Scheduling of Overload-Tolerant Computation and Multi-Mode Communication in Real-Time Systems

by

Akramul Azim

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Real-time tasks require sufficient resources to meet deadline constraints. A component should provision sufficient resources for its workloads consisting of tasks to meet their deadlines. Supply and demand bound functions can be used to analyze the schedulability of workloads. The demand-bound function determines the maximum required computational units for a given workload and the supply-bound function determines the minimum possible resources supplied to the workload. A component will experience an overload if it receives fewer resources than required. An overload will be transient if it occurs for a bounded amount of time. Most work concentrates on designing components that avoid overloads by over-provisioning resources even though some computational units such as control system components can tolerate transient overloads. Overload-tolerant components can utilize resources more efficiently if over-provisioning of resources can be avoided.

First, this dissertation presents the design of an efficient periodic resource model for scheduling computation of components that can tolerate transient overloads under the Earliest Deadline First (EDF) scheduling policy. We propose a periodic resource model for overload-tolerant components to address three problems: (1) characterize overloads and determine metrics of interest (i.e., delay), (2) derive a model to compute a periodic resource supply for a given workload and a worst-case tolerable delay, and (3) find a periodic resource supply for given control system specifications with a worst-case delay. The derived periodic resource supply can be used to derive an overload-tolerant component interface. Overload-tolerant real-time components can connect with each other in a distributed manner and thus require communication scheduling for reliable and guaranteed transmissions. Moreover, applications may require multi-mode communication for efficient data transmission.

Second, this dissertation discusses communication schedules for multi-mode distributed components. Since distributed multi-mode applications are prone to suffer from delays incurred during mode changes, good communication schedules have low average mode-change delays. A key problem in designing multi-mode communication in real-time systems is the generation of schedules to move away the complexity of schedule design from the developer. We propose a mechanism to generate multi-mode communication schedules using optimization constraints associated with timing requirements. We illustrate a workflow from specifications to the generation of communication schedules through a real-time video monitoring case-study. Experimental analysis for the case-study demonstrates that schedules generated using the proposed method reduce the average mode-change delay compared to a randomized algorithm and the well-known EDF scheduling policy.

Finally, this thesis discusses the synthesis of schedules for computation and communication to achieve not only performance but also separation of concerns for reducing complexity and increasing safety. To integrate overload-tolerant components using real-time communication, we derive specifications of component interfaces using the characterization of overloads and the proposed periodic resource model. The generation of communication schedules uses the specifications of interfaces which include timing requirements of possible transient overloads. A walk-through case-study explains the steps necessary to generate communication schedules using component interfaces. The interfaces provide safety through isolation of transient overload-tolerant components and the generated communication schedules provide high performance as a result of their low average mode-change delay.

Acknowledgements

I would like to express my sincere gratitude and profound indebtedness to my supervisor Prof. Sebastian Fischmeister for his constant guidance, insightful advice, helpful criticism, valuable suggestions, commendable support, endless patience, and numerous reviews towards the completion of this dissertation. I feel very proud and satisfied to have worked with him. Without his inspiring enthusiasm, encouragement, and financial support, it would not be possible to complete the thesis.

A very special thanks to Prof. Shreyas Sundaram and Prof. Rodolfo Pellizzoni for their passionate guidance and effective reviews to solve important research problems. I would also like to thank Prof. Hiren Patel, Prof. William Cowan, and Prof. Nathan Fisher for giving valuable feedback and suggestions on the dissertation. I would like to thank all the research partners who contributed financial assistance throughout the completion of this dissertation.

A huge thanks to the all past and present members of real-time embedded software group who have provided me valuable suggestions and comments. I would like to give a special thanks to Gonzalo Carvajal and Ahmed Alhammad for helping me in various situations towards the completion of this dissertation.

I am very much thankful to all my family members. My parents have made invaluable sacrifices throughout the period of my study and I feel proud to mention their sacrifices. I lost my father Md. Lokiot Ullah in 11.11.11 when I was in Canada during the course of my doctoral study. I would like to thank again Prof. Sebastian Fischmeister to give me immediate support and allow me to stay with my mother Bilkis Begum during those toughest days ever in my life. My father always inspired me to obtain higher studies and it may not have been possible without his support, inspiration, and sacrifice. My mother has always taken care me to give her best possible support. I am grateful to my mother for her support, inspiration, and sacrifice. I would like to thank my elder brother Anwarul Azim and cutest sister Sanjid Mahiba for their support and inspiration. Without them, I would not feel the importance of having brother and sister in someone's life. Finally, I am very much thankful to my wife Rafia Islam for his patience, support, sacrifice, and inspiration. She has given me mental support in my days when I needed her inspiration the most. I would like to thank all my well-wishers who have prayed for my success and helped me and my family directly or indirectly.

