ;
14: end while

```

为 Δ^d ，因此我们有

$$\text{限制 5.3 } |\rho[t_{sp}, t_{ep}]| \leq \Delta^d.$$

最后，我们要解决的问题的形式化定义为：

P1: 给定任务集 \mathcal{T} ， $t_{n \rightarrow r}$ ， t_{sp} ， t_{ep}^u 和 Δ^d ，确定 t_{ep} 以及丢弃的周期包集合使得目标函数(5.1)达成，以及以上所有条件满足。接下来我们首先讨论如何确定节奏模式的结束时间，之后介绍D²-PaS中适应的丢包算法。

5.3.2 系统节奏模式的结束时刻选择

系统节奏模式结束时刻 t_{ep} （以下简称结束时刻）的选择不仅影响系统处于节奏模式下的时间长度，而且对丢弃的周期包数量也有影响。在OLS中也定义了区间 $[t_{sp}, t_{ep}]$ 。其中， t_{ep} （其称为转换时刻）必须跟一个任务的正常释放时刻对齐，因为OLS这类集中式的调度策略需要在系统转换回正常模式后继续使用原始的静态调度表。而在D²-PaS中，由于每个节点在本地独立的生成自己的调度表，我们不再受此限制，这也增加了我们丢弃更少数量周期包的可能。

一个可行的结束时刻应当保证 (i) 在其之前，当前的干涉已经被完全处理，(ii) 所有在其之后释放的包都能满足它们的正常截止期约束。我们首先确定一个可行结束时刻的充分条件。

定理 5.3: 如果以下条件满足, 则 t_{ep} 是一个可行的结束时刻。

条件 5.1 $t_{ep} \geq f_{0,m+R_0}$, 其中 $f_{0,m+R_0}$ 是在 τ_0 的节奏状态内释放的最后一个包。

条件 5.2 所有在 t_{ep} 之前释放的节奏包 $\chi_{0,k}$ 必须在它们节奏和正常状态下时, 满足其相应的节奏和正常截止期约束。

条件 5.3 所有在 t_{ep} 之前释放的周期包 $\chi_{i,k}$ 或者必须在 $\min(d_{i,k}, t_{ep})$ 之前完成, 或者被丢掉。

由于对该定理的证明比较简单, 此处我们省略。定理5.3中的三个条件共同保证了(1)当前的干涉能够被完全处理,(2)所有节奏包能够在系统运行于节奏模式时满足截止期约束,(3)系统没有从节奏模式内带入到正常模式的工作量。那么根据EDF调度策略的特性, 所有在 t_{ep} 之后释放的包都能够满足截止期。

我们的目的便是找到一个满足定理5.3中所有条件的 t_{ep} 作为系统节奏模式的结束时刻。因为所有节奏包必须在截止期之前完成, 为了满足条件5.3, 选择的 t_{ep} 必须满足 $t_{ep} \geq r_{0,m+R} + H_0$, 也就是早于 $\chi_{0,m+R}$ 的最早可能完成时刻。相似的, 我们很容易能够找到一个满足条件5.3的 t_{ep} , 只要能够保证 $t_{ep} \geq r_{0,m+R+p} + H_0$ 其中 $\chi_{0,m+R+p}$ 是 τ_0 在 t_{ep} 之前释放的最后一个包。对于5.3, 我们注意到在 $(r_{0,k}, r_{0,k} + H_0)$ ($m+R+1 \leq k \leq m+R+q$)内的任一时间片都不能作为 t_{ep} , 因为在 $r_{0,k}$ 释放的节奏包至少需要 H_0 单位时间来完成。因此, 满足 $t \in [r_{0,m+R} + H_0, t_{ep}^u] \setminus \cup_{m+R+1 \leq k \leq m+R+q} (r_{0,k}, r_{0,k} + H_0)$ 的时刻 t 能够满足条件5.3。在该集合内选择丢弃周期包最少的时刻是十分耗时的, 为解决这一问题, 我们首先定义遗留包的概念, 并且介绍一个关键的引理。

定义 5.4 (遗留包): 如果一个包在 t 之前释放并且截止期在 t 之后, 则它是在时刻 t 的一个遗留包。

引理 5.4: 假设 t^* 是一个满足定理5.3中所有条件的结束时刻。令 $r_{i,k}$ ($0 \leq i \leq n$)表示满足 $r_{i,k} \geq t^*$ 的最早的释放时刻, 则 $|\rho[t_{sp}, t^*]| \geq |\rho[t_{sp}, r_{i,k}]|$ 。

证明: 首先假设所有在 t^* 时刻的遗留包都在 t^* 之前完成, 那么 $[t^*, r_{i,k})$ 内全部为空闲时间片, 这表示 $|\rho[t_{sp}, t^*]| = |\rho[t_{sp}, r_{i,k}]|$ 。

其次, 假设当 t^* 为结束时刻时, 一些在 t^* 时刻的遗留包被丢弃了。这种情况下, 考虑将结束时刻从 t^* 调整到 $r_{i,k}$ 。因为 $r_{i,k}$ 是满足 $t^* \leq r_{i,k}$ 的最早的任务释放时刻, 则在 $[t^*, r_{i,k})$ 内没有包被释放。由于系统采用EDF调度策略并且 $t^* \leq r_{i,k}$, 在结束时刻调整之前完成的包在调整之后依然能够完成。此外, 对于哪些被丢掉的在 t^* 时刻的遗留包, 由于调整之后它们有更长的时间来传输, 因此可能其中一部分就能够执行完成。因此将 $r_{i,k}$ 作为结束时刻会导致丢掉相同数量甚至更少的遗留包,

即 $|\rho[t_{sp}, r_{i,k}]| \leq |\rho[t_{sp}, t^*]|$ 。 □

引理5.4意味着将两个连续的任务释放时刻之间的任一时刻作为结束时刻，相比直接将后一个释放时刻作为结束时刻，可能会丢掉更多的周期包。因此，我们只需要从满足条件5.3和条件5.3的释放时刻中选择结束时刻就可以了。我们将所有这些满足条件的可行的结束时刻称为结束时刻备选集 $\Gamma(t_{ep})$ ，其中

$$\Gamma(t_{ep}) = \left\{ \forall r_{i,j} \in [r_{0,m+R} + H_0, t_{ep}^u] \setminus \bigcup (r_{0,k}, r_{0,k} + H_0) \right\}$$

$$v - 0 \leq i \leq n, m + R + 1 \leq k \leq m + R + q, \quad (5.2)$$

对备选集里的每一个备选结束时刻执行可调度性分析和丢包算法仍然十分耗时并且可能是不必要的，反而，我们可以利用以下无遗留包时刻的定义直接找到最优的结束时刻。

定义 5.5 (无遗留包时刻): t_{np} 是一个无遗留包时刻当且仅当所有在 t_{np} 时刻的遗留包都能够根据EDF算法在 t_{np} 之前完成。(这里EDF策略需要注意的是，如果一个包错过了截止期，测它没有执行的部分被丢掉，而执行的部分则被保留。)

根据无遗留包时刻 t_{np} 的定义，没有任务工作量将会被带入到 t_{np} 时刻之后的系统中。这跟可行的结束时刻定理中的条件5.3是一致的。此外，对于一个满足条件5.3和5.3的无遗留包时刻， t_{np} 应当大于等于 $\min(f_{0,m+R}, d_{0,m+R})$ 。根据这些限制， τ_0 所有在 $[t_{sp}, t_{np})$ 内释放的包都有足够的时间在截止期前完成。因此，任意一个无遗留包时刻 t_{np} ($t_{np} \geq \min(f_{0,m+R}, d_{0,m+R})$) 都可以是一个备选结束时刻。以下定理则说明最早的一个满足以上条件的无遗留包时刻是能够产生最少丢包的结束时刻。

定理 5.5: 在 $\min(f_{0,m+R}, d_{0,m+R})$ 之后的第一个无遗留包时刻相比所有 $\Gamma(t_{ep})$ 内的其它时刻能够使得丢弃的周期包的数量最少。

证明: 令 t_{np} 为 $\min(f_{0,m+R}, d_{0,m+R})$ 之后的第一个无遗留包时刻， t_1 和 t_2 分别是任意一个在 t_{np} 之前和之后的时刻，由一个最优的丢包算法所相应生成的丢包集合分别为 $\rho^*[t_{sp}, t_1)$ ， $\rho^*[t_{sp}, t_2)$ 和 $\rho^*[t_{sp}, t_{np})$ 。我们首先证明 $|\rho^*[t_{sp}, t_{np})| \leq |\rho^*[t_{sp}, t_1)|$ 。根据可行的结束时刻所需要满足的条件， t_1 保证所有在其之后释放的所有为丢弃的周期包以及节奏包都满足它们的正常及节奏截止期。当将 t_{np} 作为结束时刻时，如果我们直接丢掉跟 $\rho^*[t_{sp}, t_1)$ 相同的包集合，所有定理5.3中的条件都能够满足。这表明 $\rho^*[t_{sp}, t_1)$ 同样是一个当将 t_{np} 作为结束时刻时的可行解。由于 $\rho^*[t_{sp}, t_{np})$ 是由一个最优的丢包算法所生产，那么一定满足 $|\rho^*[t_{sp}, t_{np})| \leq |\rho^*[t_{sp}, t_1)|$ 。

接下来我们通过反证法来证明 $|\rho^*[t_{sp}, t_{np})| \leq |\rho^*[t_{sp}, t_2)|$ 。假设 $|\rho^*[t_{sp}, t_{np})| > |\rho^*[t_{sp}, t_2)|$ 。 $\rho^*[t_{sp}, t_2)$ 包含在 t_{np} 之前释放的以及在 t_{np} 时刻或之后释放的丢弃包，

分别表示为 S_1 和 S_2 。也就是说， $|\rho^*[t_{sp}, t_2]| = |S_1| + |S_2|$ 。根据假设，我们有 $|S_1| < |\rho^*[t_{sp}, t_{np}]|$ 。由于 $\rho^*[t_{sp}, t_2]$ 是一个可行解， S_1 应当保证所有释放时间和截止期都在 t_{np} 之前的节奏包和未被丢掉的周期包都满足它们的截止期。并且，根据无遗留包时刻的定义，所有在 t_{np} 时刻的遗留任务都能在 t_{np} 之前完成。因此当将 t_{np} 作为结束时刻时， S_1 也是一个可行解，且如假设 $|S_1| < |\rho^*[t_{sp}, t_{np}]|$ 。这显然跟 $\rho^*[t_{sp}, t_{np}]$ 是由最优算法所生成的事实相矛盾。因此定理得证。 \square

根据定理5.5，如果在 $\min(f_{0,m+R}, d_{0,m+R})$ 之后 t_{ep}^u 之前存在一个无遗留包时刻，我们可以直接将它作为结束时刻。结合引理5.4，公式5.2，以及定理5.5，我们设计了算法5来确定系统节奏模式的结束时刻。

5.3.3 丢包算法

在实时调度领域，最小化迟到实例的问题已经被很好地研究过^[153-155]。本节介绍如何将我们形式化定义的最小化丢包问题映射到最小化迟到实例的问题上，之后便可以利用Lawler算法^[153]在多项式时间内得到解。

根据处理器环境、作业实例限制和调度标准，最小化迟到实例的问题可以分为不同的类别。这三个参数可以表示为 $C_1|C_2|C_3$ ^[156]。我们的丢包问题可以被规约到文章^[153]研究的问题P2 ($1 | pmtn, r_j | \sum w_j \times L_j$)上。

定义 5.6 (P2): 给定一个任务实例集合 τ_j ，每个实例的释放时间 r_j 、执行时间 e_j 、截止期 d_j 和权值 w_j ，在单处理器上找到一个可抢占的调度表使得所有迟到实例权值总值最小。如果一个实例在其截止期之前完成，则其是及时的 ($L_j = 0$)；否则它是一个迟到实例 ($L_j = 1$)。

为了实现映射，我们首先介绍对于每个备选结束时刻 $t_{ep}^c \in \Gamma(t_{ep})$ 的活跃包的概念。

定义 5.7 (活跃包): 对于时刻 t ， $\chi_{i,k}$ 是一个活跃包当且仅当以下两个条件之一满足：(1) $t_{sp} \leq r_{i,k} < t$, and (2) $t_{sp} < d_{i,k} \leq t$ 。

现在令 $\Psi(t_{ep}^c)$ 为包含所有需要在 $[t_{sp}, t_{ep}^c]$ 内被调度的活跃包集合。根据定义5.7， $\Psi(t_{ep}^c)$ 包含以下包：(i) $\chi_{i,k, s.t., t_{sp} \leq r_{i,k} < d_{i,k} \leq t_{ep}^c$ 。(ii) 在开始时刻 t_{sp} 的遗留包，即 $\chi_{i,k, s.t., r_{i,k} < t_{sp}$ 并且 $d_{i,k} > t_{sp}$ ，这些包的释放时间设置为 t_{sp} ，执行时间设置为它们在 t_{sp} 时刻的剩余执行时间（即跳数）。(iii)在结束时刻 t_{ep}^c 的遗留包，即 $\chi_{i,k, s.t., r_{i,k} < t_{ep}^c$ 并且 $d_{i,k} > t_{ep}^c$ ，根据定理5.3中的条件5.3和5.3，所有在 t_{ep}^c 时刻的遗留包的截止期都设置为 t_{ep}^c ，执行时间设置为 H_i 。需要注意的是，由于广播任务负责向网络中的节点传播信息，理想情况下，任何广播包都不应该被丢弃。但是，如果调度某个

算法 5 Determining End Point of Rhythmic Mode

- 1: Construct EDF schedule for all the packets released before t_{ep}^u
 - 2: $\mathcal{S} \leftarrow \{t_{np} \mid \min(f_{0,m+R}, d_{0,m+R}) \leq t_{np} \leq t_{ep}^u\}$
 - 3: **if** $\mathcal{S} \neq \emptyset$ **then**
 - 4: $t_{np} \leftarrow \min(\mathcal{S})$
 - 5: **if** no deadline misses before t_{np} **then**
 - 6: **return** {NULL}
 - 7: **end if**
 - 8: **return** $\Gamma(t_{ep}) = \{t_{np}\}$;
 - 9: **end if**
 - 10: Construct $\Gamma(t_{ep})$ according to Equation (5.2)
 - 11: **return** $\Gamma(t_{ep})$
-

广播包而导致节奏包错失截止期，那么该广播包必须被丢弃以保证出现的干涉能够被正确的处理。因此，为了保证只有在必要的时候我们才丢弃广播包，当在决定哪些包需要被丢弃时，我们设置广播任务的优先级高于其它所有周期任务。

我们的丢包问题映射到**P2**的过程如下。 $\Psi(t_{ep}^c)$ 中的每个释放时间 $r_{i,k}$ 、截止期 $d_{i,k}$ 和跳数 H_i 的包 $\chi_{i,k}$ 映射到一个具有相同释放时间、截止期和执行时间的任务实例。（一个在CPU上执行单位时间的任务实例等价于实时无线传感网络的一个包的传输。）令 n_p 和 n_b 分别表示 $\Psi(t_{ep}^c)$ 中的周期包和广播包的数量。不同类型包的权值设置如下。（i）每个周期包的权值设置为1。（ii）每个广播包的权值设置为 $n_p + 1$ 。（iii）每个节奏包的权值设置为 $n_p + n_b \cdot (n_p + 1) + 1$ 。给每个广播包设置比所有周期包权值总和还大的权值保证了丢掉任意一个广播包比丢掉所有周期包都要“昂贵”。此外，每个节奏包的权值的设置也是为了保证只有在丢弃掉所有周期包和广播包后系统仍不可调度时才会考虑丢弃节奏包。由于每个节奏包的执行时间都小于等于截止期，即使所有周期包和广播包都被丢掉之后，也不会有节奏包需要被丢弃。至此，一个能够最小化迟到实例权值总和的调度表也是一个丢包数量的调度表。利用[153]中的Lawler算法，这样一个调度表可以在 $O(n_\Psi^5)$ 时间内找到， n_Ψ 是 $\Psi(t_{ep}^c)$ 中的包的数量。

时间复杂度为 $O(n_\Psi^5)$ 的Lawler算法可以在实际执行时非常耗时，因为对于每个备选结束时刻 t_{ep}^c 的活跃包集合的大小可能非常大，并且 t_{ep}^c 的数量可能也不小。因此，我们还是希望能够设计一个更有效率的丢包算法。一个重要的观察是，对于相同的利用率，一个执行时间更短的任务在一段固定时间内比一个执行时间长的任务能够释放更多数量的包。由于最小化丢包数量等价于最大化可调度包数量，

一个直观的启发式算法思想是按照执行时间从小到大的顺序调度包，以此具有更高的可能性调度尽可能多数量的包。我们设计的启发式算法的细节如算法6所示。初始的，算法将活跃包集合 $\Psi(t_{ep}^c)$ 中的所有节奏包存入已调度包集合 \mathcal{SPS} (1-2行)，为了保证网络中更新信息的即使传输，我们在周期包前优先调度广播包 (5-9行)。之后我们循环的选择执行时间最短的包 χ_s ，并且检查它是否可调度。如果是，则 χ_s 被保留在可调度包集合 \mathcal{SPS} 中。否则， χ_s 被丢掉 (11-15行)。最终，算法返回丢包集合 $\rho[t_{sp}, t_{ep}^c]$ (17行)。利用该提出的启发式算法，一个可行的调度表可以在 $O(n_{\Psi}^2)$ 时间内找到。

我们的丢包问题呈现出同非精确计算调度问题^[157]的相似性，以下定理给出了我们启发式算法的近似率 (证明见[157])。

定理 5.6: 对于任意实时无线传感网络和给定的 t_{sp}, t_{ep}^c ，我们提出的启发式所生成的丢包数量至多是最优算法所生成的两倍。

算法7总结了网关节点 V_g 如何确定哪些周期包需要丢弃掉来解决问题P1。对于每个算法5确定的备选结束时刻， V_g 首先确定相应的活跃包集合，然后利用一个丢包算法 (Lawler算法或本章提出的启发式算法) 来生成一个丢包集合 $\rho[t_{sp}, t_{ep}^c]$ (2-3行)。如果 $|\rho[t_{sp}, t_{ep}^c]| \leq \Delta^d$ ，该丢包集合被保存在 \mathcal{S} 中。最终，算法返回 \mathcal{S} 中丢包数量最小的解 (第6行)。如果丢包数量超过了最大允许的丢包数 Δ^d ， D^2 -PaS将 $\Gamma(t_{ep})$ 中最早的值作为备选结束时刻，并且把所有相应的活跃包都丢掉。

5.4 处理并发干涉

在上一节中，我们假设在任意时刻，最多只有一个任务处在它的节奏状态下，这一假设简化了 D^2 -PaS对干涉的处理。但是，在真实的实时无线传感网络中，多个干涉可能共同发生，或者系统的节奏模式可能发生重叠。在本节中，我们一般化系统模型，允许多个任务同时处在它们的节奏状态，并且扩展 D^2 -PaS来处理并发节奏事件。因为并发出现的节奏事件都具有高优先级，它们也会竞争网络资源。因此，我们不能简单的直接应用5.3.3节中介绍的丢包策略来处理。总的来说， D^2 -PaS依然能够处理并发的干涉，只是我们需要应对两个具体的挑战。(i) 当出现并发干涉时，何时以及如何来计算和更新调度表。(ii) 当多个任务同时处于它们的节奏状态时，如何保证系统总是可靠的，即不会发生运行错误。