Dedication

This thesis is dedicated to my beloved departed father Md. Lokiot Ullah and my mother Bilkis Begum.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Real-time systems are usually time-sensitive and consist of time-critical distributed applications. If the computation is based on outdated data, incorrect result may appear. For example, a counter counts the number of occurrences of some specified external event and has to output the result. A timing failure on the input in observing the events can cause a functional failure on the output.

Generally, two types of real-time systems exist: soft real-time and hard-real time systems. Hard real-time systems must meet the deadline of every task, never failing to respond within a specified time. Conversely, soft real-time systems can tolerate some amount of late response to events. Due to the possibility of system failure in hard real-time systems, they must meet all timing constraints under fault and load conditions. Therefore, developers must carefully design and implement hard real-time systems to guarantee that deadlines are met.

1.1 Overload-Tolerant Computation

Real-time applications require sufficient resources, both computation and communication, to meet their timing requirements. Given a set of applications, various schedulability tests exist to evaluate whether each application receives the required resources. For instance, in hard real-time systems, a schedulability test, based on supply and demand bound functions [76, 12, 64, 59], show that insufficient resources are available when the supply-bound function (sbf) drops below the demand bound function (dbf) for a given time interval. However, this condition is too conservative for systems such as soft real-time systems or

firm real-time systems that can tolerate the occurrence of overloads (i.e., situations where the application requires more resources than are available) with a bounded duration. For example, if the system can tolerate dropping or delaying tasks, the schedulability test will accept a system whose supply bound function intermittently drops below the demand bound function.

Overloads can be either transient or permanent. A transient overload always has a bounded time span until its resolution and can occur because of excessive task execution times, or because of the simultaneous arrival of asynchronous events. A permanent overload always has an unbounded time span and will occur if the system is badly designed and unschedulable. A large class of systems can tolerate the occurrence of overloads with a specified duration. For example, while control systems require a certain measure of reliability in their feedback loops to function correctly, these systems are usually robust to short delays in their control updates (depending on the dynamics of the system) [89, 44]. Other applications that fit into this category are classic soft real-time applications such as video-on-demand that can show a static image for a few frames when overloaded, or audio applications.

Compositionality is a way to convert multiple independent timing requirements of different components of a system into a single real-time system timing requirement. Satisfying timing requirements together with logical correctness can make a real-time system predictable, guaranteeing that it will operate correctly when deployed. Abstracting timing at the higher level of system design permits designers to model the system by eliminating the low-level timing requirements of different components in cyber-physical systems. Since it is often preferable to work at higher levels of abstraction, compositionality analysis facilitates design productivity.

Motivated by feedback control systems that are robust to small delays in the feedback loop, [11] investigates an efficient periodic resource supply model for workloads that can tolerate transient overloads and delays and extends *dbf* and *sbf*-based techniques to analyze such systems. This dissertation discusses the relationship between characterization of overloads and the worst-case delay. This characterization is used to derive methods for finding a suitable resource to meet desired delay constraints.

This dissertation presents the analysis of periodic resource models using the supply and demand bound functions described in [75] by introducing overloads. Consequently, this extended model can find the resource supply required for a hierarchical or compositional system with transient overloads. In this dissertation, we mainly focus on the application to control systems. The workflow of the application to control systems is the following: (1) a control engineer models a physical system and defines the control objective, (2) the control

engineer designs a feedback controller for the plant based on a given sampling period, (3) the control engineer determines the maximum tolerable delay in calculating and applying the feedback inputs, based on the dynamics of the system, and (4) the system provisions the computational resources required to perform the computations within the given delay.