下面我们以两个并发出现的干涉为例，来介绍我们如何解决以上问题 (该技术可以被轻易地扩展到处理任意多数量的并发干涉)。为了简化描述，我们根据节奏事件请求到达网关节点的时间以及每个节奏事件所引发的系统节奏模式的

算法 6 Packet Dropping using our Heuristic

Input: $\Psi(t_{ep}^c)$

Output: $\rho[t_{sp}, t_{ep}^c]$

```

1:  $\mathcal{SPS} \leftarrow \{\text{all rhythmic packets in } \Psi(t_{ep}^c)\}; // \text{ scheduled packet set}$ 
2:  $\Psi(t_{ep}^c) \leftarrow \Psi(t_{ep}^c) \setminus \mathcal{SPS};$ 
3:  $\rho[t_{sp}, t_{ep}^c] \leftarrow \emptyset;$ 
4: while  $\Psi(t_{ep}^c) \neq \emptyset$  do
5:   if Broadcast packet exists in  $\Psi(t_{ep}^c)$  then
6:     Add the broadcast packet  $\chi_s$  with the shortest execution time in  $\Psi(t_{ep}^c)$  to  $\mathcal{SPS};$ 
7:   else
8:     Add the unicast packet  $\chi_s$  with the shortest execution time in  $\Psi(t_{ep}^c)$  to  $\mathcal{SPS};$ 
9:   end if
10:   $\Psi(t_{ep}^c) \setminus \{\chi_s\};$ 
11:  if  $\mathcal{SPS}$  is schedulable under EDF then
12:    Continue;
13:  end if
14:   $\mathcal{SPS} \leftarrow \mathcal{SPS} \setminus \{\chi_s\};$ 
15:   $\rho[t_{sp}, t_{ep}^c] \leftarrow \rho[t_{sp}, t_{ep}^c] \cup \{\chi_s\};$ 
16: end while
17: return  $\rho[t_{sp}, t_{ep}^c];$ 

```

开始时可，将两个并发节奏事件的相对位置归为三种情况。如图5.7所示为针对节奏任务 τ_1 和 τ_2 的三种情况。不失一般性的，我们假设任务 τ_1 对应的节奏事件请求先到达网关节点 V_g 。在图5.7中， t'_i 表示网关节点收到任务 τ_i 对应的节奏事件请求时刻； t''_i 表示网关节点开始向网络中节点广播生成的处理节奏任务 τ_i 的丢包信息时刻； t_{sp}^i 和 t_{ep}^i 分别表示由 τ_i 所引发的系统节奏模式的开始时刻和结束时刻。我们称 $[t_{sp}^i, t_{ep}^i]$ 为 τ_i 的节奏窗口，其中所有 τ_i 释放的包都是不能丢弃的节奏包。从图5.7中可以轻易看出，所有其它 t'_1 和 t'_2 以及 t_{sp}^1 和 t_{sp}^2 的相对位置情况都能够对应到图中三种情况之一。

情况1: $t'_1 < t'_2 \leq t'' - 1$ (见图5.7-(a))，即两个节奏事件请求都在最近的一个广播时刻 t'' 之前到达网关节点 V_g （在此情况下，我们不需要在符号上区别两个干涉，即 $t'' = \{t''_1 \text{ 或 } t''_2\}$ ， $t_{sp} = \{t_{sp}^1 \text{ 或 } t_{sp}^2\}$ 以及 $t_{ep} = \{t_{ep}^1 \text{ 或 } t_{ep}^2\}$ ）。由于处理 τ_1 对应的节奏事件的更新调度表信息还未被广播除去，在 V_g 又收到 τ_2 的节奏事件请求后，它可以根 τ_1 和 τ_2 的信息直接重新计算更新的调度表信息。具体的， V_g 确定一段共有的

算法 7 Determining Dropped Packets at Gateway

Input: $\Gamma(t_{ep})$

Output: $\rho[t_{sp}, t_{ep}]$

- 1: **for** ($\forall t_{ep}^c \in \Gamma(t_{ep})$) **do**
- 2: Construct $\Psi(t_{ep}^c)$; // active packet set
- 3: Generate a dropped periodic packet set $\rho[t_{sp}, t_{ep}^c]$ based on $\Psi(t_{ep}^c)$ using Lawler's algorithm or our heuristic;
- 4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\rho[t_{sp}, t_{ep}^c]\}$;
- 5: **end for**
- 6: **return** $\arg \min_{\rho[t_{sp}, t_{ep}] \in \mathcal{S}} |\rho[t_{sp}, t_{ep}]|$;

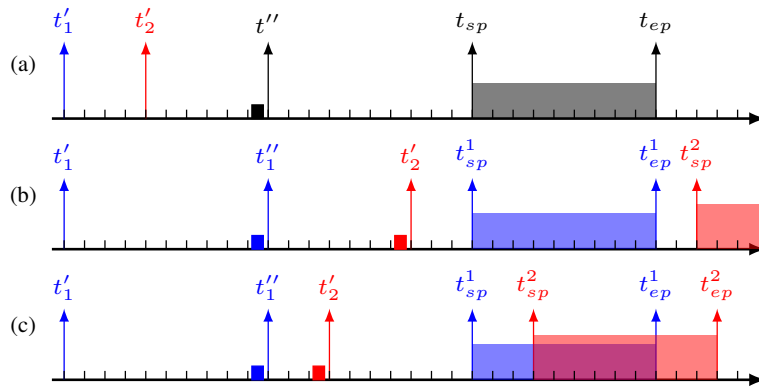


图 5.7 三种并发干涉相对位置情况

Fig. 5.7 (a) Case 1, $t'_1 < t'_2 \leq t'' - 1$; (b) Case 2, $t'_2 > t''_1$ and $t''_{sp} \geq t''_{ep}$; (c) Case 3, $t'_2 > t''_1$ and $t''_{sp} < t''_{ep} < t''_{ep} - 1$. Each block before t''_i denotes the time slot when the gateway handles the rhythmic event of τ_i . Each rectangle between t''_{sp} and t''_{ep} denotes the rhythmic window of τ_i .

节奏窗口 $[t_{sp}, t_{ep})$ ，计算丢包信息以保证两个任务释放的节奏包都能够在 $[t_{sp}, t_{ep})$ 内满足它们的截止期，然后将该信息一起打包，在 t'' 时刻广播发送给整个网络。需要注意的是，为了避免不必要的多余的计算，我们不再让网关节点在收到节奏事件请求后立即做更新调度表的计算，而是等到下一个广播时刻之前的最后一个时间片（即 $t'' - 1$ 时刻）来针对所有当前出现的节奏事件做计算。这一原则也适用于以下两种情况（考虑到网关节点强大的计算能力，即使一个节奏事件请求刚好在 $t'' - 1$ 时刻到达 V_g ，改时间片剩余的时间也足够 V_g 完成相应的计算）。

情况2： $t'_2 > t''_1$ 且 $t''_{sp} \geq t''_{ep}$ （见图5.7-(b)），即 τ_2 对应的节奏事件请求到达网关节点 V_g 时，处理 τ_1 节奏任务的丢包信息已经广播出去了，并且两个节奏事件对应的节奏窗口没有重叠。这种情况下， V_g 可以简单的分别处理两个节奏事件，即分别在 $t''_1 - 1$ 和 $t''_2 - 1$ 时刻生成处理节奏任务 τ_1 和 τ_2 的丢包信息。

情况3： $t'_2 > t''_1$ and $t''_{sp} < t''_{ep} < t''_{ep} - 1$ （见图5.7-(c)），即两个节奏事件对应的节奏窗

口相互重叠。因为当 τ_2 对应的节奏事件请求到达网关节点 V_g 时，系统已经开始使用 τ_1 的动态调度表，此时 τ_2 的动态调度表计算需要将 τ_1 节奏事件的相关信息考虑在内，因为 τ_1 的丢包信息，即 $\rho_1[t_{sp}^1, t_{ep}^1)$ ，会影响结束时刻 t_{ep}^2 的选择以及活跃包集合 $\Psi_2(t_{ep}^2)$ 的确定。为了处理这种复杂的情况，我们按照算法3的流程来修改D²-PaS。

(1) 在算法3的第4行，网关节点 V_g 检查系统是否过载，并且确定由 τ_2 对应的节奏事件引发的系统节奏模式的结束时刻，即 t_{ep}^2 。如图5.7-(c)所示，系统在 $[t_{sp}^1, t_{sp}^2)$ 内使用 τ_1 的动态调度表，之后从 t_{sp}^2 后使用更新的调度表。因此，当 V_g 运行算法5来确定 t_{ep}^2 时，需要去建立系统在 $[t_{sp}^2, t_{ep}^2)$ 内的EDF调度表，此处有两个地方细节应当修改：(a) 从 t_{sp}^1 时刻开始， τ_1 开始采用它的节奏周期和截止期；(b) 对 $\rho_1[t_{sp}^1, t_{ep}^1)$ 内任意一个丢掉的包 $\chi_{i,k}$ ，如果 $r_{i,k} < t_{sp}^2$ ，则丢掉 $\chi_{i,k}$ 。否则， $\chi_{i,k}$ 被保留，因为 t_{sp}^2 之后的调度表会被更新。

(2) 如果系统是过载的， V_g 确定在 $[t_{sp}^2, t_{ep}^2)$ 内需要丢掉的周期包（算法3的第5-6行）。根据算法7的第2行， V_g 建立活跃包集合 $\Psi_2(t_{ep}^2)$ 。相似的，考虑 τ_1 的丢包信息，在 $\rho_1[t_{sp}^1, t_{ep}^1)$ 内的任意 t_{sp}^2 时刻的遗留任务应当从 $\Psi_2(t_{ep}^2)$ 内移除，并且所有 t_{sp}^2 时刻之后释放的包必须包含在 $\Psi_2(t_{ep}^2)$ 内，即使它们之前在处理 τ_1 时被丢弃了。

(3) 根据 $\Psi_2(t_{ep}^2)$ ， V_g 运行算法6来生成 $[t_{sp}^2, t_{ep}^2)$ 内需要丢弃的周期包，来保证两个节奏任务 τ_1 和 τ_2 所有释放的包的截止期。这里，我们称任务 τ_i 在其节奏窗口内，即 $[t_{sp}^i, t_{ep}^i)$ ，释放的包为节奏包。

根据以上介绍的D²-PaS的扩展，我们解决了何时以及如何更新动态调度表来处理新到达的 τ_2 对应的节奏事件。下面我们考虑当 τ_1 和 τ_2 同时在它们的节奏状态时，即情况1中的 $[t_{sp}, t_{ep})$ 以及情况3中的 $[t_{sp}^2, t_{ep}^1)$ ，如何保证系统总是可靠的。这里我们所说的可靠是指，不管任何状态下，所有节奏包的截止期都能够满足。

当假设系统在运行过程中任何时刻最多只有一个任务处在节奏状态时，如果丢掉所有的周期包，那么所有节奏包的截止期都一定能够满足，因为每个任务在正常状态和节奏状态时的执行时间都小于等于截止期，即 $H_i \leq \min(D_i, D_{i,x} | x = 1, \dots, R_i)$ 。因此，在D²-PaS下系统是可靠的。但是，如果多个任务并发的进入了它们的节奏状态，那么系统可能还是会过载即使把所有其它的周期包都丢掉。为了解决这一问题，我们允许网关节点延迟任务进入它的节奏状态的时间，并且对于所有等待处理的干涉采取先来先服务（First-Come-First-Serve, FCFS）的方式进行这种延迟。（这类似于一种访问控制策略，也可以采用其他类似的优先级策略。）具体的，假设多个节奏事件请求在 $t'' - 1$ 时刻前到达网关节点（见图5.7-(a)）。

V_g 按照所有这些干涉出现的顺序对它们进行排序，之后顺序的确定每个节奏任务的 $t_{n \rightarrow r}$ 。如果 V_g 检测到允许一个任务（假设 τ_0 ）在 t_{sp} 之后它的第一个释放时间转换到节奏状态（即 $t_{n \rightarrow r} = r_{0,m+1}$ 如图5.2所示），会导致节奏包错失截止期即使所有周期包都被丢弃的话，网关节点 V_g 延迟 $t_{n \rightarrow r}$ 到 τ_0 的下一个释放时间 $r_{0,m+2}$ ，然后再做出相同的检测。 V_g 重复此过程直到找到 τ_0 的一个释放时间能被设置为 $t_{n \rightarrow r}$ 。之后网关节点为队列里的下一个节奏任务设置 $t_{n \rightarrow r}$ 。

虽然延迟一个任务进入到它节奏状态的时间会增加处理干涉的延迟，但是它会避免系统由于没有处理好干涉事件而崩溃。总结以上所有的讨论，算法8给出了处理两个并发干涉的伪代码，并且可以被轻易扩展到处理任意多数量的并发干涉。

5.5 测试平台实现

我们将本章设计的D²-PaS框架实现在了一个真实的实时无线传感网络测试平台上，以验证其在真实环境下的实用性。我们的测试平台是基于OpenWSN^[158]协议栈，并且进行了一些必要的改进来支持D²-PaS框架。如图5.8-(a)所示，一个OpenWSN网络由多个OpenWNS设备、一个OpenWSN根节点、以及一个OpenLBR (Open Low-Power Border Router)组成。根节点和OpenLBR通过一个有限连接(例如UART)用OpenBridge协议进行通讯，它们共同组成了网关节点 V_g 。如图5.8-(b)所示，我们的测试平台包含7个无线设备（TI CC2538 SoC + SmartRF06测试板），其中一个配置为根节点并且连接到运行在一个Linux机器上的OpenLBR。其它设备配置为设备节点，组成一个多跳的实时无线传感网络。我们使用一个8通道的逻辑分析器来验证系统的运行过程。由于本节测试平台的实现和评估并不是主要由作者本人完成的，因此我们不作详细的介绍，感兴趣的读者可以参考文章[159]。

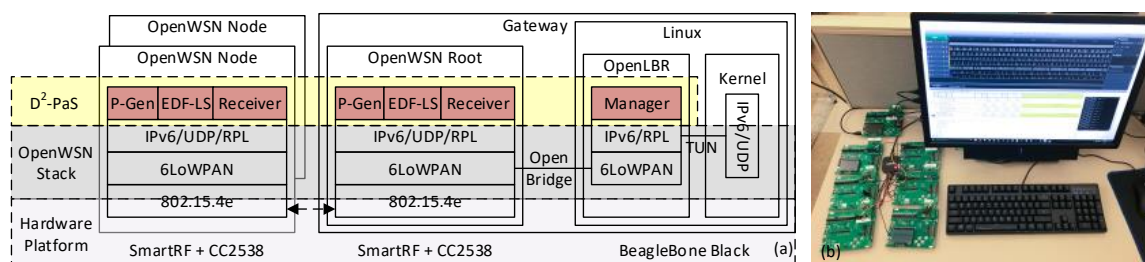


图 5.8 实时无线传感网络测试平台

Fig. 5.8 An overview of the real-time wireless network testbed: (a) the system software architecture with the D²-PaS implementation being highlighted; (b) a picture of the hardware components of the testbed.

算法 8 Handling Two Concurrent Disturbances

```

1: At  $t_2'' - 1$ , a rhythmic event request from  $\tau_2$  is to be handled;
2: if  $t_1' < t_2' \leq t'' - 1$  (Case 1) then
3:   Handle both disturbances together;
4: end if
5: if  $t_2' > t_1''$  then
6:   Determine  $t_{sp}^2$  according to the upcoming broadcast packet;
7:   if  $t_{sp}^2 \geq t_{ep}^1$  (Case 2) then
8:     Handle  $\tau_2$  independently;
9:   end if
10:  if  $t_{sp}^1 < t_{sp}^2 < t_{ep}^1$  (Case 3) then
11:    while the system cannot admit  $\tau_2$  to enter its rhythmic state at  $t_{n \rightarrow r}^2$  do
12:       $t_{n \rightarrow r}^2 + = P_2$ ;
13:    end while
14:    Handle  $\tau_2$  with  $\tau_1$ 's rhythmic information considered;
15:  end if
16: end if

```

5.6 模拟实验

在本节中，我们总结验证D²-PaS性能的模拟实验结果。在模拟实验中，我们将本章提出的D²-PaS与当前最优的方法OLS方法相比较。因为OLS并不支持处理并发节奏事件，我们首先考虑单个干涉模型，即任何时刻最多只有一个任务在其节奏状态。之后我们扩展OLS使其能够处理并发干涉，然后再来比较它们的性能。

5.6.1 实验配置

为了更好的控制系统工作量，我们改变部署在实时无线传感网络上的任务集的正常利用率（即所有任务都在正常状态下系统的利用率）。具体的，我们根据一个目标正常利用率 U^* 来生成随机周期任务集，每个任务集都是从一个初始的空集 \mathcal{T} 开始，然后向其中逐个添加随机任务。

每个周期任务 τ_i 根据以下参数生成：(i)跳数 H_i 从 $\{2, 3, \dots, 10\}$ 中随机取值，(ii)周期 P_i 等于截止期 D_i ，并从 $\{15, 16, \dots, 50\}$ 中随机取值（我们有意的并没有指定值 P_i 和 D_i 的单位，因为只有它们的相对值对可调度性研究有意义，而我们实验中的取值是考虑了实际中实时无线传感网络的设置）。一个任务集生成后，我们随机选择其中一个任务作为节奏任务 τ_0 ，它的节奏周期 \vec{P}_0 ($\vec{P}_0 = \vec{D}_0$)由

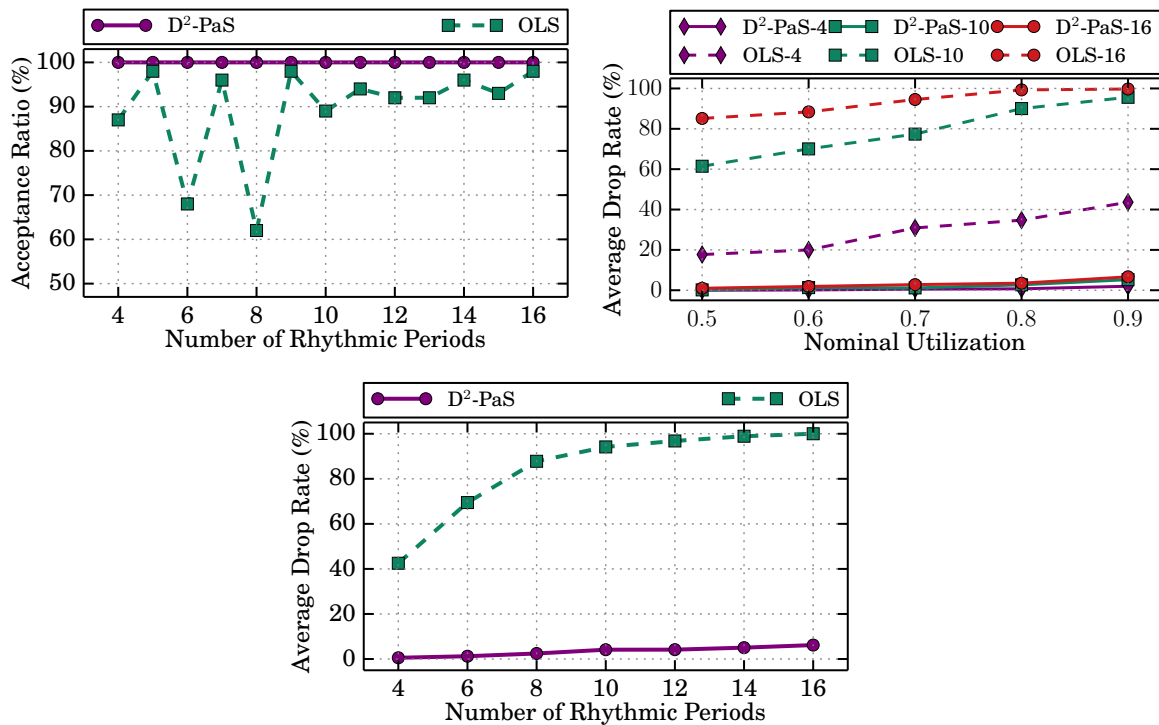


图 5.9 模拟实验结果。(a) 正常利用率 $U^* = 0.5$ 的接受率对比；(b) 不同 R_0 设置下的平均丢包率对比；(c) 正常利用率 $U^* = 0.9$ 的平均丢包率对比。

Fig. 5.9 Simulation results. (a) Comparison of the acceptance ratio with a nominal utilization $U^* = 0.5$; (b) Comparison of the avg. drop rate under different settings of R_0 (denoted as D²-PaS- R_0 or OLS- R_0 in the legend); (c) Comparison of the avg. drop rate with a nominal utilization $U^* = 0.9$.