1.2 Multi-Mode Communication

Distributed real-time systems require reliable communication networks for the exchange of time-critical data. As the number and complexity of distributed stations increase in modern applications, traditional fieldbuses are quickly becoming obsolete due to reduced bandwidth and incompatible protocols that limit scalability and integration of multiple domains. Support for mixed-criticality and higher bandwidth will be mandatory for next-generation distributed systems, and these requirements are pushing towards the definition of a generic network capable of integrating multiple real-time domains with traditional computers in a single infrastructure.

Increasing interest on Real-Time Ethernet (RTE) provides strong evidence of this trend [32]. After years of research on exploring the real-time capabilities of Ethernet devices, some solutions are finally providing experimental evidence for hard real-time communication on top of Ethernet infrastructure [4, 3]. These solutions rely on enhanced devices with specific modules based on the Time Division Multiple Access (TDMA) approach typically used in safety-critical fieldbuses. Custom hardware implementations enable execution of predefined TDMA schedules with high synchronization accuracy while operating at the high-speed of Ethernet links.

State-based scheduling [36, 8, 10] is an alternative arbitration technique for hard real-time systems. State-based schedules allow developers to describe multiple operational modes, or states, and encode transitions from one state to another based on conditions that change at runtime. This property increases the flexibility, enables quick responses to changing operational conditions, and improves bandwidth utilization compared to static TDMA configurations, while keeping the system analyzable and verifiable. Multiple case studies show the advantages of this approach in domains such as control theory [86], hybrid systems [6], hierarchical scheduling [34], and in general bursty demand models [67]. The Network Code framework [38] provides a complete development environment including an expressive language to describe TDMA schedules with conditional branching, tools to verify the schedules before runtime, and a powerful hardware-accelerated platform to deploy and test solutions in practical scenarios [23]. However, a big limitation of this approach is that developers must still write the program at low-level, requiring a good understanding of the

underlying technology and runtime environment, and perform fine tuning of configuration parameters to fit particular application requirements. This approach is time-consuming, prone to errors, and inadequate for medium/large-scale systems, specially since conditional transitions add a new level of complexity to the schedule design and verification.

This dissertation presents a model-driven approach [8] to generate state-based schedules from high-level descriptions, which are then mapped to verified executable abstractions based on Network Code. The designer must provide an architectural description of the system, state-machine representations of the individual components and their interactions, and timing specifications. The proposed workflow then finds the system states, performs reachability and schedulability analysis, and generates abstract representations of state-based schedules based on specific design and application constraints. Designers can then use Network Code specific tools to translate these abstractions to Network Code programs, and generate synthesizable Hardware Description Language (HDL) code that optimizes the utilization of logical resources and power consumption for each distributed communication processor according to the generated schedule.

1.3 Synthesis of Computation and Communication

Computation and communication in safety-critical applications have to be completely separated and isolated from one another. The conservation of complexity is a justification of separation of concerns. With the growing focus on safety-critical systems, the principle of separation is increasingly important for adaptive embedded systems. Adaptive and reconfigurable embedded systems that integrate safety-critical and non-critical components, or that integrate safety and adaptive behaviours require separation of concerns to control system complexity. Resource dependencies can be minimized by factoring out the reservation and consumption parts into separate programs. The approach of splitting the whole program into a minimal set of resource dependencies makes the programs easier to understand and analyze. They can then be joined in a deterministic manner through specified timed interactions such as timed interfaces.

In an interface-based design, an interface describes how a component can be used. A well-designed component interface provides sufficient information to connect other components in a system [84]. Since an interface-based design of components in the system allows separation of concerns, scheduling mechanisms for computation and communication can differ. For example, in a computational component, a number of tasks execute on processors based on widely-used CPU scheduling algorithms such as EDF or Rate Monotonic

(RM) [76], and message transmission occurs from one component to another using a real-time network scheduling algorithm such as static TDMA or state-based TDMA [38]. An efficient CPU scheduling algorithm is not necessarily suitable for networks because of the differences in characteristics. In a system that supports multiple communication modes, state-based scheduling has lower average mode-change delay than EDF [8], which schedules tasks based on deadlines. Moreover, state-based TDMA schedules are suitable for achieving predictability, reliability, and safety in real-time networks [38]. However, finding a state-based schedule for a time-triggered architecture to satisfy the timing requirements of tasks is challenging, because the scheduler has to consider not only the requirements on each component, but also the global requirements of system-wide behaviour, including messages transmitted on the networks.