以下参数控制生成：(i) 首个节奏周期比 $\gamma = P_{0,1}/P_0$ 固定为0.2，(ii) \vec{P}_0 中包含元素的数量 R_0 从 $\{4, 6, \dots, 16\}$ 中随机取值。我们固定 γ 的值，用 R_0 来调节节奏任务在系统处于节奏模式下时的工作量，因为 R_0 能够提供更好对节奏任务工作量的控制。我们假设系统进入节奏模式后，节奏任务周期 $P_{0,k}$ 根据 γ 首先从 P_0 减少到 $P_{0,1}$ ，然后逐渐的递增直到回到原始的正常周期值 P_0 ，递增的过程服从 $P_{0,k} (1 \leq k \leq R_0) = \lfloor P_0 \times (\gamma + (k-1) \times \frac{1-\gamma}{R_0}) \rfloor$ 。

D²-PaS和OLS的一些其它参数设置如下。(i) 系统节奏模式的开始时刻 t_{sp} 从 $\{50, \dots, 200\}$ 中随机选择。(ii) 转换时刻比率因子 α 设置为2（文章[105]中OLS同其他方法比较时也设置为2）。 α 决定了OLS中转换时刻的上限，即 $t_{sw}^u = t_{r \rightarrow n} + (\alpha - 1) \times P_0$ 。为了保证公平比较，我们同样设置D²-PaS中的结束时刻上限 $t_{ep}^u = t_{sw}^u$ 。(iii) 一个广播包最大负载信息量为90字节。为了表示调度表中的一个更新时间片，OLS需要3个字节，其中分别需要13位、7位和4位来表示时间片ID、任务ID和当前跳数。这意味着OLS中最大允许的更新时间片数量 Δ^u 等于30。而对于D²-PaS，我们只需要两个字节来表示一个丢弃的包，其中分别需要7位和9位来表示任务ID和包ID。因

此D²-PaS中的最大允许丢包数量 $\Delta^d = 45$ 。(iv) 另外一个OLS需要的参数是动态规划中最大允许保留子调度表数量 β 。为了公平比较,我们设置 $\beta = 40$,这是[105]中发现的OLS最优性能的 β 值设置。

由于D²-PaS和OLS中丢包算法的性能取决于任务集设置而非网络拓扑,因此我们仅控制任务集的生成并且假设同一对比集中网络拓扑是相同的。

5.6.2 处理单个干涉的实验

我们通过以下三个指标比较D²-PaS和OLS在处理单个节奏事件的性能:(i) 接受率(acceptance ratio, AR),即可调度任务集数量与总任务集数量之比,一个系统是可调度的当且仅当所有节奏包的截止期都能够得到满足;(ii) 平均丢包率(average drop rate, DR),定义为丢包数量与 $[t_{sp}, t_{ep}]$ 内总的活跃包数量之比;(iii) 计算时间(computation overhead, CO),即处理一个节奏事件所需要的时间。

图5.9-(a)所示为对于正常利用率 $U^* = 0.5$ 的系统,接受率AR随 R_0 (\vec{P}_0 中包含的周期值数量)变化的曲线。其它系统利用率的比较结果类似,因此在此省略。图中每个数据点都是1000个随机实验产生的结果取平均值所得。我们从图中可以看出OLS并不能总是找到一个可行的转换时刻来满足所有节奏包的截止期约束,它的接受率在90%左右,但是最低达到60%。而另一方面,D²-PaS能够达到100%的接受率,因为在D²-PaS中总是存在至少一个可行的结束时刻。

图5.9-(b)所示为不同 R_0 设置下的平均丢包率随正常系统接受率变化的曲线,其中实线代表OLS而虚线代表D²-PaS。可以清晰的看到在所有参数设置下D²-PaS的性能都要优于OLS。如图5.9-(b),当系统利用率小于90%时,D²-PaS的丢包率非常低(平均仅为1%)。这主要是因为一个节奏任务所带来的额外工作量在多数情况下不足以使得系统过载。因此,系统不需要丢掉任何周期包就能够处理干涉(在5.6.3节中,我们会介绍D²-PaS的丢包率会随并发节奏事件数量增加而升高)。然而,另一方面,随着 R_0 的增加,OLS的性能下降非常快。为了提供更细粒度的比较,我们也设置系统利用率 $U^* = 0.9$,然后把 R_0 的值从4增加到16来观察丢包率的变化,图5.9-(c)所示D²-PaS的性能大大优于OLS,丢包率的提升从40%一直打到95%。

我们还比较了D²-PaS和OLS处理一个干涉事件所需消耗的时间开销。执行时间是从一个Intel i3-2120 CPU (3.30GHz)上获得的。当系统利用率从50%增加到90%时,OLS的平均执行时间为2.2s到10.4s,而D²-PaS仅需不到1s来处理一个相同的干涉。这主要是由于OLS的核心是一个动态规划算法,当活跃包的数量以及备选

转换时刻数量非常大时，实际运行时间会非常就。高昂的时间开销的严重后果是OLS可能无法在下一个广播时刻到来之前生成动态调度表，因此无法及时的处理干涉。

5.6.3 处理并发干涉的实验

在本节中，我们评估D²-PaS在处理并发干涉时的性能，为了公平比较，我们扩展OLS使其支持对并发干涉的处理。具体的，因为OLS是一种集中式的调度方法，即依赖网关节点来处理所有的工作，在任意时刻当一个新的节奏事件请求到达时，网关节点仅需重新生成一个动态调度表并把它通过广播包发送给整个网络。跟D²-PaS相同，所有出现的干涉都是以先来先服务的原则被处理。

除去在前一小节中介绍的模拟实验中用到的参数以外，随机任务集生成用到的一些其它参数为：(i) 并发节奏事件数量 \mathcal{N}_r 。由于OLS是基于一个非常耗时的动态规划算法，我们只令 \mathcal{N}_r 等于2或3。(ii) 任务 τ_i 对应的节奏窗口的开始时刻 t_{sp}^i ，为了使比较更具代表性，我们令每个节奏任务对应的节奏窗口与之前一个相互重叠，即情况3的 $t_{sp}^i < t_{ep}^{i-1}$ 。具体的，处理第一个节奏事件的开始时刻从 $\{50, \dots, 200\}$ 中随机取值，每个之后的开始时刻 t_{sp}^i 则随机从 $\{t_{sp}^{i-1} + 1, \dots, t_{ep}^{i-1} - 1\}$ 取值。

图5.10-(a)所示为对于利用率 U^* 为0.9的系统，在不同 \mathcal{N}_r 设置下接受率随节奏周期数量 R_i 变化的曲线。如图所示，D²-PaS仍然能够达到100%的接受率，因为在D²-PaS中总是存在一个可行的结束时刻。然而，OLS并不能总是找到一个可行的转换时刻来保证所有节奏包的截止期约束。此外，当并发干涉的数量增加时，OLS的接受率也随之降低，因为此时OLS更难找到一个可行的转换时刻。

图5.10-(b)所示为在不同 \mathcal{N}_r 设置下丢包率随系统利用率 U^* 变化的曲线。因为我们更关心的是丢包率是如何随并发节奏任务变化而变化的，因此我们固定每个节奏任务的节奏周期数量 $R_i = 4$ 。如图5.10-(b)所示，增加并发干涉的数量时，D²-PaS和OLS的性能都会随之降低，但是D²-PaS比OLS降低的更慢，并且在所有设置下依然有非常大的性能提升。

5.7 小结

在本章中，我们介绍了D²-PaS，一个能够处理实时无线传感网络中干涉事件的分布式的动态包调度框架。不同于集中式的调度方法，由网关节点生成动态调度表然后广播给整个网络的方法，D²-PaS利用每个设备节点的计算能力，在本地

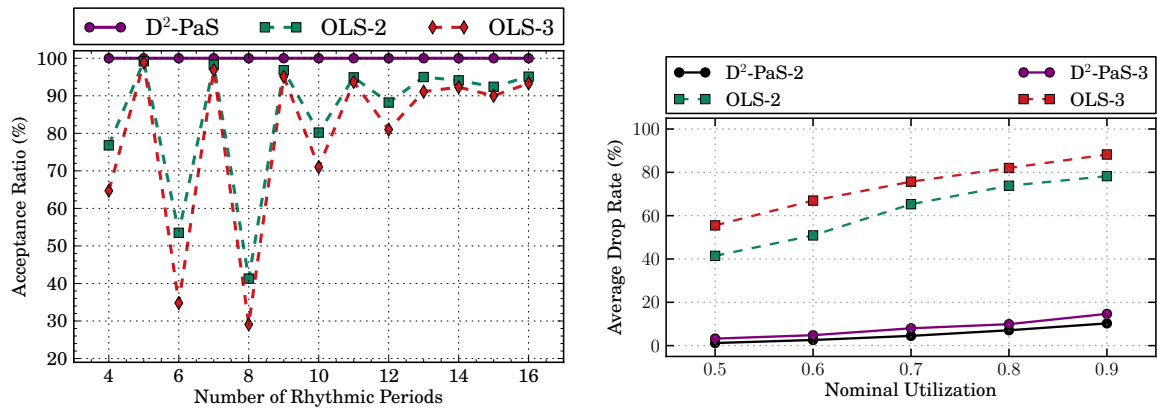


图 5.10 不同参数设置下处理并发干涉的实验结果

Fig. 5.10 Simulation results for handling concurrent disturbances under different settings of \mathcal{N}_r (denoted as $D^2\text{-PaS-}\mathcal{N}_r$ and $OLS\text{-}\mathcal{N}_r$). (a) Comparison of the acceptance ratio with a nominal utilization $U^* = 0.5$; (b) Comparison of the average drop rate with $R_i = 4$.

生成自己的调度表。网关节点只需运行一个轻量级的丢包算法然后在系统过载时将丢包信息广播给网络。大量的模拟实验和真实测试平台实验验证了D²-PaS的实用性和有效性。下一步，我们会将D²-PaS扩展到多信道网络中，并且考虑网络链路不可靠的情况，以及设计完全分布式的调度框架来摆脱对于网关节点的依赖。

第6章 处理实时无线传感网络中干涉的完全分布式包调度策略

正如我们在上一章讨论过的，实时无线传感网络在许多CPS应用中扮演着非常重要的作用，而为了能够达到要求的服务质量，包调度的问题则是重中之重。并且，随着当前CPS应用规模和复杂度的不断增大，以及实时无线传感网络中不时出现的干涉事件这一现象，包调度的问题的解决也变得愈发棘手。本章研究的重点依然是由网络监控的外部环境所引发的外部干涉事件，并且在上一章的研究基础之上，我们想要设计一种有效的完全分布式的包调度框架，使得能够被应用于大规模的实时无线传感网络，并且能够令人满意的处理网络中发生的干涉。

处理实时无线传感网络中干涉事件的主要挑战在于系统运行过程中干涉发生的不可预测性。具体的，通常我们并不知道何时以及哪一个干涉即将出现，以及干涉出现时网络的状态是怎样的（例如有多少包已经完成而多少包还未抵达执行器节点等）。从计算复杂度的角度来说，由于很难在网络开始运行之前遍历所有可能的干涉发生情况，研究者们转而利用在线动态调度方法来响应干涉所引发的系统工作量变化（如[102, 105, 160]）。最近的针对动态方法的研究工作就是上一章我们提出的分布式动态包调度框架D²-PaS^[159]，D²-PaS的主要思想是依赖于一个网络中的集中控制节点（如网关节点）在干涉出现时生成一个动态调度表，然后将最少量的信息通过广播包发送到整个网络中，当所有其他节点收到这一信息后，它们以一种分布式的方式在本地生成更新的调度表来响应干涉。虽然在丢包率方面D²-PaS具有非常好的性能，但由于依赖于网络中的一个集中控制节点，它所能应用的网络规模十分受限，尤其在当前实时无线传感网络的部署常常需要跨越很大的地理范围的背景下（例如在油田中部署上千个节点）。此外，D²-PaS需要一段非常长的响应时间来处理发生的干涉，因为系统只有在所有节点都通过广播包收到更新的调度表信息后，才能进入节奏模式开始处理干涉。所有这些缺陷都会导致干涉发生时网络性能降低。

在本章中，我们提出一种完全分布式的包调度框架FD-PaS（fully distributed packet scheduling framework），来处理实时无线传感网络中出现的干涉。跟D²-PaS相同，FD-PaS也利用同样的技术让网络中的每个节点在本地生成自己的调度表。但是，FD-PaS可以不依赖任何集中控制节点（如D²-PaS中的网关节点），直接当干涉

出现时在节点本地做出在线的决策。这一结果是通过只将出现的干涉信息利用运行任务的路由发送给网络中一部分节点实现的。在这种方式下，FD-PaS不再需要一个广播包来通知所有节点干涉的发生，这大大缩短了响应干涉的时间。为了保证这种干涉信息只传给部分节点的策略正确运行，我们需要克服一下一些问题。例如，为了避免由于节点间信息不对称所生成的调度表不一致而导致的传输冲突，我们在数据链路层提出一种多优先级无线包抢占机制MP-MAC，以此保证高优先级的包总是可以通过抢占低优先级的包传输而占据当前时间片。此外，为了最小化丢包数量，我们形式化了一个丢包问题来确定一段临时的动态调度表处理发生的干涉。我们首先证明了该问题是NP难的，然后介绍了一种最优的整数线性规划（ILS）方法，以及提出一种能够在设备节点本地高效运行的启发式算法。我们将设计的MP-MAC和动态调度表创建方法（它们共同构成了FD-PaS框架）实现在了一个真实的实时无线传感网络测试平台上，并且通过大量的模拟实验我们验证了FD-PaS的正确性以及提供对干涉快速响应上的有效性。

6.1 系统模型

本章采用一种典型的实时无线传感网络系统架构，网络由许多传感器节点和执行器节点通过无限的方式直接经由一个控制节点或经若干中间节点相连。（这里需要注意的是，控制节点主要负责网络初始化设计及运行控制计算，FD-PaS框架并不需要控制节点来做任何在线决策以及更新调度表之类的工作。）我们称所有非控制节点为设备节点，并且假设所有设备节点都具有路由功能，且都配备一个全向的天线，在单信道半双工模式下运行。整个网络可以由一个全连接图 $G = (V, E)$ 表示，节点集合为 $V = \{V_0, V_1, \dots, V_c\}$ ，其中 V_c 代表控制节点。 $(V_i, V_j) \in E$ 表示从节点 V_i 到节点 V_j 的一条可靠链路。

我们用任务的概念描述网络中从传感器节点发送到执行器节点的包。具体的，系统中运行着一个固定数量地任务集 $\mathcal{T} = \{\tau_0, \tau_1, \dots, \tau_n\}$ 。每个 τ_i 具有一条唯一的路由路径包含 H_i 跳。它周期性地从传感器节点采集数据后生成一个数据包，并按照其路由路径转发给控制节点 V_c ，之后将控制节点生成地控制指令传输给相应的执行器节点。

当网络外部干涉（如突然的温度或压力变化）发生时，很多CPS应用会需要更频繁地采样和控制，而这会导致网络资源需求量的增加。为了模型化的描述这种网络资源需求量的增加，我们依然采用节奏任务模型^[152]，因为它已经在处理实际事件驱动控制系统中的干涉方面得到研究者的认可^[105]（本章所提出

的FD-PaS方法不仅限于节奏任务模型，而是能够处理任意给定网络资源需求量变化的任务集)。在节奏任务模型中，每个单播任务 τ_i 具有两种状态：正常状态和节奏状态。在正常状态下， τ_i 按照正常周期 P_i 和正常截止期 $D_i \leq P_i$ 来运行。当一个跟任务 $D_i \leq P_i$ 相关的干涉出现时， τ_i 进入到节奏状态，此时它的周期和截止期先是为了应对干涉事件而骤减，然后再逐渐的递增直到恢复至原始的正常周期和正常截止期。我们用向量 $\vec{P}_i = [P_{i,x}, x=1, \dots, R_i]^T$ 和 $\vec{D}_i = [D_{i,x}, x=1, \dots, R_i]^T$ 来表示 τ_i 在节奏状态下周期和截止期变化的过程。一旦 τ_i 进入到节奏状态，它的周期和截止期就按照向量中的值逐渐变化，当它的周期和截止期恢复到正常值时， τ_i 也相应的回到正常状态且继续按照正常周期和截止期运行。

这里我们假设在系统运行过程中的任意时刻，至多只有一个任务在它的节奏状态。为了简化描述，我们称当前进入到节奏状态的任务为节奏任务（表示为 τ_0 ），所有其它保持在正常状态运行的任务称为周期任务（ $\tau_i (1 \leq i \leq n)$ ）。如图6.1所示，当 τ_0 进入到它的节奏状态时，我们称系统也相应的转换到节奏模式下运行以处理干涉事件。当处理完当前干涉后（通常是当相应的节奏任务回到它的节奏状态后的某一时刻），系统就回到正常模式下继续运行。因为干涉事件可能会对系统运行产生严重后果，当系统运行于节奏模式时，节奏任务的截止期为硬截止期，即必须满足，而其它周期任务可以允许截止期偶尔的错失。

每个任务 τ_i 都会释放任意多个任务实例， τ_i 的第 k 个实例我们称为一个包 $\chi_{i,k}$ ，它的释放时间为 $r_{i,k}$ ，绝对截止期为 $d_{i,k}$ ，以及完成时间（即包到达执行器的时刻）为 $f_{i,k}$ 。不是一般性的，我们假设节奏任务 τ_0 在 $r_{0,m+1}$ 时刻（表示为 $t_{n \rightarrow r}$ ）进入它的节奏状态，在 $r_{0,m+R_0+1}$ 时刻（表示为 $t_{r \rightarrow n}$ ）回到它的正常状态。因此， τ_0 在 $[t_{n \rightarrow r}, t_{r \rightarrow n})$ 区间内在它的节奏状态下执行， $t_{n \rightarrow r}$ 和 $t_{r \rightarrow n}$ 满足 $t_{r \rightarrow n} = t_{n \rightarrow r} + \sum_{x=1}^{R_0} P_{0,x}$ 。任意节奏任务 τ_0 在其节奏状态下释放的包我们都成为节奏包，相应的表示为 $\chi_{0,k}$ ，而任意其它周期任务 $\tau_i (1 \leq i \leq n+1)$ 释放的包称为周期包。包 $\chi_{i,k}$ 在其第 h 跳上的发送接收称为一个传输，表示为 $\chi_{i,k}(h) (1 \leq h \leq H_i)$ 。根据实际工业信息物理系统，我们的模型采用时分复用（Time Division Multiple Access, TDMA）的方式进行数据链路层传输，每个节点按照给定的调度表发送和接收包，每个传输 $\chi_{i,k}(h)$ 必须在一个单位时间片内完成。

基于以上系统模型描述，本章中我们想要解决的问题是：

问题1: 假设对于一个给定的实时无线传感网络，当系统没有干涉出现时，系统使用一个给定的静态调度表并且能够保证所有包都能在它们的正常截止期前完成。当在 $r_{0,m}$ 时刻（任务 τ_0 包的一个释放时刻，我们假设干涉只能在传感器

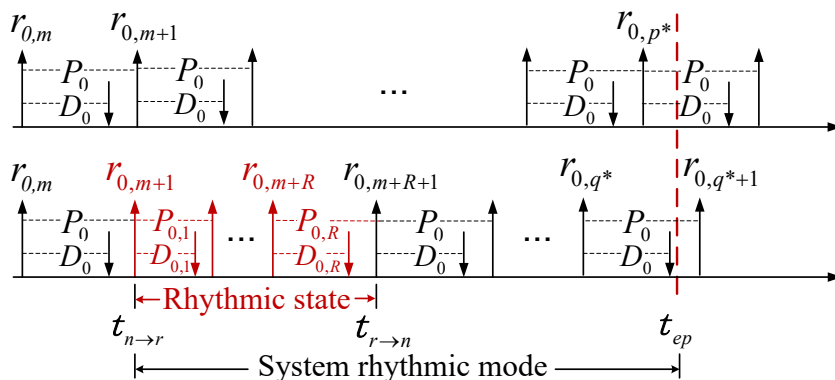


图 6.1 节奏任务模型示例。上下子图分别表示 τ_0 在正常状态和节奏状态下的释放时间和截止期。

Fig. 6.1 Timing parameters of the rhythmic task τ_0 in the system rhythmic mode. Top and bottom subfigures denote the nominal and actual release times and deadlines of τ_0 respectively.