1.4 Contributions

This dissertation provides a novel approach for scheduling computation and computation through separation of concerns. Throughout the course of my thesis research, co-authors and I have made contributions on scheduling of computation and communication. The dissertation concentrates on a subset of them. In the context of scheduling of computation, co-authors and I have the following contributions:

C1 We characterize transient overloads for workloads under the EDF scheduling policy for a given resource supply [11]. We also derive a periodic resource supply model for periodic workloads with transient overloads [11] under the assumption that tasks deadlines are equal to their periods. We demonstrate that a control application can use the resource model to calculate the sufficient amount of resources even in the presence of tolerable overloads.

C2 We introduce the concept and design considerations for a mode-change technique that may use completed tasks stored in checkpoints to avoid unnecessary re-execution and reduce the mode-change delay incurred while switching between modes [9].

In the context of scheduling of communication, co-authors and I have the following contributions:

C3 We characterize the design considerations of high-confidence real-time communication [36] and demonstrate that state-based schedules have such characteristics.

C4 We exhibit the relationship of clocked graphs and state-based scheduling [69]. We show that clocked graphs can be used as an input language of state-based scheduling.

C5 We discuss using state-based scheduling [10] in an unreliable network, specially in a wireless environment.

C6 We discuss using state-based scheduling [27, 28] in networked control systems.

C7 We validate the technical feasibility of state-based schedules [27, 28] by implementing a plugin for TrueTime, which is a Matlab/Simulink-based simulator for real-time control systems.

C8 We propose a workflow from high-level specifications to find whether a valid and feasible state-based schedule exists for multi-mode communication which has low average mode-change delay [8].

In the context of scheduling both computation and communication together, co-authors and I have the following contributions:

C9 We analyze end-to-end delay for scheduling tasks and messages in a system [65].

C10 We define interfaces of overload-tolerant components using the derived periodic resource model in [11] and generate state-based schedules using the component interfaces.

This dissertation discusses **C1**, **C3**, **C8**, and **C10** to present closely related contributions for scheduling of overload-tolerant computation and multi-mode communication in real-time systems through separation of concerns. This separation of concerns not only is advantageous for safety certification but also ensures good performance.

1.5 Organization of the Thesis

The structure of the these is the following:

- Chapter 2 presents the background of real-time scheduling theory. This chapter discusses some of the known scheduling schemes for computation and communication along with an illustration of state-based scheduling.

- Chapter 3 presents the supply and demand bound analysis for overload-tolerant soft real-time systems through characterizing overloads in terms of overload metrics. This chapter presents schedulability conditions for overload-tolerant systems and an approach towards finding a suitable resource supply for a given workload and tolerable worst-case delay.

- Chapter 4 presents a model-driven design to allow generation of schedules from high-level specifications for practical implementation of state-based TDMA and leveraging the branching capabilities.

- Chapter 5 discusses overload-tolerant component interfaces that can be specified using the resource supply and characterization of transient overloads. A workflow discusses the steps necessary to generate state-based schedules using component interfaces for scheduling of both computation and communication through separation of concerns.

- Chapter 6 presents concluding remarks for scheduling of overload-tolerant computation and multi-mode communication through generation of state-based schedules using component interfaces.

- Chapter 7 presents some future works that can be done as extensions to improve and extend the work presented in this dissertation.

Chapter 2

Fundamentals

Applications require resources to operate. Such resources include computation time (i.e., access to a processing unit to execute instructions), memory (i.e., temporary or permanent data storage), and communication bandwidth (i.e., access to a shared medium to transmit information to remote stations). Before an application uses such resources, it must acquire them. A scheduling mechanism can control the access of resources given to applications. For example, the dispatcher in the operating system decides at each scheduling point which process that is ready to run to give access to the processing unit.

Real-time systems are time-sensitive and consist of time-critical distributed applications. If the computation is based on outdated data, incorrect result may appear. For example, a counter counts the number of occurrences of some specified external event and has to output the result. A timing failure on the input in observing the events can cause a functional failure on the output. This chapter presents an overview of scheduling schemes used for processors and network in real-time systems.

2.1 Scheduling of Real-Time Computation

Real-time systems are characterized by timing constraints that must be met to achieve the expected behaviour. Real-time systems execute tasks to perform the desired computational activity. Real-time tasks are usually of two types: hard and soft. A task will be said to be hard if the failure occurs because of missing the task deadline. Conversely, a task will be said to be soft if the system performance degrades but no failure occurs because of missing the task deadline. A number of parameters such as arrival time, execution time, start time,

completion time, deadline, response time, and slack characterize a task. Arrival time of a task is the time at which it becomes ready for execution, also referred as the release time. Execution time of a task is the time required to execute the task's computation without interruption. Start time of a task is the time at which it starts execution. Completion time of a task is the time at which it finishes execution. A task's typical timing constraint is one of two types of deadline: absolute and relative. The absolute deadline is specified with respect to time zero, and the relative deadline is specified with respect to the arrival time. Response time of a task is the difference between the completion time and the release time. Slack time of a task is the maximum time that the task can be delayed without missing its deadline.

A scheduling problem consists of a set of tasks with timing properties, and a set of resources (i.e., processors). In addition, precedence constraints among the tasks may exist. In this context, scheduling refers to assigning the processors and resources to tasks in order to complete the execution of all tasks without violating the timing requirements. A scheduling algorithm guarantees this. In the following, we present some of the traditional scheduling algorithms.

The RM [60] algorithm schedules a set of periodic independent hard real-time tasks based on priorities. RM scheduling assigns the priorities based on the task periods. The task with the shortest period is assigned the highest priority, and the task with the longest period gets the lowest priority. The scheduler selects the task with the highest static priority to run next.

EDF is a dynamic preemptive scheduling policy [60] that assigns priorities to tasks based on the earliest completion time. Therefore, the task with the earliest deadline is scheduled first, and the task with the longest deadline is scheduled last. The priorities are assigned dynamically based on the remaining time to the deadline of tasks. The scheduler selects the task with the highest priority to run next in the same way as RM.

The Least Laxity First (LLF) scheduling algorithm makes the same assumptions as the EDF algorithm. The scheduler chooses the task that has the shortest laxity. Laxity is the difference between a task deadline and the computation time of the task. Therefore, the task with the shortest laxity is assigned the highest priority.

Many variations of RM and EDF scheduling algorithms exist in the literature. Research work [31] on partitioned multiprocessor scheduling examined the use of EDF and RM, combined with bin-packing heuristics such as First-Fit, Next-Fit, Best-Fit, and Worst-Fit. The global multiprocessor scheduling [31] also exists in the literature such as global EDF, global LLF, Proportionate Fair (PFair), and Largest Local Remaining Execution Time First (LLREF) permits tasks to migrate from one processor to another.

In real-time systems, overloads can occur which cause different scheduling techniques to perform differently [22]. An overload is categorized into two types: transient and permanent. A transient overload can occur because of the excess demand of the execution time of tasks. This type of overload can occur from the simultaneous arrival of asynchronous events. If the utilization of the processor exceeds one for reasons like activation of a new periodic task or increasing the activation rate of existing tasks, then a permanent overload will occur for a uniprocessor system. For a multiprocessor system, the utilization must be no greater than the number of processors to avoid permanent overloads.

Another term related to overloads is called overrun. Overruns occur when a task execution time exceeds the expected time. This situation may occur for two reasons: (1) new jobs arrive more frequently than expected or (2) computation time exceeds its expected value. The difference between overload and overrun is that the latter is related to a single job whereas the overload is related to the processor. A single task overrun does not necessarily cause an overload in the system.

2.2 Scheduling of Real-Time Communication

Scheduling schemes for processors and for networks that permit distributed access to resources are different. Processor scheduling relies on network scheduling for the distributed access to resources. In addition, one difference between the processor scheduling and network scheduling is that the basic scheduling unit in network scheduling is a packet or message that contains information. The packet transmission cannot be preempted. The length of the packet defines the required time for transmission.

The Time-Triggered Protocol (TTP) [53, 54] integrates time-triggered communication with temporal error detection, a fault-tolerant clock-synchronization service, and a membership service. The schedules are generated off-line, and the parameters for the transmission slots are determined *a priori*.