采样环境信息时会检测到干涉，即某个包的释放时刻) 检测到一个干涉时，确定系统在节奏模式下的一段动态调度表，满足以下限制：(i) 系统可以在不晚于 $r_{0,m+1} = r_{0,m} + P_0$ 时刻前开始处理干涉，(ii) 所有节奏包都能够满足截止期，(iii) 在系统节奏模式时丢掉最少数量的周期包，(iv) 系统可以安全的转换回正常模式，并且所有包都能够满足它们的正常周期。

限制 (i) 保证了干涉可以在最早的可能时刻开始被响应（即任务的下一个正常释放时间）。如果限制 (i) 不能满足，相应的控制系统会运行不稳定或者遭受系统性能下降。限制 (ii) (iii) 和 (iv) 的意义如其本身所述。

6.2 FD-PaS框架概述

在本节中，我们首先给出一个例子来说明集中式调度在处理干涉时的缺陷，之后我们介绍本章提出的完全分布式包调度框架FD-PaS。

6.2.1 集中式方法的缺陷

为了正确的处理网络中出现的干涉，研究者们提出了集中式动态调度方法来应对网络资源需求量的变化。例如，OLS可以基于动态规划算法生成一个在线动态调度表来处理节奏事件描述的干涉。为了进一步提升性能，即减少丢包率，我们在上一节中提出了D²-PaS框架，利用实时无线传感网络的同步特性以及设备节点的计算能力在本地生成一致的调度表。通过有效的减少网关节点广播给网络的动态信息，D²-PaS大大的提高动态调度表生成的有效性。

然而，集中式方法会带来非常长的处理干涉的延迟时间，尤其是对于大规模的网络。考虑一个实时无线传感网络例子（见图6.2），该网络包含3个任

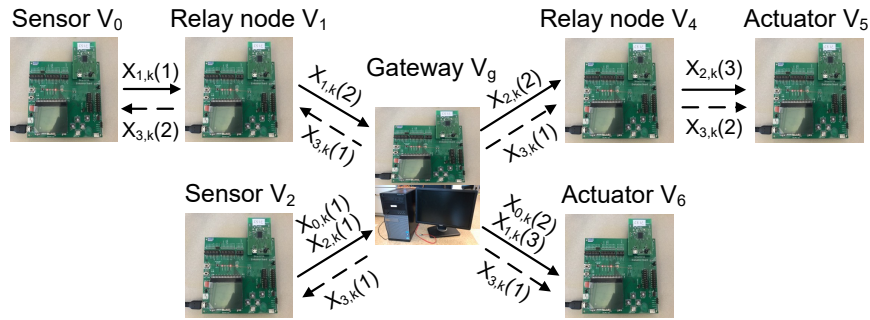


图 6.2 一个包含7个节点的实时无线传感网络例子

Fig. 6.2 An example RTWN with three unicast tasks and one broadcast task running on 7 nodes.

务 (τ_0, τ_1 和 τ_2) 运行在7个节点上 (V_0, \dots, V_5 以及 V_c), 其中 V_0 和 V_2 是传感器节点, V_4 和 V_5 是执行器节点, V_1 和 V_3 是中间结点, V_c 是控制器 (即集中式方法中的网关节点)。由于集中式方法依赖于控制节点向网络中广播动态调度部信息, 因此网络中还需要运行一个广播任务 τ_3 。所有任务的路由信息、周期和截止期在表6.1所示。

假设所有任务同时在0时刻释放它们的第一个包, 并且所有节点都根据EDF调度策略生成自己的本地调度表 (见图6.3)。假设在9时刻检测到一个外部干涉的出现, 传感器节点 V_2 通过任务 τ_0 向控制节点 V_c 发送一个节奏事件请求。之后 V_c 确定 τ_0 进入节奏状态的时刻 $t_{n \rightarrow r}$ 。为了快速地响应干涉, $t_{n \rightarrow r}$ 应该被设置的尽可能早, 但必须在系统中所有节点都收到动态调度表信息之后。在这个例子中, V_c 必须等到最近的一个广播时可 (即26时刻) 才能开始向网络中广播更新信息。之后只有等到所有节点都收到该信息后 (即时刻30), τ_0 才能从它的最近释放时刻36开始进入到节奏状态。也就是说, 在这个例子中, 虽然传感器节点在9时刻就已经检测到了干涉的发生, 但系统直到36时刻才进入到节奏模式开始处理干涉, 即整整3个 τ_0 的正常周期后。

从以上例子我们可以看出, 集中式方法在响应干涉时具有相当长的响应时间, 尤其对于大规模的网络。此外, 集中式方法依赖于网络中的某一节点 (如网

表 6.1 任务参数

Table 6.1 Task parameters for the motivational example

Task	Routing Path	$P_i (= D_i)$
τ_0	$V_2 \rightarrow V_c \rightarrow V_5$	9
τ_1	$V_0 \rightarrow V_1 \rightarrow V_c \rightarrow V_5$	9
τ_2	$V_2 \rightarrow V_c \rightarrow V_3 \rightarrow V_4$	10
τ_3	$V_c \rightarrow *$	18

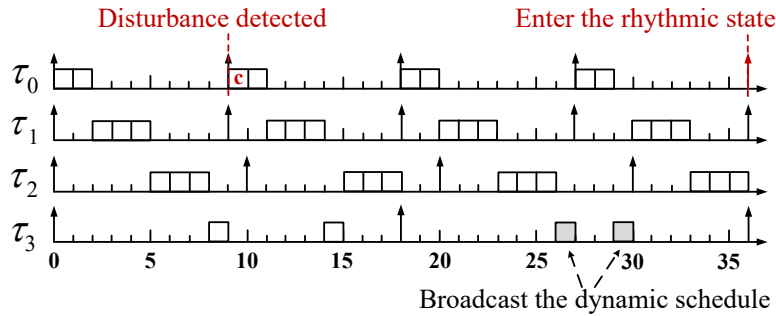


图 6.3 实时无线传感网络例子的本地EDF调度表

Fig. 6.3 Local EDF schedules of the tasks in the motivating example. The block with symbol c denotes the transmission of the rhythmic event request. The shaded blocks denote the two transmissions of the broadcast task to propagate the dynamic schedule generated at the controller node to the whole network.

关节点) 来做在线决策。这两个缺陷严重妨碍了将其部署在大规模实时无线传感网络中。

6.2.2 FD-PaS框架总览

为了在实时无线传感网络中快速的响应出现的干涉，本章我们提出一种完全分布式包调度框架FD-PaS。其主要思想是在节奏任务路径上的节点本地做在线决策和生成动态调度表，并且避免跟其它还在使用原始静态调度表的节点发生传输冲突。

图6.4所示为FD-PaS框架的系统运行过程模型。当系统启动后，每个节点利用 D^2 -PaS中的本地调度表生成机制在其自身生成静态调度表 S 并且按照其传输包。当在 $t' = r_{0,m}$ 时刻节奏任务 τ_0 检测到干涉发生时，会发送一个通知信息给所有负责处理干涉的节点，我们令 $V_j \in \mathbf{V}_{rhy}$ 表示这些节点。当收到干涉通知信息后， \mathbf{V}_{rhy} 内的每个节点确定系统节奏模式的时间区间，并且生成一个用来处理干涉的动态调度表 \tilde{S} 。从 $r_{0,m+1}$ 时刻开始，即检测到干涉后的一个正常周期， \mathbf{V}_{rhy} 内的节点按照动态调度表 \tilde{S} 运行，而所有其它节点依旧按照原始的静态调度表 S 发送和接收包。因此，不同于集中式方法依赖于一个控制节点生成动态调度表，之后广播给每个节

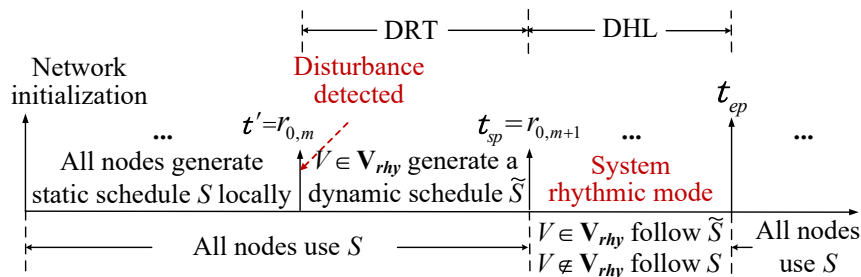


图 6.4 FD-PaS框架的运行模型

Fig. 6.4 Overview of the execution model of FD-PaS.

点，FD-PaS能够大大的所见响应干涉的时间。为了简化和清晰描述，我们称从 t' 到系统节奏模式开始时刻之间的时间区间称为干涉响应时间，称系统节奏模式区间为干涉处理时间(见图6.4)。

为了保证FD-PaS能够正确运行，我们需要解决以下一些问题。首先，当干涉发生时，只有检测到干涉的传感器节点直到哪个任务将要进入到它的节奏状态，而其他负责处理干涉的节点对此并不知情。其次，如果 \mathbf{V}_{rhy} 中的节点按照动态调度表 \tilde{S} 运行，而所有其它节点依旧按照原始的静态调度表 S 发送和接收包，可能会发生包传输冲突，而这会导致节奏包错失截止期。第三，为了正确处理干涉，需要为 \mathbf{V}_{rhy} 中的节点设计有效的算法来生成一段动态调度表，并且最小化丢掉的周期包数量。我们在接下来的章节详细接收FD-PaS如何解决以上问题。

6.3 传播干涉信息

在集中式方法中，由于动态调度表需要部署在每个节点上，因此实时无线传感网络中的所有节点都必须直到干涉的信息。然而这样一种方法限制了在大规模网络中的应用，并且如在上节例子中看到的，常常会违反**问题1**中的限制 (i)，不能很快的响应干涉。为了克服这一缺陷，我们提出将出现的干涉信息仅传播给网络中一部分需要负责处理干涉的节点，表示为 \mathbf{V}_{rhy} ，以此最小化干涉响应时间。这一策略的实现需要回答以下三个问题：(1) 网络中的哪些节点属于 \mathbf{V}_{rhy} ? (2) 如何将干涉信息传播给 \mathbf{V}_{rhy} 中的节点? (3) 每个 \mathbf{V}_{rhy} 中的节点是否有足够的时间在系统进入节奏模式前完成动态调度表的计算? 接下来我们逐一回答这些问题。

考虑问题 (1) 和问题 (2)，当一个干涉出现时，节奏任务 τ_0 要进入它的节奏状态并且按照节奏周期 \vec{P}_0 和截止期 \vec{D}_0 执行，此时需要一个更新的动态调度表来适应 τ_0 所增加的工作量。为了保证节奏任务每个包的每一跳 $\chi_{0,k}(h)$ 的成功传输， $\chi_{0,k}(h)$ 的发送和接收节点都必须按照同一个调度表运行。因此，所有节奏任务路由路径上的节点都需要直到干涉信息，这样才能生成一致的动态调度表，所以它们都应当包含在 \mathbf{V}_{rhy} 中。例如，对于图6.3中的实时无线传感网络例子，当 τ_0 是节奏任务时， $\mathbf{V}_{rhy} = \{V_2, V_c, V_5\}$ 。当在 $r_{0,m}$ 时刻检测到干涉时，干涉信息可以被打包在当前包 $\chi_{0,m}$ 然后发送给 \mathbf{V}_{rhy} 中的节点。按照这种方式广播干涉信息可以保证所有 \mathbf{V}_{rhy} 中的节点都可以在检测到干涉后的一个正常周期(即 P_0)内收到干涉信息，因为静态调度表可以保证每个包都能在它的截止期前传送给路由路径上的每个节点以满足截止期约束。

现在考虑问题 (3)，正如**问题1**中限制 (i) 所要求的，在 $r_{0,m}$ 时刻检测到干涉

后，系统需要在下个正常释放时刻 $r_{0,m+1}$ 开始进入到节奏模式处理干涉。这就要求 (i) 干涉信息必须在 τ_0 于 $r_{0,m+1}$ 时刻进入节奏状态前抵达所有 \mathbf{V}_{rhy} 中的节点，(ii) \mathbf{V}_{rhy} 中的每个节点都能够在它自身开始接收或发送动态调度表里的第一个传输前完成动态调度表的计算。上一段介绍的干涉信息广播机制可以确保 (i) 能够满足。关于条件 (ii)，我们上一章的工作已经证明了一个本地空闲时间片 (10ms) 足够一个设备节点 (如TI CC2538 SoC) 来完成调度表的计算。那么接下来的定理说明了对于每个设备节点，条件 (ii) 里所说的时间区间内总是存在至少一个本地空闲时间片。

定理 6.1: 如果一个实时无线传感网络在给定的静态调度表下是可调度的，任一 \mathbf{V}_{rhy} 内的节点 V_j ($V_j \neq V_c$)，在它收到干涉信息的 t_1 时刻 ($t_1 \geq r_{0,m}$) 和 τ_0 在 $r_{0,m+1}$ 时刻进入节奏状态后 V_j 所涉及的第一个包传输时刻 t_2 之间的区域内，至少存在一个本地空闲时间片。

证明: 我们首先引用我们上一章的一个引理。

引理 6.2: 如果系统是可调度的，即每个包的所有传输都能够在截止期前到达执行器节点，对于任意一个设备节点 V_j 和路径经过 V_j 的任务 τ_i ，在 V_j 的调度表中任意三个连续的 τ_i 的传输中间一定存在至少一个本地空闲时间片。

因为在我们的系统模型中，传感器节点和执行器节点通过控制节点 V_c 相连，每个任务的路由路径至少包含两跳，即每个任务至少有两跳的传输。假设传输 $\chi_{0,m}(h)$ 发生在 t_1 时刻，并且它是 V_j 收到干涉信息的传输 (如果 V_j 是传感器节点，它在 $r_{0,m}$ 时刻检测到干涉)，则在 $\chi_{0,m}(h)$ 和 $\chi_{0,m+1}(h)$ (在 t_2 时刻 V_j 所涉及的第一个动态调度表内的传输) 之间至少还存在一个 τ_0 的传输，那么根据引理6.2， V_j 在 $\chi_{0,m}(h)$ 和 $\chi_{0,m+1}(h)$ 之间 (即 t_1 和 t_2 之间) 至少存在一个本地空闲时间片。定理得证。 \square

基于定理6.1，我们提出的干涉信息广播策略能够保证任何干涉都能够在一个节奏任务的正常周期内得到响应，**问题1**中的限制 (i) 满足。

6.4 避免传输冲突

根据上一小节我们提出的干涉信息广播策略，只有节奏任务路径上的节点包含在 \mathbf{V}_{rhy} 中，所有 \mathbf{V}_{rhy} 中的节点在本地生成动态调度表并在系统进入节奏模式后按照其运行，而所有其它的节点依旧按照静态调度部运行。在这种运行模式下，除非干涉信息广播给了网络中所有的节点，那么一定存在节点间调度表不一

致的情况，这很可能会导致传输冲突的发生。为了确保干涉能够正确地被处理，在FD-PaS框架中，即使一个节奏传输在某个时间片内跟其它周期传输发生了冲突，我们也一定要保证节奏包总是会成功传输。

在传统实时无线传感网络，如WirelessHART和6TiSCH中，普遍采用基于TDMA的数据链路层协议，以提供同步的和免冲突的信道通信。此外，大部分这些协议在每个传输开始前执行CCA（Clear Channel Assessment）操作来避免干涉。但是CCA并不能为包赋优先级，当在一个时间片内出现多个包传输时，并不能保证高优先级的传输（如节奏包传输）能够得到信道使用权。为了解决这一问题，我们针对IEEE 802.15.4e标准提出一种多优先级MAC协议MP-MAC，支持为实时无线传感网络中的包传输赋优先级。类似的，由于本部分工作不是主要由作者本人完成，我们不在此讨论具体的细节，感兴趣的读者可以参考文章[161]。

6.5 本地动态调度表生成

MP-MAC机制能够保证一旦节点在本地生成了动态调度表， \mathbf{V}_{rhy} 中的节点能够按照该调度表处理干涉，并且不会跟网络中其它节点发生传输冲突。因为 \mathbf{V}_{rhy} 中的节点收到的是相同的干涉信息，因此在这些节点中生成的动态调度表是一致的。动态调度表的生成必须保证（1）所有节奏包满足截止期约束，（2）丢掉的周期包数量最小，（3）系统能够在节奏模式结束后重新用回原始的静态调度表并且所有包都能够满足它们的正常截止期。在本节中，我们介绍FD-PaS如何在 \mathbf{V}_{rhy} 中节点生成动态调度表。

6.5.1 问题定义

在FD-PaS框架中，网络开始运行后按照一个静态调度表运行，并且如果没有干涉发生的话，所有任务都能够满足截止期。静态调度表是在每个节点中利用上一章中介绍的本地调度表生成机制计算的。我们用 $S = \{(t, i, h)\}$ 表示静态调度表，其中 t 表示时间片ID， i 是任务ID以及 h 是当前跳数。对于任意给定的时间片 t ，如果 t 分配给了任务 τ_i 的第 h 跳传输，则 $S[t] = (i, h)$ 。否则 $S[t] = (-1, -1)$ 表示一个空闲时间片。

如图6.1所示，当在 $r_{0,m}$ 时刻检测到一个干涉发生时， τ_0 请求从它的下一个释放时间开始进入节奏状态，即 $t_{n \rightarrow r} = r_{0,m+1}$ 。之后系统进入到节奏模式，并且伴随着 τ_0 所带来的工作量增加。因此，系统在回到正常模式使用静态调度表 S 之前，需要一个动态调度表 \tilde{S} 。 \tilde{S} 从 $t_{n \rightarrow r}$ 时刻开始，结束于一个精心选择的系统节奏模式结

束时刻。为了保证能够快速的处理干涉，我们还定义一个用户指定参数 t_{ep}^u ，规定了最大允许的干涉处理时间。虽然我们可以利用 $S[t_{n \rightarrow r}, t_{ep}]$ 中的空闲时间片来调度增加的节奏任务工作量，但这些空闲时间片并不能保证满足所有节奏包的截止期。这种情况下，一些周期传输必须被丢掉。因为任意节点 $V_j \notin \mathbf{V}_{rhy}$ 一直使用静态调度表 S 来传输包周期包，周期包传输在动态调度表中不能被调整。因此，如果 S 中任意周期传输 $\chi_{i,k}(h)$ 在动态调度表 \tilde{S} 中被一个节奏包传输替换，那么我们则需要丢掉整个 $\chi_{i,k}$ 。之后，我们需要确定哪些周期包传输应当被节奏包传输所替换，然后生成动态调度表 \tilde{S} 满足 (i) 所有节奏包满足截止期约束以及 (ii) 被丢掉的周期包数量最小。

形式化的，为了满足**问题1**中的限制 (ii) (iii) 和 (iv)，我们需要解决以下两个子问题。

问题1.1-结束时刻选择：给定任务集 \mathcal{T} ， $t_{n \rightarrow r}$ ， t_{ep}^u 以及静态调度表 S ，该子问题需要确定结束时刻 t_{ep} 并且满足以下两条限制。

$$\text{限制 6.1 } f_{0,m+R} \leq t_{ep} \leq t_{ep}^u$$

这里， $f_{0,m+R}$ 是 τ_0 在节奏状态下释放的最后一个包的结束时刻， $f_{0,m+R} \leq t_{ep}$ 保证了当前节奏事件能够在系统转换回正常状态之前被正确处理。

限制 6.2 系统能够从 t_{ep} 之后转换回正常状态继续使用静态调度表，并且所有 t_{ep} 之后释放的所有包都能够满足正常截止期。

问题1.2-动态调度表生成：该子问题生成动态调度表 $\tilde{S}[t_{n \rightarrow r}, t_{ep}]$ ，使得被丢掉的周期包数量（表示为 $\rho[t_{n \rightarrow r}, t_{ep}]$ ）最小，且满足以下两条限制。

限制 6.3 所有节奏包都能够满足截止期。

限制 6.4 在动态调度表 $\tilde{S}[t_{n \rightarrow r}, t_{ep}]$ 中，任意周期包传输时间片 $S[t] = (i, h) (1 \leq i \leq n)$ 只能被替换成一个节奏包传输时间片 $S[t] = (0, h)$ ，或者保持不变。

接下来我们讨论FD-PaS如何解决以上问题。

6.5.2 结束时刻选择

选择正确的动态调度表结束时刻非常重要，因为它不仅影响干涉处理时间的长短，而且影响丢掉的周期包数量。OLS中也有一个类似的概念称之为转换时刻。因为OLS和FD-PaS都需要系统在 t_{ep} 之后重新使用静态调度表，为了选择FD-PaS中的结束时刻，我们借用OLS的一些思想，包括对齐 τ_0 的实际释放时刻和正常释放时刻，以及通过只考虑 τ_0 的实际释放时间来减少备选结束时刻数量。

FD-PaS和OLS在选择结束时刻是有两个主要的不同点。首先，为了满足限

制6.2, 我们需要确定哪些包必须在系统从 t_{ep} 开始重新使用静态调度表之前完成。因为OLS必须满足一个用户制定的在动态调度表 \tilde{S} 中最多可以更新的时间片数量限制, 需要建立一个包含所有需要在 $\tilde{S}[t_{n \rightarrow r}, t_{ep})$ 内被调度的包传输的集合。然而, FD-PaS由于它的分布式特性而无此限制, 因此只要建立一个活跃包集合包含所有需要被调度的包集合。其次, 根据限制6.4, 在动态调度表中, 周期包传输必须不能被调整并且只能被节奏包传输所替代。因此, 对于活跃包集合, 我们只需要考虑 $\tilde{S}[t_{n \rightarrow r}, t_{ep})$ 内需要被调度的节奏包。这些区别需要在节奏时刻选择过程中进行一定的修改, 细节如下。

令 $\Psi(t_{ep})$ 表示活跃包集合包含所有需要在 $[t_{n \rightarrow r}, t_{ep})$ 内被调度的节奏包。显然, 任意释放时间和截止期都在 $[t_{n \rightarrow r}, t_{ep})$ 内的节奏包都应当被包含在 $\Psi(t_{ep})$ 内, 问题是如何处理那些释放时间在 t_{ep} 之前而截止期在 t_{ep} 之后的节奏包。如图5.2所示, 令 χ_{0,q^*} 表示这样一个节奏包。为了保证系统可以从 t_{ep} 开始重新使用静态调度表, τ_0 在 t_{ep} 周的实际释放时间必须跟未进入节奏状态情况下的正常释放时间对齐。跟OLS相同, 我们通过将 r_{0,q^*+1} 调整到最近的 τ_0 的释放时间(表示为 r_{0,p^*}), 缩短 r_{0,q^*} 和 r_{0,q^*+1} 之间的距离。而更具挑战性的部分是调整 χ_{0,q^*} 的截止期和执行时间, 因为包传输的数量可能会依据哪一跳发生在 t_{ep} 而改变。我们通过考虑以下两种情况, 根据 t_{ep} 的位置调整创建的包 χ_{0,q^*} 的截止期和执行时间。