An extension of switched Ethernet is Time-Triggered Ethernet (TTEthernet). TTEthernet [52] supports standard Ethernet traffic and provides a deterministic message transport. A TTEthernet switch supports two types of messages: (1) standard Event-Triggered Messages (ET-messages) and (2) deterministic Time-Triggered Messages (TT-messages). The formats of ET-messages and TT-messages follow the Ethernet standard, and the contents of the Ethernet type field or header make the difference between these two types of messages. The TTEthernet switch transmits TT-messages with a constant amount of small delay and ET-messages are transmitted when no TT-messages are transmitted. Dif-

ferent conflict resolution strategies exist for handling conflicts between ET-messages and TT-messages.

FlexRay [68] is a communication protocol for safety-critical applications designed by the FlexRay consortium. FlexRay is a combination of a time-triggered protocol and an event-triggered protocol. FlexRay uses a fault-tolerant clock synchronization that is similar to clock synchronization in TTP except for the membership service. The FlexRay event-triggered protocol is similar to the ARINC 629 mini-slotting protocol. FlexRay has two successive intervals: one for the time-triggered communication and the other one for the event-triggered communication.

Many researchers have proposed dynamic methods for the co-design of network scheduling and control applications. For example, Lluesma *et al.* [61] develop Jitterbug as a tool to analyze real-time control performance under the network induced jitter. Ji *et al.* [50] leverages a stochastic optimal method to analyze the communication time delay in the network, but focuses on the Networked Control System (NCS) with only one control application. Velasco *et al.* [80] propose a bandwidth allocation algorithm by adjusting the sampling period of the control application locally at run time to optimize the overall control performance. However, the algorithm needs accurate knowledge of the bandwidth utilization at run time as a prerequisite. Walsh and Ye [82] present a dynamic arbitration technique to grant network access to the control loop with the highest error using the Maximum-Error-First with Try-Once-Discard (MEF-TOD) scheme. Yepez *et al.* [88] propose a Largest Error First (LEF) scheduling algorithm based on the continuous feedback from the Quality of Service (QoS) of the control applications. Similar to [88], Xia *et al.* [87] propose a scheduling algorithm based on the importance of each control application. However, these dynamic scheduling algorithms in [87] focus on priority-based networks like CAN, and need a centralized scheduler to make the schedule decision. Branicky *et al.* [19] applied the RM algorithm to schedule a set of control applications in NCS, and the optimal scheduling algorithm was formulated that took both the RM constraints and the NCS-stability constraints into consideration. Ren *et al.* [72] propose a QoS management scheme for parallel NCSs. Hong [45] proposes a scheduling algorithm to adjust the data sampling time such that the performance requirement of each control loop is satisfied while the utilization of network resource is significantly increased. Later, an extension of this algorithm for the bandwidth allocation applies to the Controller Area Network (CAN) protocol [46], which can satisfy the performance requirements of real-time application systems and fully utilize the bandwidth of CAN. Rehbinder and Sanfridson [71] propose an optimal off-line scheduling method using the control theory. However, these algorithms are static without considering the workload variation.

State-Based Scheduling. A state-based schedule is a tree-like structure with a root

and a set of leaves where each vertex in the structure specifies a message to be transmitted and each edge a possible state transition. Edges contain enabling conditions. At run time, for each vertex exactly one edge is enabled at any given time. Whenever an execution reaches a leaf of the tree, it will loop back to the root. Figure 2.1 shows a state-based schedule for a patient monitoring system.

State-based schedules can still lead to unbounded communication delays, because the schedule itself may encode collisions on the medium which force retransmission. Developers must choose the right type of communication to prevent this. State-based schedules can model and execute two different types of traffic: guaranteed and best effort. Also, developers can increase the level of detail by either communicating individual variables or using general message passing. The difference between these types of communication lies within the ownership of the queues, meaning which stations know the different types of queues.

For example, the communication type of *guaranteed variable updates* will occur, if only one station transmits in that state of the state-based schedule and the transmission is specifically bound to a variable. The update is guaranteed since no other station will transmit and thus the communication will be free of collisions. Conversely, *best effort messaging* will occur, if more than one station is permitted to transmit data from their send queue in the state of the state-based schedule. If more than one station has a message in its send queue, then communication problems such as collisions or packets drop might occur. These different types of communication are visible from the specification of the state-based schedule, and the system also directly executes the state-based schedule as it gets encoded in the Network Code language [37].