情况1: 如果 $t_{ep} < r_{0,p^*}$, 则 d_{0,q^*} 被调整到 t_{ep} 。假设在静态调度表中 τ_0 在 t_{ep} 之后的第一个包传输发生在 t_{h_0} 时刻, 并且 $S[t_{h_0}]$ 是用来传输 τ_0 的第 h 跳, 即 $S[t_{h_0}] = (0, h_0)$ 。如果 $t_{h_0} \geq r_{0,p^*}$, 这表示 $S[t_{h_0}]$ 是用来调度 χ_{0,p^*} 的第1跳, 即 $S[t_{h_0}] = (0, 1)$ 。之后 χ_{0,p^*} 的执行时间设置为 H_0 。如果 $t_{h_0} < r_{0,p^*}$, 则相应地执行时间被设置为 $h_0 - 1$ 。

情况2: 如果 $t_{ep} \geq r_{0,p^*}$, 假设在静态调度表中 χ_{0,p^*} 的第一跳在 t_1 时刻传输, 即 $S[t_1] = (0, 1)(r_{0,p^*} \leq t_1 < d_{0,p^*})$ 。截止期 d_{0,q^*} 被调整为 $\min(t_{ep}, t_1)$ 来保证 χ_{0,q^*} 的截止期小于等于 χ_{0,q^*+1} 的第一个传输。同样, χ_{0,q^*} 的执行时间设置为 H_0 。

给定 t_{ep}^u , $[f_{0,m+R}, t_{ep}^u]$ 内的任意一个时刻都可以被选择为结束时刻 t_{ep} 。然而, 为了避免很耗时地检查每一个时刻, 我们只需要考虑 τ_0 在 $[f_{0,m+R}, t_{ep}^u]$ 内的实际释放时间作为备选结束时刻, 表示为 t_{ep}^c 。也就是,

$$\{t_{ep}^c\} = \{r_{0,k}, \forall r_{0,k} \in [f_{0,m+R}, t_{ep}^u]\} \quad (6.1)$$

那么, 动态调度表生成的**问题1.2**可以被进一步定义如下。

问题1.2: 给定备选结束时刻 t_{ep}^c , 活跃包集合 $\Psi(t_{ep}^c)$ 和静态调度表 $S[t_{n \rightarrow r}, t_{ep}^c)$, 确定动态调度部 $\tilde{S}[t_{n \rightarrow r}, t_{ep}^c)$ 使得被丢掉的周期包数量最小, 即

$$\min |\rho[t_{n \rightarrow r}, t_{ep}^c]| \quad (6.2)$$

以及满足限制6.3和限制6.4。

6.5.3 动态调度表生成

确定丢包集合（即解决(6.2)中定义的问题）并不是一个简单的问题，下面的引理说明了该问题的特性。

引理 6.3: 给定结束时刻 t_{ep} ，一个活跃包集合 $\Psi(t_{ep})$ 包含所有 τ_0 需要被调度的节奏包以及一个静态调度表 $S[t_{n \rightarrow r}, t_{ep}]$ ，确定含丢包数最少并且满足限制6.3和限制6.4的丢包集合 $\rho[t_{n \rightarrow r}, t_{ep}]$ 这一问题NP难的。

证明: 我们通过把集合覆盖问题^[162]规约到丢包问题的一个特殊情况来证明该引理。

集合覆盖问题定义如下：给定一个包含 n 个元素的集合 $X = \{x_1, x_2, \dots, x_n\}$ 以及一个集合 $C = \{C_1, C_2, \dots, C_m\}$ 包含 m 个非空子集合 X 其中 $\cup_{i=1}^m C_i = X$ 。集合覆盖问题就是要确定一个集合 $C_s \subseteq C$ ，它的合集等于 X 并且 $|C_s|$ 最小。

给定一个集合覆盖问题，我们可以在多项式时间内建立一个丢包问题的特殊情况如下：

(1) 假设在利用完静态调度表 $S[t_{n \rightarrow r}, t_{ep}]$ 中 τ_0 原本的传输时间片以及空闲时间片来传输 $\Psi(t_{ep})$ 中的节奏包后， τ_0 依然有 n 个包需要被调度，表示为 $\{x_1, x_2, \dots, x_n\}$ ，并且每个包 x_i 只需要一个额外的时间片来传输。

(2) 在静态调度表 $S[t_{n \rightarrow r}, t_{ep}]$ 中，有 m 个周期包，表示为 $\{C_1, C_2, \dots, C_m\}$ 。对于每个包 C_j ，如果存在一个 C_j 的传输落入到节奏包 x_i 的时间窗口内，即 $[r_{x_i}, d_{x_i})$ ，则 $x_i \in C_j$ 。

因此，我们可以确定一个能够满足所有节奏包截止期的最少丢包集合 $\rho[t_{n \rightarrow r}, t_{ep}]$ ，当且仅当合集等于 X 的最小子集合 C_s 能够被确定。引理得证。□

在确定了丢包集合之后，我们就可以通过利用静态调度表中的空闲时间片和丢掉周期包的传输时间片来调度节奏任务的所有包传输，这样就能在线性时间内得到动态调度表。因此根据引理6.3我们可以直接推出以下定理，并且我们省略掉相应的证明。

定理 6.4: 动态调度表生成，即问题1.2，是NP难的。

接下来我们重点解决丢包问题，我们首先通过给每个周期包分配一个二元变量表示该包是否应当被丢弃来建立一个整数线性规划ILP算法来得到该问题的最优解。我们引入以下符号。

$\Lambda_j = [\lambda_j^1, \lambda_j^2, \dots, \lambda_j^n] (1 \leq j \leq m)$ 表示周期包 χ_j 的传输向量，其中每个 λ_j^i 是静态调度表中 χ_j 的可以在动态调度表中被节奏包 $\chi_{0,i}$ 所替换的传输数量。具体的，如果 $S[t] = (j, h)$ 并且 $r_{0,i} \leq t < d_{0,i}$ ，则包 χ_j 的传输 $\chi_j(h)$ 可以被 $\chi_{0,i}$ 替换。

w_j 代表周期包 χ_j 是否被丢掉。如果 χ_j 被丢掉，则 $w_j = 1$ 。否则， $w_j = 0$ 。

$A = [a_1, a_2, \dots, a_n]$ 表示可用时间片向量，其中每个 a_i 代表静态调度表中所有空闲时间片和原本的节奏传输时间片可以在动态调度表中被节奏包 $\chi_{0,i}$ 使用的数量。

为了丢掉最少数量的周期包以保证所有节奏包的截止期得到满足，我们建立以下的ILP目标函数：

$$\min \sum_{\chi_j \in \Phi} w_j \quad (6.3)$$

因为节奏包传输具有最高优先级，每个节奏包 $\chi_{0,i}$ 的截止期能够满足的条件就是在动态调度表中，至少为 $\chi_{0,i}$ 在其窗口内预留 H_0 个时间片。另外，静态调度表中的空闲时间片和原本的节奏传输时间片也可以用来满足 $\chi_{0,i}$ 的传输需求。因此，目标函数6.3受以下条件限制：

$$\forall \chi_{0,i} \in \Psi, \quad \sum_{\chi_j \in \Phi} \lambda_j^i \cdot w_j \geq H_{0,i} - a_i \quad (6.4)$$

考虑到丢包算法需要被部署在每个资源受限的设备节点上运行，ILP算法的计算复杂度可能会非常高，因为节奏包集合规模 ($|\Psi|$) 以及周期包集合规模 ($|\Phi|$) 可能都会很大，尤其对于大规模的网络来说。因此我们提出一种在时间和空间上都更有效率的贪婪启发式算法来解决丢包问题。

启发式算法的主要思想是丢掉在动态调度表中能够贡献给节奏包传输时间片数量最多的周期包。令 $\Psi = \{\chi_{0,i} | 1 \leq i \leq n\}$ 表示活跃包集合包含所有需要被调度的节奏包。 $\chi_{0,i}$ 的执行时间用 $H_{0,i}$ 表示。令 $\Phi = \{\chi_j | 1 \leq j \leq m\}$ 表示周期包集合，其中每个包 χ_j 至少在静态调度表 $S[t_{n \rightarrow r}, t_{ep}]$ 中有一个传输时间片。算法9描述了我们提出的启发式算法如何丢弃周期包的过程。

给定一个静态调度表 $S[t_{n \rightarrow r}, t_{ep}]$ ，一个周期包集合 $\Phi = \{\chi_j | 1 \leq j \leq m\}$ ，其中每个包 χ_j 建立并维护一个传输向量 $\Lambda_j = [\lambda_j^1, \lambda_j^2, \dots, \lambda_j^n]$ (第1-2行)。考虑 $S[t_{n \rightarrow r}, t_{ep}]$ 中原本的空闲和节奏包传输时间片，算法为所有 $\Psi(t_{ep})$ 中的节奏包创建一个需求向量 $[v_1, v_2, \dots, v_n]$ 其中 v_i 描述了节奏包 $\chi_{0,i}$ 还额外需要的时间片数量(在公式(6.4)中 $v_i = H_{0,i} - a_i$) (第3行)。如果需求向量中的所有元素值都等于0，表示静态调度表中原本的空闲和节奏传输时间片已经足够来调度所有 $\Psi(t_{ep})$ 中的节奏包，不再需要额

算法 9 Greedy Heuristic for Dropping Packets

Input: $\Psi(t_{ep}), S[t_{n \rightarrow r}, t_{ep}]$

Output: $\rho[t_{n \rightarrow r}, t_{ep}]$

- 1: $\Phi \leftarrow$ periodic packet set $\{\chi_j | 1 \leq j \leq m\}$;
 - 2: $\Lambda_j \leftarrow$ transmission vector $[\lambda_j^1, \lambda_j^2, \dots, \lambda_j^n]$ for each periodic packet χ_j ;
 - 3: Construct $\Psi(t_{ep})$'s demand vector $[v_1, v_2, \dots, v_n]$ considering the idle slots and rhythmic transmission slots in $S[t_{n \rightarrow r}, t_{ep}]$;
 - 4: **if** each v_i equals 0 **then**
 - 5: **return** \emptyset ;
 - 6: **end if**
 - 7: **while** true **do**
 - 8: Add the periodic packet χ_{max} with the maximum $\sum_{i=1}^n \lambda_j^i$ in Φ to $\rho[t_{n \rightarrow r}, t_{ep}]$;
 - 9: $\Phi \leftarrow \Phi \setminus \{\chi_{max}\}$;
 - 10: **for** $i \in \{1, \dots, n\}$ **do**
 - 11: $v_i \leftarrow \max(0, v_i - \lambda_{max}^i)$;
 - 12: **end for**
 - 13: **if** each v_i equals 0 **then**
 - 14: **return** $\rho[t_{n \rightarrow r}, t_{ep}]$;
 - 15: **end if**
 - 16: Update Λ_j for each $\chi_j \in \Phi$;
 - 17: **end while**
-

外丢弃周期包，此时算法返回一个空集(第4-6行)。否则，算法按如下贪婪方式丢弃周期包。在每次循环中， Φ 中具有最大 $\sum_{i=1}^n \lambda_j^i$ 值的周期包 χ_{max} 被添加到丢包集合中，并且从 Φ 中移除(第8-9行)。接下来算法更新 $\Psi(t_{ep})$ 的需求向量，为每个 v_i 减去 λ_{max}^i (第10-12行)。如果在丢掉 χ_{max} 后所有节奏包都可调度了，即每个 v_i 都等于0，那么算法就返回丢包集合 $\rho[t_{n \rightarrow r}, t_{ep}]$ (第13-15行)。否则，算法根据节奏包的状态相应的更新每个周期包 χ_j 的传输向量(第16行)。具体的，如果节奏包 $\chi_{0,i}$ 已经可调度了，即 $v_i = 0$ ，那么 λ_j^i 设置为0。如果 $0 < v_i < \lambda_j^i$ ，也就是丢弃 χ_j 对调度节奏包 $\chi_{0,i}$ 来说是富余的，我们则令 $\lambda_j^i = v_i$ 。算法重复这一过程直到所有节奏包都可调度，然后返回一个丢包集合。

最终，算法10总结了 \mathbf{V}_{rhy} 中的节点如何生成动态调度表以解决丢包问题1.2。

6.6 性能评估

在本节中，我们从模拟实验和测试平台两方面给出重要的实验结果来评

算法 10 Dynamic Schedule Generation

Input: $t_{ep}^u, S[t_{n \rightarrow r}, t_{ep}^u]$

Output: $\tilde{S}[t_{n \rightarrow r}, t_{ep}]$

- 1: Construct the end point candidate set $\{t_{ep}^c\}$ according to (6.1);
 - 2: **for** ($\forall t_{ep}^c \in \{t_{ep}^c\}$) **do**
 - 3: Construct $\Psi(t_{ep}^c)$; // active packet set
 - 4: Generate a dropped packet set $\rho[t_{n \rightarrow r}, t_{ep}^c]$ based on $\Psi(t_{ep}^c)$ using ILP solution or the proposed heuristic;
 - 5: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\rho[t_{n \rightarrow r}, t_{ep}^c]\}$;
 - 6: **end for**
 - 7: Select the $\rho[t_{n \rightarrow r}, t_{ep}^*]$ with the minimum number of dropped packet in \mathcal{S} ;
 - 8: Generate dynamic schedule $\tilde{S}[t_{n \rightarrow r}, t_{ep}^*]$ based on $\rho[t_{n \rightarrow r}, t_{ep}^*]$ and $S[t_{n \rightarrow r}, t_{ep}^*]$;
 - 9: **return** $\tilde{S}[t_{n \rightarrow r}, t_{ep}^*]$;
-

估FD-PaS框架在实时无线传感网络中的性能。

6.6.1 模拟实验

6.6.1.1 实验设置

在模拟实验中，我们将FD-PaS同OLS和D²-PaS两种能够处理实时无线传感网络中干涉的方法进行比较。对于FD-PaS，我们用算法9中提出的启发式算法来生成丢包集合。在实验中我们用以下两个指标进行性能对比。

系统成功率 (Success Ratio, SR): 系统成功率定义为可行的任务集数量与随机生成任务集总数之比。一个任务集是可行的当且仅当干涉出现时，系统可以在指定的干涉响应时间内开始处理干涉。

丢包率 (Drop Rate, DR): 丢包率定义为丢掉的周期包数量与系统在节奏模式下释放的包的总数之比。

为了比较的公平，我们随机生成任务集，每个随机任务集的生成都是根据一个目标系统利用率 U^* ，然后从一个空集合 \mathcal{T} 开始，向其中逐个添加随机生成的任务。每个随机任务 τ_i 的生成是由以下参数控制。(i) τ_i 的跳数 H_i 从 $\{2, 3, \dots, 16\}$ 中随机取值；(ii) 正常状态下的周期 P_i 从 $\{H_i, \dots, 500\}$ 内随机取值；(iii) 正常状态下的截止期 D_i 等于正常周期 P_i 。当 \mathcal{T} 中的所有任务都生成之后，我们随机地从中选择一个任务作为节奏任务 τ_0 ，并且假设干涉是在 τ_0 的第 k 个包的释放时刻检测到的， k 的值从 $\{1, \dots, 20\}$ 中随机取出。 τ_0 的节奏周期向量 \vec{P}_0 ($\vec{D}_0 = \vec{P}_0$)由以下参数控制生成：(i) \vec{P}_0 中包含的元素数量 R ；(ii) 初始节奏周期比率 $\gamma = P_{0,1}/P_0$ 。为

为了更好地控制节奏任务的工作量，我们固定 γ 值为0.2，然后调整 R 的值，即可以是 $\{4, 6, \dots, 16\}$ 内的任意整数。确定了 γ 和 R 的值后，每个节奏周期 $P_{0,k}$ 的值可以通过 $P_{0,k} (1 \leq k \leq R) = \lfloor P_0 \times (\gamma + (k-1) \times \frac{1-\gamma}{R}) \rfloor$ 计算得出。

实验需要的其它参数总结如下：（1）最大允许的干涉响应时间 α ，其值等于节奏任务正常周期 P_0 的整数倍；（2）结束时刻比率因子 β ，该值决定了结束时刻的上限 t_{ep}^u ，即 $t_{ep}^u = t_{r \rightarrow n} + (\beta - 1) \times P_0$ 。显然，一个更大的 β 值会产生更好的性能，即更少的丢包数，但是也会导致更长的干涉处理时间DHL。为了尽可能的让 β 值更小但又不会导致性能下降，我们设置 $\beta = 4$ ，即干涉必须在节奏任务回到正常周期后的3个周期内被处理完。其它一些OLS和D²-PaS用到的参数，例如广播包所能装载的信息量大小等，我们沿用上一章模拟实验中相同的设置。

6.6.1.2 实验结果

在实验的第一部分，我们比较OLS, FD-PaS和D²-PaS的系统成功率，我们固定随机生成任务集的正常利用率 $U^* = 0.5$ ，然后改变最大允许的干涉响应时间 α 从 P_0 到 $6P_0$ ，以 P_0 为单位递增（见图6.5）。其它任务集正常利用率下的实验结果相似，因此在此省略。在实验中，每个数据点是基于10000个随机生成任务集对应的结果取平均。如图6.5所示，D²-PaS和OLS的系统成功率完全相同，因为它们具有共同的机制，即依赖于网络中的控制节点将干涉信息广播给所有其他节点。对于D²-PaS和OLS，只有在 $\alpha = 6P_0$ ，即最大允许干涉响应时间设置为节奏任务的6个正常周期长度时，系统才是永远可行的。然而，在大多数实际设置中，实时无线传感网络需要在一个正常周期内就提供对干涉的快速响应，即 $\alpha = P_0$ 。在这种情况下，D²-PaS和OLS只有25%的系统成功率，而FD-PaS总是能够获得100%的系统成功率，因为在FD-PaS下，按照问题1中限制（i）的要求，系统总是能够从节奏任务的下一个释放时刻开始处理干涉。

在实验的第二部分，我们通过改变随机生成任务集的系统利用率和节奏任务的节奏周期数量 R ，来比较OLS, FD-PaS和D²-PaS的平均丢包率。图6.6所示为实

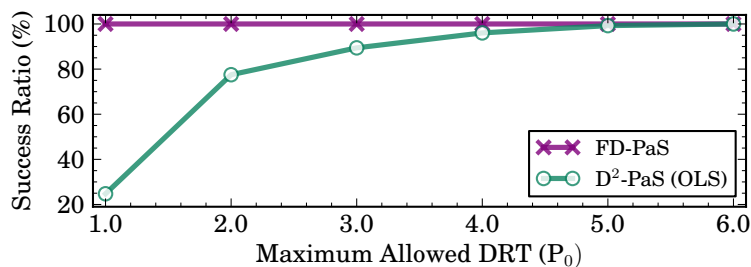


图 6.5 系统利用率 $U^* = 0.5$ 下系统成功率对比

Fig. 6.5 Comparison of SR with a nominal utilization $U^* = 0.5$.

验结果，每个数据点1,000此实验取平均值的结果。从图6.6中可以看出，FD-PaS相比OLS具有明显更低的平均丢包率(平均低53%，最好情况下低82%)。OLS的平均丢包率高的主要原因是OLS具有非常高的广播负载需求。另一方面，相比较D²-PaS，FD-PaS平均需要多丢掉12%的周期包，因为FD-PaS是在本地节点做丢包决策，信息相对局限。但是，由于FD-PaS干涉响应时间方面巨大的性能提升（当最大允许干涉响应时间设置为 P_0 时提升了75%），在平均丢包率上的性能损失是可以接受的。

从图6.6中可以观察到的另一个不太寻常的现象是，FD-PaS的平均丢包率当 R 从4增加到10时首先下降，然后当 R 从10继续增加到16时又上升。从直觉上，平均丢包率应当随着系统利用率 U^* 和节奏周期数量 R 增加而单调递增（正如OLS和D²-PaS表现出来的一样）。通过大量的实验（细节我们这里不再给出），我们发现FD-PaS的平均丢包率很大程度上取决于时间片利用率这一参数，即所有丢弃的周期包传输中被节奏包传输利用到的数量所占的比例。当 R 从一个较小的值(如4)开始增加时，时间片利用率也随之增加（例如，当 R 值较小，即使节奏包只需要额外占用周期包一个时间片时，这一整个周期包都需要被丢掉，而它剩余的时间片实际上也被浪费掉了）。也就是说，当节奏任务的工作量增大时，虽然我们需要丢掉更多的周期包来满足它的截止期约束，丢掉的周期包数量增长的速度要低于系统节奏模式中释放的包数量的增长。这解释了为什么FD-PaS的平均丢包率当 R 从4增加到10时会下降，另一方面，我们发现当 R 继续增加，时间片利用率也开始随之下降，此时丢包数量的增长速度要快于系统节奏模式中释放的包数量的增长。这导致了FD-PaS的平均丢包率当 R 从10继续增加到16时，也开始上升。

6.6.2 测试平台实验

除了模拟实验，我们还在真实的实时无线传感网络测试平台上实现了提出的FD-PaS框架，以此验证它的有效性。我们的测试平台是基于OpenWSN协议栈和TI CC2538 SoC。MP-MAC的实现是通过改进OpenWSN的数据链路层功能，而动态调度表生成算法是实现在了应用层中。由于本部分工作并非主要由作者本人完成，我们不作太多细节的介绍，感兴趣的读者可以参考文章[161]。此处我们只展示在实验平台上针对FD-PaS框架的实验验证结果。

我们通过将FD-PaS部署在一个包含7个节点的多跳网络（见图6.2）中来验证FD-PaS的正确性和有效性。网络中运行了3个任务， $\tau_0 = \{\{V_0, V_1, V_c, V_3, V_4\}, 15\}$ ， $\tau_1 = \{\{V_2, V_c, V_3, V_4\}, 20\}$ 以及 $\tau_3 = \{\{V_1, V_c, V_5\}, 20\}$ ，其中第一个元素代表路由信息，第二个元素表示任务的周期（截止期）。我们假设 τ_0 是节奏任务并且 $\vec{P}_0(\vec{D}_0) =$

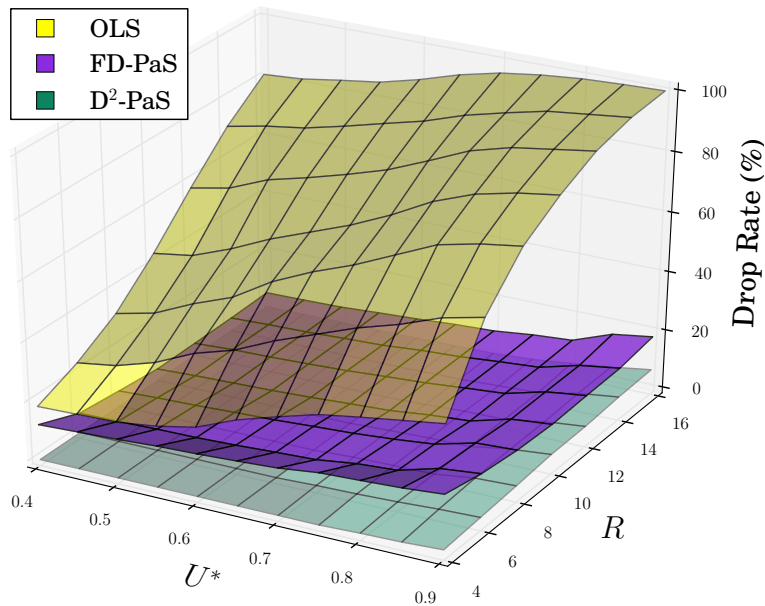


图 6.6 平均丢包率对比

Fig. 6.6 Comparison of the average DR.

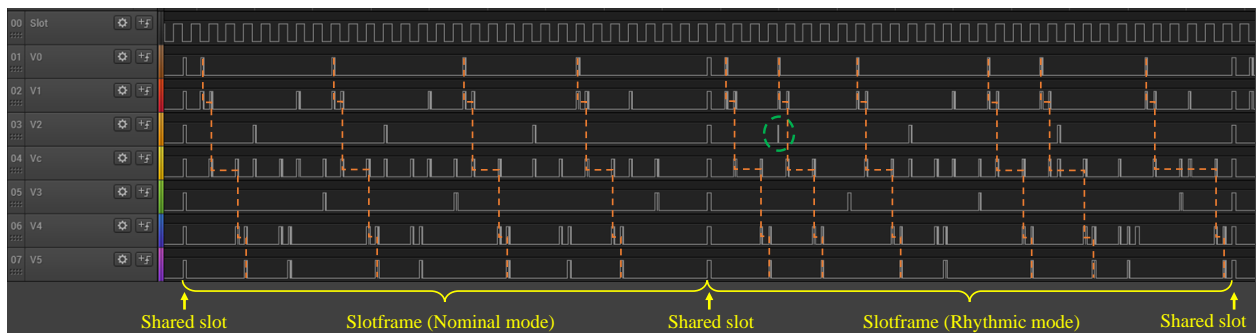


图 6.7 在实验平台平台上的时间片信息和频段状态

Fig. 6.7 Slot information and radio activities in the test case captured by Logic Analyzer.

[8,9,10,11,12]。我们利用一个逻辑分析器（Logic Analyzer）从每个设备捕捉频段活动状态。图6.7所示为得到的结果。

在图6.7中，在最上面的一个波形内的一个下落或上升的边表示一个新的时间片的开始，下面7个波形则是网络中相应的7个节点的频段活动状态（发送或接受一个包），对应的节点编号在图左边给出。为了简化实验，我们只将每个时间片周期（slotframe）内的第一个时间片用作共享时间片来做网络通讯管理。时间片周期的边界可以在图中轻易地辨别，因为所有的设备都会在共享时间片内处于频段活跃状态。图6.7包含两个时间片周期，其中系统在第一个时间片周期内运行在正常模式下，而在第二个时间片周期开始转换到节奏模式下运行。在实验中，我们通过在网络运行过程中触发一个按钮来仿真一个干涉事件的发生。当收到

这个事件后，从 τ_0 的下一个释放时间，系统转换到节奏模式运行。在节奏模式下（第2个时间片周期）， τ_0 需要处理6个包，而它在第1个时间片周期内只需要处理4个。 τ_0 包的传输在图6.7中用橘色虚线标出。为了调度节奏模式下节奏任务增加的工作量，第2个时间片周期内 τ_1 释放的第1个包被丢掉了。然而，由于周期任务 τ_1 的传感器节点 V_2 并没有收到干涉信息，并且按照静态调度表发送包， V_2 依然会在 V_0 根据生成的动态调度表发送它的第二个节奏包的同一个时间片内发送 τ_1 的周期包（图6.7中绿色圆圈）。但是由于我们设计的MP-MAC协议， τ_0 的节奏包传输成功被发出，而 τ_1 的周期包传输发送失败。该例子完全验证了本章提出的FD-PaS框架的正确性和有效性。

6.7 小结

在本章中，我们提出了完全分布式包调度框架FD-PaS来处理实时无线传感网络中出现的干涉。不同于集中式调度方法在控制节点生成动态调度表，并且通过广播包发送给网络中的节点的方法，FD-PaS在本地节点做在线丢包决策，并且不依赖与网络中任一控制节点。这样一种完全分布式的方法不仅大大提高了FD-PaS的可扩展性，并且能够提供有保证的对干涉的快速响应。我们提出的FD-PaS框架包含多优先级数据链路层设计MP-MAC和一个部署在设备节点上的动态调度表生成算法。大量的实验验证了FD-PaS的正确性和有效性。

第7章 结 论

信息物理系统 (Cyber-Physical System, CPS) 作为一种连接物理世界与信息世界的新兴控制系统在2006年美国科学院首次提出后近年来得到了快速的发展, 并且吸引了工业界和学术界的广泛关注。由于在信息物理系统的许多应用场景 (如飞行控制系统、工业控制系统等) 中都需要保证任务能够满足一定的时间约束, 即任务必须在规定的时间 (即截止期) 内正确地完成所有功能的执行, 实时调度技术在对信息物理系统的研究中扮演着非常重要的作用。虽然在过去的几十年里, 面向传统单核及多核处理器的实时调度技术已经发展的比较成熟, 但是由于信息物理系统在实际应用场景中所呈现出的多样性和特殊性, 这些传统实时调度技术并不能直接应用到每种不同的CPS系统中。基于这一背景, 本文根据信息物理系统具体的不同应用场景, 分别研究了混合关键性系统、严格周期任务系统以及实时无线传感网络中的任务和包的调度问题, 提出了一系列调度技术来提高处理器的平均实时性能以及实时无线网络中动态应对突发事件的响应能力。

7.1 本文主要贡献与结论

本文主要有4个贡献点:

(1) 针对混合关键性系统, 我们设计了一种新的应用于EDF-VD调度策略的可调度性分析方法。不同于之前的分析方法分别单独分析每个不同关键性级别下的系统可调度性, 我们的方法创新性地将系统的运行时行为作整合性研究, 以此获取更加精确的分析结果。大量的实验结果表明, 我们的分析方法能够显著的提升EDF-VD算法调度下的混合关键性系统的可调度性, 尤其是对拥有大于两个关键性级别的多级关键性系统, 提升的效果更加明显。这种显著的可调度性的提升是建立在更高的分析复杂度的基础之上, 针对这一问题, 我们通过结合本文所提出的分析方法以及之前的研究工作, 进一步提出了一种平衡分析精确度和分析复杂度的启发式算法, 以此, 为系统设计者提供更灵活的调度算法选择空间。

(2) 针对严格周期任务系统, 本文首创性地提出一种有效的方法来为每个严格周期任务分配开始执行时间, 使得系统中所有的任务可调度且不会发生冲突。本文首先给出一个充分的可调度性判定条件来检查系统是否可调度。为了提高分析的有效性, 我们通过研究任务周期之间的关系, 提出一种合理的任务选择策略, 以提高为每个任务成功分配开始执行时间的可能性。通过大量的随机实

验,验证了本文方法的可调度性显著高于现有其他相关工作,并且接近于最优的精确分析策略,并且本文算法的运行效率显著高于最优策略。因此,本文提出的任务开始执行时间分配策略在精确性和有效性方面取得了优异的平衡。

(3) 针对实时无线传感网络,本文提出一种新颖的分布式动态包调度框架 D^2 -PaS。 D^2 -PaS支持在线地处理网络中随机出现的紧急事件(本文定义为干涉),保证当干涉出现时,所有重要的网络包都能够在截止期内抵达目的节点,而同时使得其他一些非紧急的包被丢掉的数量最小。作为一种分布式的调度策略, D^2 -PaS让每个网络中的节点在本地分布式地生成调度表,当干涉出现时,这种方式能够大大地减少网络中的控制节点(如网关)向每个节点广播的动态调度表相关的信息。作为一种动态策略, D^2 -PaS在网络中的控制节点上部署一个轻量级的丢包算法,能够在线动态地响应网络中出现的干涉。我们将设计的 D^2 -PaS框架实现在一个真实的多跳实时无线网络测试平台上来验证其可用性。大量的模拟实验进一步确认了 D^2 -PaS的有效性。

(4) 同样针对实时无线传感网络,本文提出一种完全分布式的包调度框架FD-PaS。FD-PaS能够保证网络中随机出现的每个干涉都能够得到快速的响应,同时在确保关键包能够准时传输的同时,其他非紧急的网络包被丢掉的数量最小。为了使得FD-PaS能够被应用于较大的实时无线传感网络中,本文提出相应的算法以及数据链路层设计来确保网络中每个节点在没有集中式控制节点的帮助下,依然都能够在线地对出现的干涉作出响应和处理。通过大量的随机模拟和真实测试平台实验,我们验证了本文提出的FD-PaS调度框架的正确性和有效性。

7.2 进一步的工作

随着处理器芯片硬件方面的飞速发展和性能的快速提升,实际信息物理系统所面临的调度问题也会更加复杂。基于已经获得的研究成果,在未来的工作中,我们将在以下几个方面进一步深入研究:

(1) 针对混合关键性系统,为了能够获得分析精度和分析复杂度的平衡,我们设计了一种思路来结合本文所提出的分析方法以及EY算法,但本文并未对此进行深层次探究。下一步,我们会研究设计一种启发式算法来自动地决定如何在混合分析方法中决定对系统关键级别的分组策略,即对哪些连续的系统关键级别运行状态用本文所提的分析方法。此外,我们会研究如何将本文所提出的针对混合关键性系统的可调度性分析技术扩展到应用划分调度算法的多核系统中,而针对此问题最主要的挑战在于设计有效的划分策略使得对系统可调度性分析的复杂度

不会指数型增长。

(2) 针对严格周期任务系统，我们会更深层次的探索严格周期任务集的开始时间分配问题。例如，我们会研究如何更有效的为任务分配开始执行时间，而不是本章中所采用的直接选择最早出现的可行区间的方法。针对每个任务的执行时间都为1的特殊严格周期任务系统，我们也会尝试找到最优的开始时间分配策略。另外，我们也将研究如何将本章的方法扩展到多核处理器调度中，其中主要的挑战在于设计有效的任务划分策略。

(3) 针对实时无线传感网络，本文所设计的 D^2 -PaS框架在应对网络中出现的干涉时获得了较好的性能（非常低的丢包率）。但由于实际RTWN中的网络环境更加复杂和多变，下一步我们会研究如何将 D^2 -PaS扩展到更加一般化的网络模型中。首先，考虑包含多个信道的实时无线传感网络，此时研究的挑战主要在于如何在设计调度策略时处理多信道传输所可能引发的传输冲突，例如同一个节点不能在同一个时间片内接收或发送来自不同信道的多个网络包。此外，还会考虑网络中的链路不可靠的情况，如网络拓扑不稳定、发送包丢失等，这些不稳定的因素会对分析本就复杂的随机干涉事件产生严重的干扰。另外，我们还会在网络中引入更加一般化的干涉模型，在本文的研究中，我们假设网络中任意两个节点间都存在干涉，即任一时间片内最多只有一个传输可以发生。而在实际中，当网络规模不断增大，会存在距离较远的节点间能够同时接受和发送网络包的情况。这种复杂的干涉模型的引入会使得分析难度进一步上升。

(4) 类似的，在对本文设计的完全分布式框架FD-PaS的扩展研究中，我们依然会首先考虑将网络模型进行更加一般化的扩展，使得FD-PaS的应用场景更加广泛。除此之外，在对FD-PaS进行分析时，我们假设网络中不会并发的出现干涉，即在任意时刻最多只有一个干涉存在。但在实际中，多个不同类型的干涉可能会并发的出现和存在，我们会研究如何扩展FD-PaS框架使其能够应对这种状况的出现。此处的主要挑战在于如何在分布式调度框架下，保证对多个并发存在的干涉处理之间不会发生冲突。

参考文献

1. Rajkumar R, Lee I, Sha L, et al. Cyber-physical systems: the next computing revolution [J], 2010, 14(6): 731–736.
2. Park P, Tomlin C. Investigating Communication Infrastructure of Next Generation Air Traffic Management [A], Proceedings of IEEE/ACM Third International Conference on Cyber-Physical Systems [C], 2012, 35–44.