Since the system will execute the state-based schedule at run time, developers can use the state-based schedule itself and state-space exploration on the schedule as evidence that the system works correctly. The schedule enables developers to provide upper bounds on the resource allocations for specific applications. Therefore, the developer can set and claim that certain variables will always receive bandwidth and stations will always receive updated values.

An Illustrative Example. Lets assume a distributed patient monitoring system in which body sensors transmit physiological parameters to the patient monitor. When the Pulmonary Vascular Resistance (PVR) of the patient passes a given threshold, the patient monitor will send an alarm message to the nurse station within bounded time.

PVR is the resistance in the pulmonary vascular bed against which the right ventricle must eject blood. To calculate the pulmonary vascular resistance, the patient monitor requires the Left Atrial Pressure (LAP) or the Pulmonary Capillary Wedge Pressure

(PCWP), the Pulmonary Artery Pressure (PAP), and the Cardiac Output (CO). PCWP provides an indirect estimate of LAP. PCWP is measured by wedging a catheter into a small pulmonary artery tightly enough to block flow from behind. LAP can be measured by placing a special catheter into the right atrium and then pushing through the inter-atrial septum. Since the patient monitor only requires the LAP or the PCWP, several modes can be created for the operation of the monitor:

- Configuration 1: The patient monitor uses the PAP, CO, and LAP.
- Configuration 2: The patient monitor uses the PAP, CO, and PCWP.
- Configuration 3: The patient monitor uses the PAP, CO, and LAP. If an alarm is pending, then the monitor will make a safety check and also acquire the PCWP, before signaling the nurse alarm. This will lower the number of false alarms as it eliminates the problem of incorrect LAP measurements.
- Configuration 4: This is similar to configuration 3, but the patient monitor first uses PAP, CO, and PCWP, and then uses LAP for the fail safe.

The calculation of the pulmonary vascular resistance is processed as a single transaction. This means that the system should always complete all data transmissions that the patient monitor requires before reconfiguring (e.g., changing configuration). This assumption is important, because the model assumes setting the configuration with a physical button which the nurse can press with a frequency of at most once in a fixed amount of time. In addition, the patient monitor must signal the nurse alarm within a bounded time when the pulmonary vascular resistance exceeds a specific threshold.

Based on the specifications, a designer will be able to find a state-based schedule. The system model assumes that communicating one value takes one time unit, and the inter-arrival time of the button pressed events is set accordingly. Figure 2.1 shows the state-based schedule that implements the specification. A vertex labeled ϵ takes zero time, and this is used to encode branches with more than two choices or for early termination of the schedule. The PVR monitoring station can operate in four configurations. In any of the four configurations, the monitoring system at first receives the value of the PAP from the circulatory system to calculate the PVR. If the received value is out of the normal range for PAP values (i.e. 10-20 $mmHg$), the system will enter the *safety interlock* state. In the *safety interlock* state, the system checks the important functions of the human cardiovascular system such as the patient's pulse rate while resting (60-100 beats per minute) to determine the patient's safety. This assumption is implicit and not shown

in the Figure 2.1. After receiving the value of CO within the normal range (4 L/min-8 L/min), the system can either receive LAP (normal range 6-12 $mmHg$) or PCWP (normal range 6-12 $mmHg$) based on the current configuration. The patient monitor will receive PCWP after PAP, if the system uses configuration 3. On the other hand, the monitor will receive LAP after the PAP, if the system runs in the default configuration (i.e., any configuration other than 1, 2, and 3). The system will enter into the safety interlock state for out of the normal range of CO, LAP, or PCWP. The system will generate an alarm and notify the nurse, when the PVR exceeds normal value (> 250 $dyn.s/cm^5$). The nurse can change the configuration of the monitoring system at any point in time but not in the middle of a transaction.

Guards g_1 and g_2 define the enabling conditions whether the PVR value of the patient exceeds the defined threshold thr. Guards g_3 to g_6 are enabling conditions depending on the configuration setting. Assume that configuration 4 is the default configuration.

For demonstration purposes, let's walk through a configuration that assumes $conf = 4$ and PVR $\geq thr$. In the root location labeled ϵ_0, only g_6 will be enabled. The state-based schedule specifies that the next three messages on the bus will be PAP, CO, and PCWP. At that point, PVR exceeds the threshold thr (PVR $\geq thr$), so g_1 is true, and the patient monitor will also receive the LAP measurement. Finally, g_1 will again be true, and the patient monitor will signal an alarm before the state-based schedule restarts at its root location.