3. Hatcliff J, King A, Lee I, et al. Rationale and Architecture Principles for Medical Application Platforms [A], Proceedings of IEEE/ACM Third International Conference on Cyber-Physical Systems [C], 2012, 3–12.
4. Facchinetti T, Vedova M L D. Real-Time Modeling for Direct Load Control in Cyber-Physical Power Systems [J], IEEE Transactions on Industrial Informatics, 2011, 7(4): 689–698.
5. Qin Z, Li Q, Chuah M C. Unidentifiable Attacks in Electric Power Systems [A], Proceedings of IEEE/ACM Third International Conference on Cyber-Physical Systems [C], 2012, 193–202.
6. Leonardi F, Pinto A, Carloni L P. Synthesis of Distributed Execution Platforms for Cyber-Physical Systems with Applications to High-Performance Buildings [A], Proceedings of IEEE/ACM International Conference on Cyber-Physical Systems [C], 2011, 215–224.
7. Schlick J. Cyber-physical systems in factory automation - Towards the 4th industrial revolution [A], Proceedings of IEEE International Workshop on Factory Communication Systems [C], 2012, 55–55.
8. Jayachandran P, Abdelzaher T. Transforming Distributed Acyclic Systems into Equivalent Uniprocessors under Preemptive and Non-Preemptive Scheduling [A], Proceedings of Euromicro Conference on Real-Time Systems [C], 2008, 233–242.
9. Bai J, Eyisi E P, Qiu F, et al. Optimal Cross-Layer Design of Sampling Rate Adaptation and Network Scheduling for Wireless Networked Control Systems [A], Proceedings of IEEE/ACM Third International Conference on Cyber-Physical Systems [C], 2012, 107–116.
10. Kim H, Kang G S. Scheduling of Battery Charge, Discharge, and Rest [A], Proceedings of IEEE Real-Time Systems Symposium [C], 2009, 13–22.

11. Wikipedia. DO-178B — Wikipedia, The Free Encyclopedia [EB/OL], 2017, <https://en.wikipedia.org/w/index.php?title=DO-178B&oldid=813670260>, [Online; accessed 7-December-2017].
12. Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment [J], *Journal of the ACM (JACM)*, 1973, 20(1): 46–61.
13. Baruah S K, Mok A K, Rosier L E. Preemptively scheduling hard-real-time sporadic tasks on one processor [A], *Proceedings of Real-Time Systems Symposium*, 1990. *Proceedings.*, 11th [C]. IEEE, 1990, 182–190.
14. Dollimore J, Kindberg T. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)* [M], Addison-Wesley Longman Publishing Co., Inc., 2005.
15. Spitzer C R. *The avionics handbook* [M], CRC Press, 2001.
16. Watkins C B, Walter R. Transitioning from federated avionics architectures to Integrated Modular Avionics [A], *Proceedings of Ieee/aiaa Digital Avionics Systems Conference* [C], 2007, 2.A.1–1–2.A.1–10.
17. Hei X, Du X, Lin S, et al. PIPAC: Patient infusion pattern based access control scheme for wireless insulin pump system [A], *Proceedings of INFOCOM* [C], 2013.
18. Karbhari V M, Ansari F. *Structural Health Monitoring of Civil Infrastructure Systems* [J], *CRC Press*, 2009.
19. Gatsis K, Ribeiro A, Pappas G. *Optimal power management in wireless control systems* [A], *Proceedings of ACC* [C], 2013.
20. Han S, Lam K Y, Chen D, et al. *Online Mode Switch Algorithms for Maintaining Data Freshness in Dynamic Cyber-Physical Systems* [J], *IEEE Transactions on Knowledge and Data Engineering*, 2016, 28(3): 756–769.
21. Han S, Chen D, Xiong M, et al. *Schedulability Analysis of Deferrable Scheduling Algorithms for Maintaining Real-Time Data Freshness* [J], *IEEE Transactions on Computers*, 2014, 63(4): 979–994.
22. Xiong M, Han S, Lam K Y, et al. *Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis, and Results* [J], *IEEE Transactions on Computers*, 2011, 57(7): 952–964.
23. Lu C, Saifullah A, Li B, et al. *Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems* [J], *Proceedings of the IEEE*, 2016, 104(5): 1013–1024.

24. Available:<http://newsinfo.nd.edu/news/17248-nsf-funds-cyber-physical-systems-project/>. Accessed April 4, 2006.
25. Sprinkle J, Sastry S. *CHESS: Building A Cyber-Physical Agenda on Solid Foundations [J]*, 2006.
26. Vestal S. *Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance [A]*, *Proceedings of IEEE International Real-Time Systems Symposium [C]*, 2007, 239–243.
27. Dorin F, Richard P, Richard M, et al. *Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities [J]*, *Real-Time Systems*, 2010, 46(3): 305–331.
28. Baruah S, Vestal S. *Schedulability analysis of sporadic tasks with multiple criticality specifications [A]*, *Proceedings of Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on [C]. IEEE*, 2008, 147–155.
29. Baruah S, Bonifaci V, D'Angelo G, et al. *Scheduling real-time mixed-criticality jobs [J]*, *IEEE Transactions on Computers*, 2012, 61(8): 1140–1152.
30. Baruah S, Li H, Stougie L. *Towards the design of certifiable mixed-criticality systems [A]*, *Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE [C]. IEEE*, 2010, 13–22.
31. Audsley N C, Dd Y. *Optimal Priority Assignment And Feasibility Of Static Priority Tasks With Arbitrary Start Times [J]*, 1991.
32. Guan N, Ekberg P, Stigge M, et al. *Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems [A]*, *Proceedings of Real-Time Systems Symposium [C]*, 2012, 13–23.
33. Gu C, Guan N, Deng Q, et al. *Improving OCBP-based scheduling for mixed-criticality sporadic task systems [A]*, *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications [C]*, 2013, 247–256.
34. Baruah S K, Bonifaci V, D'Angelo G, et al. *Mixed-Criticality Scheduling of Sporadic Task Systems [A]*, *Proceedings of European Conference on Algorithms [C]*, 2011, 555–566.
35. Baruah S, Bonifaci V, D'Angelo G, et al. *The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems [A]*, *Proceedings of Real-Time Systems [C]*, 2012, 145–154.
36. Ekberg P, Yi W. *Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks [A]*, *Proceedings of ECRTS [C]*, 2012, 135–144.

37. Ekberg P, Yi W. *Bounding and shaping the demand of generalized mixed-criticality sporadic task systems* [J], *Real-Time Systems*, 2014, 50(1): 48–86.
38. Easwaran A. *Demand-Based Scheduling of Mixed-Criticality Sporadic Tasks on One Processor* [A], *Proceedings of IEEE Real-Time Systems Symposium* [C], 2013, 78–87.
39. Baruah S, Chattopadhyay B, Li H, et al. *Mixed-criticality scheduling on multiprocessors* [J], *Real-Time Systems*, 2014, 50(1): 142–177.
40. Li H, Baruah S. *Global Mixed-Criticality Scheduling on Multiprocessors* [A], *Proceedings of Real-Time Systems* [C], 2012, 166–175.
41. Gu C, Guan N, Deng Q, et al. *Partitioned mixed-criticality scheduling on multiprocessor platforms* [A], *Proceedings of Conference on Design, Automation and Test in Europe* [C], 2013, 292.
42. Burns A, Davis R. *Mixed Criticality Systems - A Review* [J], 2013.
43. Jeffay K, Stanat D F, Martel C U. *On non-preemptive scheduling of period and sporadic tasks* [A], *Proceedings of Real-Time Systems Symposium, 1991. Proceedings., Twelfth* [C], 1991, 129–139.
44. Baruah S K, Chakraborty S. *Schedulability analysis of non-preemptive recurring real-time tasks* [A], *Proceedings of Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. International* [C], 2006, 8 pp.
45. Buttazzo G, Cervin A. *Comparative Assessment and Evaluation of Jitter Control Methods* [A], 2007.
46. George L. *Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling* [J], HAL - INRIA, 1996.
47. Korst J, Aarts E H L, Lenstra J K, et al. *Periodic multiprocessor scheduling* [A], *Proceedings of International Conference on Parallel Architectures and Languages Europe* [C], 1991, 166–178.
48. Korst J H M. *Periodic multiprocessor scheduling* [J], 1992.
49. Kermia O, Sorel Y. *Schedulability Analysis for Non-Preemptive Tasks under Strict Periodicity Constraints* [A], *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* [C], 2008, 25–32.
50. Marouf M, Sorel Y. *Schedulability conditions for non-preemptive hard real-time tasks with strict period* [A], 2011.

51. Marouf M, Sorel Y. Scheduling non-preemptive hard real-time tasks with strict periods [A], *Proceedings of Emerging Technologies and Factory Automation [C]*, 2011, 1–8.
52. Eisenbrand F, Hahnle N, Niemeier M, et al. Scheduling Periodic Tasks in a Hard Real-Time Environment [A], *Proceedings of International Colloquium Conference on Automata, Languages and Programming [C]*, 2010, 299–311.
53. Available:<https://www.isa.org/isa100/>. ISA100.
54. Available:<https://en.hartcomm.org/>. WirelessHART.
55. Available:<http://www.wina.org>. Wireless Industrial Networking Alliance.
56. Available:<http://www.zigbee.org>. ZigBee Alliance.
57. Davis R I, Burns A, Bril R J, et al. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised [J], *Real-Time Systems*, 2007, 35(3): 239–272.
58. Saifullah A, Gunatilaka D, Tiwari P, et al. Schedulability Analysis under Graph Routing in WirelessHART Networks [A], *Proceedings of Real-Time Systems Symposium [C]*, 2015, 165–174.
59. Available:<http://www.industry.usa.siemens.com/automation/us/en/process-instrumentationand-analytics/process-instrumentation/brochures/Documents/PIBR-0A940-1014-one-stop-shop.pdf>. Siemens, Process instrumentation and weighing technology..
60. Available:http://en.hartcomm.org/hcp/tech/applications/applications_success_mitsubishi_chemical.html. HART Communication Foundation, Mitsubishi chemical connects HART technology to control room savings up to /20C30,000 a day..
61. Sha M, Gunatilaka D, Wu C, et al. Implementation and experimentation of industrial wireless sensor-actuator network protocols [A], *Proceedings of European Conference on Wireless Sensor Networks [C]*. Springer, 2015, 234–241.
62. Available:<http://www.tinyos.net/>. TinyOS.
63. Wu C, Gunatilaka D, Saifullah A, et al. Maximizing network lifetime of wireless sensor-actuator networks under graph routing [J], 2015.
64. Wu C, Gunatilaka D, Sha M, et al. Conflict-aware real-time routing for industrial wireless sensor-actuator networks [J], 2015.
65. Li H, Shenoy P, Ramamritham K. Scheduling Messages with Deadlines in Multi-Hop Real-Time Sensor Networks [A], *Proceedings of IEEE Real Time and Embedded Technology and Applications Symposium [C]*, 2005, 415–425.

66. Wang X, Wang X, Fu X, et al. *Flow-Based Real-Time Communication in Multi-Channel Wireless Sensor Networks [J]*, 2009, 5432: 33–52.
67. Kanodia V, Li C, Sabharwal A, et al. *Distributed multi-hop scheduling and medium access with delay and throughput constraints [A]*, *Proceedings of International Conference on Mobile Computing and NETWORKING [C]*, 2001, 200–209.
68. Lu C, Blum B M, Abdelzaher T F, et al. *RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks [A]*, *Proceedings of Eighth IEEE Real-Time and Embedded Technology and Applications Symposium [C]*, 2002, 55.
69. Karenos K, Kalogeraki V, Krishnamurthy S V. *A Rate Control Framework for Supporting Multiple Classes of Traffic in Sensor Networks [A]*, *Proceedings of Real-Time Systems Symposium, 2005. RTSS 2005. IEEE International [C]*, 2005, 11 pp.–297.
70. Karenos K, Kalogeraki V. *Real-Time Traffic Management in Sensor Networks [A]*, *Proceedings of Real-Time Systems Symposium, 2006. RTSS '06. IEEE International [C]*, 2006, 422–434.
71. He T, Blum B M, Cao Q, et al. *Robust and timely communication over highly dynamic sensor networks [J]*, *Real-Time Systems*, 2007, 37(3): 261–289.
72. Ahn G S, Campbell A T, Veres A, et al. *SWAN: service differentiation in stateless wireless ad hoc networks [A]*, *Proceedings of Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE [C]*, 2002, 457–466 vol.2.
73. He T, Stankovic J A, Lu C, et al. *SPEED: a stateless protocol for real-time communication in sensor networks [A]*, *Proceedings of International Conference on Distributed Computing Systems [C]*, 2003, 46.
74. Felemban E, Lee C G, Ekici E. *MMSPEED: multipath Multi-SPEED protocol for QoS guarantee of reliability and Timeliness in wireless sensor networks [J]*, *IEEE Transactions on Mobile Computing*, 2006, 5(6): 738–754.
75. Bui B D, Pellizzoni R, Caccamo M, et al. *Soft Real-Time Chains for Multi-Hop Wireless Ad-Hoc Networks [A]*, *Proceedings of Real Time and Embedded Technology and Applications Symposium, 2007. RTAS '07. IEEE [C]*, 2007, 69–80.
76. Gu Y, He T, Lin M, et al. *Spatiotemporal Delay Control for Low-Duty-Cycle Sensor Networks [A]*, *Proceedings of Real-Time Systems Symposium, 2009, RTSS [C]*, 2009, 127–137.

77. Koubaa A, Alves M, Tovar E. *i-GAME: An Implicit GTS Allocation Mechanism in IEEE 802154 for Time-Sensitive Wireless Sensor Networks [A]*, *Proceedings of Euromicro Conference on Real-Time Systems [C]*, 2006, 183–192.
78. Caccamo M, Zhang L Y, Sha L, et al. *An Implicit Prioritized Access Protocol for Wireless Sensor Networks [J]*, *IEEE Rtss*, 2002. 39.
79. Liu K, Abu-Ghazaleh N, Kang K D. *JiTS: just-in-time scheduling for real-time sensor data dissemination [A]*, *Proceedings of IEEE International Conference on Pervasive Computing and Communications [C]*, 2006, 5 pp.–46.
80. Li B, Wang D, Wang F, et al. *High Quality Sensor Placement for SHM Systems: Refocusing on Application Demands [A]*, *Proceedings of Conference on Information Communications [C]*, 2010, 650–658.
81. Mangharam R, Rowe A, Rajkumar R, et al. *Voice over Sensor Networks [A]*, *Proceedings of IEEE International Real-Time Systems Symposium [C]*, 2006, 291–302.
82. Facchinetti T, Almeida L, Buttazzo G C, et al. *Real-time resource reservation protocol for wireless mobile ad hoc networks [A]*, *Proceedings of IEEE International Real-Time Systems Symposium [C]*, 2004, 382–391.
83. Stankovic J A, Abdelzaher T F, Lu C, et al. *Real-time communication and coordination in embedded sensor networks [J]*, *Proceedings of the IEEE*, 2003, 91(7): 1002–1022.
84. Korber H J, Wattar H, Scholl G. *Modular Wireless Real-Time Sensor/Actuator Network for Factory Automation Applications [J]*, *IEEE Transactions on Industrial Informatics*, 2007, 3(2): 111–119.
85. Xiangheng Liu A J G. *Cross-layer design of distributed control over wireless network [M]*, 2005: 111–136.
86. Liu X, Goldsmith A. *Wireless network design for distributed control [A]*, *Proceedings of Decision and Control, 2004. Cdc. IEEE Conference on [C]*, 2004, 2823–2829 Vol.3.
87. Xiao L, Johansson M, Hindi H, et al. *Joint Optimization of Wireless Communication and Networked Control Systems [J]*, *Lecture Notes in Computer Science*, 2005, 3355(1): 248–272.
88. Carley T W, Ba M A, Barua R, et al. *Contention-Free Periodic Message Scheduler Medium Access Control in Wireless Sensor / Actuator Networks [A]*, *Proceedings of IEEE International Real-Time Systems Symposium [C]*, 2003, 298.

89. Saifullah A, Sankar S, Liu J, et al. *CapNet: A Real-Time Wireless Management Network for Data Center Power Capping* [A], *Proceedings of Real-Time Systems Symposium* [C], 2014, 334–345.
90. Alur R, D’Innocenzo A, Johansson K H, et al. *Modeling and Analysis of Multi-hop Control Networks* [A], *Proceedings of Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS* [C], 2009, 223–232.
91. Saifullah A, Xu Y, Lu C, et al. *Real-Time Scheduling for WirelessHART Networks* [A], *Proceedings of Real-Time Systems Symposium* [C], 2010, 150–159.
92. Saifullah A, Xu Y, Lu C, et al. *End-to-End Delay Analysis for Fixed Priority Scheduling in WirelessHART Networks* [A], *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium* [C], 2011, 13–22.
93. Schoolfield G C. *End-to-End Communication Delay Analysis in Industrial Wireless Networks* [J], *Computers IEEE Transactions on*, 2013, 64(5): 1361–1374.
94. Saifullah A, Xu Y, Lu C, et al. *Priority Assignment for Real-Time Flows in WirelessHART Networks* [J], 2011, 6794(29): 35–44.
95. Soldati P, Zhang H, Johansson M. *Deadline-constrained transmission scheduling and data evacuation in WirelessHART networks* [A], *Proceedings of Control Conference* [C], 2015, 4320–4325.
96. Wu C, Sha M, Gunatilaka D, et al. *Analysis of EDF scheduling for Wireless Sensor-Actuator Networks* [A], *Proceedings of Quality of Service* [C], 2014, 31–40.
97. Zhang H, Osterlind F, Soldati P, et al. *Rapid Convergecast on Commodity Hardware: Performance Limits and Optimal Policies* [A], *Proceedings of Sensor Mesh and Ad Hoc Communications and Networks* [C], 2010, 1–9.
98. Crenshaw T L, Hoke S, Tirumala A, et al. *Robust implicit edf: A wireless mac protocol for collaborative real-time systems* [J], *ACM Transactions on Embedded Computing Systems (TECS)*, 2007, 6(4): 28.
99. Shen W, Zhang T, Gidlund M, et al. *SAS-TDMA: a source aware scheduling algorithm for real-time communication in industrial wireless sensor networks* [J], *Wireless Networks*, 2013, 19(6): 1155–1170.
100. Ferrari F, Zimmerling M, Mottola L, et al. *Low-power wireless bus* [A], *Proceedings of SenSys* [C], 2012.

101. Sha M, Dor R, Hackmann G, et al. *Self-Adapting MAC Layer for Wireless Sensor Networks [A], Proceedings of RTSS [C], 2013.*
102. Chipara O, Wu C, Lu C, et al. *Interference-Aware Real-Time Flow Scheduling for Wireless Sensor Networks [A], Proceedings of ECRTS [C], 2011.*
103. Zimmerling M, Mottola L, Kumar P, et al. *Adaptive real-time communication for wireless cyber-physical systems [J], ACM Transactions on Cyber-Physical Systems, 2017, 1(2): 8.*
104. Li B, Nie L, Wu C, et al. *Incorporating emergency alarms in reliable wireless process control [A], Proceedings of ICCPS [C], 2015.*
105. Hong S, Hu X S, Gong T, et al. *On-Line Data Link Layer Scheduling in Wireless Networked Control Systems [A], Proceedings of ECRTS [C], 2015.*
106. Rajhans A, Cheng S W, Schmerl B R, et al. *An Architectural Approach to the Design and Analysis of Cyber-Physical Systems [J], 2009, 21: 14–38.*
107. Chang C, Wawrzynek J, Brodersen R W. *BEE2: A High-End Reconfigurable Computing System [J], Design and Test of Computers IEEE, 2005, 22(2): 114–125.*
108. Henzinger T A, Jhala R, Majumdar R, et al. *Thread-Modular Abstraction Refinement [J], Lecture Notes in Computer Science, 2003, 2725: 262–274.*
109. Eker J, Janneck J W, Lee E A, et al. *Taming heterogeneity - the Ptolemy approach [J], Proceedings of the IEEE, 2003, 91(1): 127–144.*
110. Lea D. *Concurrent Programming in Java: Design Principles and Patterns [M], Addison-Wesley, 1997: 1–98.*
111. Schmidt D C, Rohnert H, Stal M, et al. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects [J], 2000, 2(6): 41.*
112. Culler D E, Dusseau A, Goldstein S C, et al. *Parallel Programming in Split-C [A], Proceedings of Supercomputing '93. Proceedings [C], 1993, 262–273.*
113. Blumofe R D, Joerg C F, Kuszmaul B C, et al. *Cilk: An efficient multithreaded runtime system [A], Proceedings of ACM Sigplan Symposium on Principles and Practice of Parallel Programming [C], 1996, 207–216.*
114. Edwards S A, Lee E A. *The case for the precision timed (PRET) machine [J], 2007. 264–265.*
115. Leung J Y T, Whitehead J. *On the complexity of fixed-priority scheduling of periodic, real-time tasks [J], Performance Evaluation, 1982, 2(4): 237–250.*

116. Mok A K. *Fundamental design problems of distributed systems for the hard-real-time environment [J]*, Massachusetts Institute of Technology, 1983.
117. Buttazzo G C. *Hard Real-Time Computing Systems [J]*, *Real-Time Systems Series*, 2011, 416: 31–38.
118. Kuo T W, Li C H. *A Fixed-Priority-Driven Open Environment for Real-Time Applications [A]*, *Proceedings of Real-Time Systems Symposium, 1999. Proceedings. the IEEE [C]*, 1999, 256–267.
119. Saewong S, Rajkumar R, Lehoczky J P, et al. *Analysis of Hierarchical Fixed-Priority Scheduling [A]*, *Proceedings of Real-Time Systems, 2002. Proceedings. Euromicro Conference on [C]*, 2002, 152–160.
120. Lipari G, Bini E. *Resource Partitioning among Real-Time Applications [A]*, *Proceedings of Real-Time Systems, 2003. Proceedings. Euromicro Conference on [C]*, 2003, 151–158.
121. Deng Z, Liu J W S. *Scheduling real-time applications in an open environment [A]*, *Proceedings of IEEE Real-Time Systems Symposium [C]*, 1997, 308.
122. Deng Z, Liu J W S, Sun J. *A scheme for scheduling hard real-time applications in open system environment [A]*, *Proceedings of Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on [C]*, 1997, 191–199.
123. Lipari G, Baruah S. *A Hierarchical Extension to the Constant Bandwidth Server Framework [J]*, *Rtas*, 2001. 26–35.
124. Lipari G, Baruah S K. *Efficient scheduling of real-time multi-task applications in dynamic systems [A]*, *Proceedings of IEEE Real Time Technology and Applications Symposium [C]*, 2000, 166.
125. Tindell K W, Burns A, Wellings A J. *An extendible approach for analyzing fixed priority hard real-time tasks [J]*, *Real-Time Systems*, 1994, 6(2): 133–151.
126. Stankovic J A, Ramamritham K, Spuri M. *Deadline scheduling for real-time systems: edf and related algorithms [J]*, Springer International, 1999, 460.
127. Serlin O, Serlin O. *Multiprogramming for hybrid computation [A]*, *Proceedings of November 14-16, 1967, Fall Joint Computer Conference [C]*, 1967, 1–13.
128. Serlin O. *Scheduling of time critical processes [A]*, *Proceedings of Managing Requirements Knowledge, International Workshop on [C]*, 1971, 925–932.
129. Lehoczky J, Sha L, Ding Y. *The rate monotonic scheduling algorithm: exact characterization and average case behavior [A]*, *Proceedings of Real Time Systems Symposium, 1989., Proceedings [C]*, 2002, 166–171.

130. Sha L, Goodenough J B. *Real-Time Scheduling Theory and Ada* [M], IEEE Computer Society Press, 1990: 53–62.
131. Sha L, Abdelzaher T, Cervin A, et al. *Real Time Scheduling Theory: A Historical Perspective* [J], *Real-Time Systems*, 2004, 28(2-3): 101–155.
132. Coffman E G. *Introduction to deterministic scheduling theory* [J].
133. Baruah S K, Rosier L E, Howell R R. *Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor* [J], *Real-Time Systems*, 1990, 2(4): 301–324.
134. Zheng Q. *Real-time fault-tolerant communication in computer networks* [J], *Umr*, 1993.
135. Joseph M, Pandya P. *Finding Response Times in a Real-Time System* [J], *Computer Journal*, 1986, 29(29): 390–395.
136. Audsley N, Burns A, Richardson M, et al. *Applying new scheduling theory to static priority pre-emptive scheduling* [J], *Software Engineering Journal*, 2002, 8(5): 284–292.
137. Elmenreich W, Paukovits C, Pitzek S. *Automatic generation of schedules for time-triggered embedded transducer networks* [M], 2005: 7 pp.–541.
138. Cucu L, Sorel Y. *Schedulability condition for systems with precedence and periodicity constraints without preemption* [J], 2003.
139. Behnam M, Isovich D. *Real-time control design for flexible scheduling using jitter margin* [J], 2007.
140. Inc A R. *Avionics Application Software Standard Interface* [J].
141. Sheikh A A, Brun O, Hladik P E, et al. *Strictly periodic scheduling in IMA-based architectures* [J], *Real-Time Systems*, 2012, 48(4): 359–386.
142. Kermia O. *Timing Analysis of TTEthernet Traffic* [J], *Journal of Circuits Systems and Computers*, 2015, 24(09): 1550140.
143. Kim J K, Kim B K. *Probabilistic Schedulability Analysis of Harmonic Multi-Task Systems with Dual-Modular Temporal Redundancy* [M], Kluwer Academic Publishers, 2004: 199–222.
144. Bernstein D J. *Fast multiplication and its applications, to appear in Buhler-Steinhagen Algorithmic number theory book* [J], URL: <http://cr.yp.to/papers.html#multapps>. ID 8758803e61822d485d54251b27b1a20d. Citations in this document, 3(4.1): 4–1.

145. Heninger N, Durumeric Z, Wustrow E, et al. *Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices [A]*, *Proceedings of USENIX Security Symposium [C]*, volume 8, 2012.
146. Han S, Zhu X, Chen D, et al. *Reliable and Real-time Communication in Industrial Wireless Mesh Networks [A]*, *Proceedings of RTAS [C]*, 2011.
147. Leng Q, Wei Y H, Han S, et al. *Improving Control Performance by Minimizing Jitter in RT-WiFi Networks [A]*, *Proceedings of RTSS [C]*, 2014.
148. Saifulah A, Lu C, Xu Y, et al. *Real-Time Scheduling for WirelessHART Networks [A]*, *Proceedings of RTSS [C]*, 2010.
149. Crenshaw T L, Hoke S, Tirumala A, et al. *Robust implicit EDF: A wireless MAC protocol for collaborative real-time systems [J]*, *ACM Trans. Embed. Comput. Syst.*, 2007.
150. Li L, Hu B, Lemmon M. *Resilient event triggered systems with limited communication [A]*, *Proceedings of CDC [C]*, 2012.
151. Buttazzo G, Bini E, Buttle D. *Rate-adaptive tasks: Model, analysis, and design issues [A]*, *Proceedings of DATE [C]*, 2014.
152. Kim J, Lakshmanan K, Rajkumar R. *Rhythmic Tasks: A New Task Model with Continually Varying Periods for Cyber-Physical Systems [A]*, *Proceedings of ICCPS [C]*, 2012.
153. Lawler E. *New and improved algorithms for scheduling a single machine to minimize the weighted number of late jobs [J]*, *Preprint, Computer Science Division, University of California*, 1989.
154. Moore J M. *An n job, one machine sequencing algorithm for minimizing the number of late jobs [J]*, *Management science*, 1968, 15(1): 102–109.
155. Baptiste P. *An $\mathcal{O}(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs [J]*, *Operations Research Letters*, 1999, 24(4): 175–180.
156. Graham R L, Lawler E L, Lenstra J K, et al. *Optimization and approximation in deterministic sequencing and scheduling: a survey [J]*, *Annals of discrete mathematics*, 1979, 5: 287–326.
157. Ho K I, Leung J Y, Wei W. *Scheduling imprecise computation tasks with 0/1-constraint [J]*, *Discrete applied mathematics*, 1997, 78(1): 117–132.
158. Watteyne T, Vilajosana X, Kerkez B, et al. *OpenWSN: a standards-based low-power wireless development environment [J]*, *Transactions on Emerging Telecommunications Technologies*, 2012, 23(5): 480–493.

159. Zhang T, Gong T, Gu C, et al. *Distributed Dynamic Packet Scheduling for Handling Disturbances in Real-Time Wireless Networks [A], Proceedings of RTAS [C], 2017.*
160. Chipara O, Lu C, Roman G C. *Real-Time Query Scheduling for Wireless Sensor Networks [A], Proceedings of RTSS [C], 2007.*
161. Zhang T, Gong T, Yun Z, et al. *FD-PaS: A Fully Distributed Packet Scheduling Framework for Handling Disturbances in Real-Time Wireless Networks [A], Proceedings of RTAS [C], 2018.*
162. Karp R M. *Reducibility Among Combinatorial Problems [J], Journal of Symbolic Logic, 2010, 40(4): 618–619.*

致 谢

四年半的博士生活即将结束，在这里我由衷的感谢这几年来给予我无私帮助和关怀的老师、同学、朋友和家人。

感谢我的导师邓庆绪教授。自从2011年硕士入学以来，我与邓老师的师生情谊已有7年。在工作中邓老师头脑敏锐，治学严谨，给予了我大量的指导，使我深刻体会到了作为科研工作者应具备的核心素养。在生活中，邓老师为人宽容、善良，是一位让人尊敬信赖的长者，更是给予了我无微不至的关怀，让我在科研生活的压力生活下感受到温暖。作为您的学生，在此向您说声感谢！

感谢我的另外一位导师胡晓波教授。在我圣母大学联合培养的一年半时间里，您作为一位国际上声誉卓著的学者，不但为我提供了世界级高水平的研究平台，更是在科研工作中给予了我大量细致的指导，让我深刻体会到了如何才能把学术工作做到极致。您的治学精神和人格魅力，将对我的一生产生深远的影响。作为您的学生，在此向您说声感谢！

感谢我的师兄关楠教授。是您把我从一个对科研懵懵懂懂的硕士研究生领入了学术的大门，感恩在我科研工作的启蒙阶段，能够遇到您这样一位严格要求、治学严谨的导师，谢谢您在工作中给予我的悉心教导，为我今后的学习打下了坚实的基础，在此向您说声谢谢！

感谢康涅狄格大学的韩松教授，谢谢您在合作时给予我的悉心指导，不论是思考问题的角度和深度，还是论文写作水平，我都有了非常大的提高。在此感谢您无私的付出！

感谢王义院长，王老师虽身居高位，但我们感受到的却是一位亲和、慈爱的长者。每次都能从您的话语中感受到您严谨的治学精神和坚毅的治学品格，这些都将让我受益终身！感谢我的师兄吕鸣松教授，您在学术上精益求精以及工作上认真负责的态度一直感染着我！感谢物联网工程研究所的陈喆老师、王璐老师、李传文老师、刘学老师、徐建有老师、汪力行老师、陈刚老师。这些可爱的老师让我们的研究所变成一个温暖的大家庭！

感谢我的师兄张轶、孔繁鑫、夏长清、刘玮、谷传才，以及金曦师姐、王妍师姐。感谢这些年来，在每一次关键时刻上你们给予我的经验和建议，感谢在我学业最困境的时期你们带给我的希望和帮助，感谢一起打拼的日子里你们给我的鼓励和关心！

感谢同实验室的师兄谭爱平、佟海滨、苏宪利，陈腾峰、秦俊平，在我博士论文的冲刺阶段，你们帮我分担了很多实验室的工作，特此感谢。感谢林宇晗、韩美玲、冯志伟、杨涛、孙磊、常爽爽、沈大伟、王侃侃等，你们的聪明才智给我的研究工作带来了取之不尽的灵感。更要感谢已经毕业的各届研究生师弟师妹，篇幅所限，恕我不能将各位的名字书写于此，谢谢你们从各方面所给予我的帮助和支持，与你们共同学习和生活的日子将是我人生最好的一段回忆。

感谢我的家人。谢谢你们对我生活最无微不至的关怀。在我遭遇困境的时候，您们比我还揪心；在我成功的时候，您们比我更开心。您们是我今生最大的财富，是我不断努力的动力源泉！

最后感谢我的妻子，在我博士在读的这些年你为我、为这个家付出了太多，这本薄薄的博士论文是献给你的礼物，感谢你的爱，让我对我们的未来充满期待，毫无畏惧！

攻博期间发表的论文

1. **Tianyu Zhang**, Tao Gong, Zelin Yun, Song, Han, Qingxu Deng and X, Sharon Hu. FD-PaS: A Fully Distributed Packet Scheduling for Handling Disturbances in Real-Time Wireless Networks. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'18), 2018. (CCF-Rank B, EI)
2. **Tianyu Zhang**, Tao Gong, Chuancai Gu, Huayi Ji, Song Han, Qingxu Deng and X, Sharon Hu. Distributed Dynamic Packet Scheduling for Handling Disturbances in Real-Time Wireless Networks. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17), 2017. (CCF-Rank B, EI)
3. **Tianyu Zhang**, Tao Gong, Song Han, Qingxu Deng and X, Sharon Hu. D²-PaS: A Distributed Dynamic Packet Scheduling Framework for Handling Disturbances in Real-Time Wireless Networks. IEEE Transactions on Mobile Computing (TMC), 2018. (CCF-Rank A, SCI, under review)
4. **Tianyu Zhang**, Nan Guan, Qingxu Deng and Wang Yi. Start time configuration for strictly periodic real-time task systems. Journal of Systems Architecture (JSA), Vol. 66 (2016): 61-68. (CCF-Rank B, SCI)
5. **Tianyu Zhang**, Nan Guan, Qingxu Deng and Wang Yi. On the Analysis of EDF-VD Scheduled Mixed-Criticality Real-Time Systems. Industrial Embedded Systems (SIES'14), 2014 9th IEEE International Symposium on, IEEE, 2014. (EI)
6. 张天宇, 关楠, 邓庆绪. 矿山安全管理系统中混合关键性实时调度研究. 小型微型计算机系统, Vol. 36, 2015.
7. 张天宇, 关楠, 邓庆绪. Xen 虚拟机Credit 调度算法的实时性能分析. 计算机科学, Vol. 42, 2015.
8. Tao Gong, Huayi Ji, **Tianyu Zhang**, Jianwei Zhou, Xiaolin Lu, X, Sharon Hu and Song Han. Demo Abstract: 6TiSCH in Full Bloom: from Dynamic Resource Management to Cloud-based Network Analytics. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'18) - Demo Session, April, 2018. (CCF-Rank B, EI, under review)
9. Tao Gong, Huayi Ji, Song Han, **Tianyu Zhang**, Chuancai Gu, X, Sharon Hu and Mark Nixon. Demo Abstract: A Cross-device Testing and Debugging System for Large-scale

- Real-Time Wireless Networks. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17) - Demo Session, April, 2017. (CCF-Rank B, EI)
10. Meiling Han, **Tianyu Zhang**, Yuhan Lin, Zhiwei Feng, Qingxu Deng. Global Fixed Priority Scheduling with Constructing Execution Dependency in Multiprocessor Real-Time Systems. *Journal of Circuits, Systems, and Computers (JCSC)*, 2018. (SCI, accepted)
 11. Tiantian Li, **Tianyu Zhang**, Jie Song and Ge Yu. TA-MCF: Thermal-Aware Fluid Scheduling for Mixed Criticality System. *Journal of Circuits, Systems, and Computers (JCSC)*, 2018. (SCI, accepted)
 12. Meiling Han, **Tianyu Zhang** and Qingxu Deng. Bounding Carry-in Interference for Synchronous Parallel Tasks under Global Fixed-Priority Scheduling. *Journal of Systems Architecture (JSA)*. (CCF-Rank B, SCI, under review)
 13. Haibin Tong, Qingxu Deng, **Tianyu Zhang** and Yuanguo Bi. A Low-cost Indoor Localization System Based on RSSI by Modifying Trilateration for Harsh Environments. *International Journal of Distributed Sensor Networks (SCI, under review)*
 14. 韩美灵, 邓庆绪, 张天宇, 冯智伟, 林宇晗. 多核处理器限制性可抢占G-EDF调度策略研究. *计算机学报*. (EI, under review)
 15. 韩美灵, 邓庆绪, 张天宇, 林宇晗. 基于G-EDF的DAG并行任务多核响应时间分析. *东北大学学报*. (EI, under review)

攻博期间参与的项目

1. 国家自然科学基金（项目编号：0973017），多核系统中实时调度策略的设计与分析技术的研究，2009.12 – 2012.12，主要研究人员.
2. 国家支撑计划（项目编号：2012BAK24B0104），基于物联网的地铁施工安全风险识别与可视化预警，2010.7 – 2014.7，主要研究人员.
3. 国家自然科学基金（项目编号：No. 61472072），基于虚拟化的多核实时系统设计与分析技术研究，2015.1-2018.12，主要研究人员.
4. 国家自然科学基金（项目编号：No. 61528202），面向安全关键性CPS的实时系统建模和能耗管理技术研究，2016.1 – 2017.12，主要研究人员.