The Network Code Framework. The Network Code framework provides the necessary abstractions and execution entities to describe and run state-based schedules for distributed real-time systems. The framework targets control at the Medium Access Control (MAC) layer using three components: (1) an assembly-like domain-specific language to represent state-based schedules, (2) a compiler with a verification engine that translates the programs into verified executable abstractions, and (3) a runtime entity that executes the schedules.

Network Code is a domain-specific programming language for the implementation of distributed real-time systems with TDMA communication systems. The language consists of a small set of assembly-like instructions with well defined operational semantics. Besides software prototypes, Network Code has been implemented as a hardware-accelerated special processor [38] on top of Ethernet and inside a network switch [24].

Listings 2.1, 2.2, and 2.3 show the Network Code programs that implement the schedule for the simple TMR system in Fig. 2.2. In a typical setup for TMR, three sensors transmit independent samples of the same variable in consecutive messages σ_1, σ_2, and σ_3. A voting controller receives these messages and performs a majority vote to determine the final

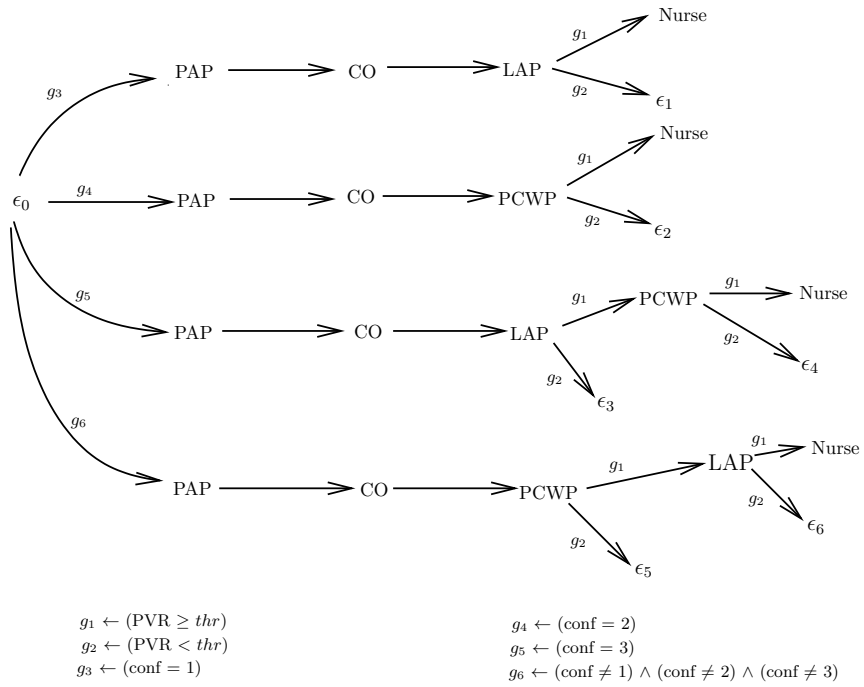


Figure 2.1: A state-based schedule for a patient monitoring system

value. On the one hand, in a static TDMA configuration, the sequence of transmitted messages is determined only by the progression of time, and then the stations will always transmit the three messages, even if σ_1 and σ_2 are already decisive for the voting. On the other hand, a state-based schedule can perform a preliminary voting after receiving the first two samples, and if the voting is already decisive, then the slot associated to the third sample can be empty, leaving the medium available for other purposes such as best-effort traffic. Fig. 2.2 illustrates this behavior. The system can operate in two modes: *mode* 1 considers that the system needs to communicate the three messages, and *mode* 2 considers that the vote is already decisive after the second message. The system starts in *mode* 1, and after transmitting σ_2 it checks the guard $g : \sigma_1 \neq \sigma_2$ to decide whether to transmit a message or leave the slot empty (ϵ) and available for other messages (such as best-effort data). After the third slot, a new round starts and the system resets to *mode* 1.

For simplicity, the system model assumes that all stations start execution at the same time and share a global time base. Network Code defines slots using the instructions future and halt. The instruction future(dl,L) starts a countdown timer with initial value dl. The instruction halt() stalls the program execution waiting for the expiration of this timer, and

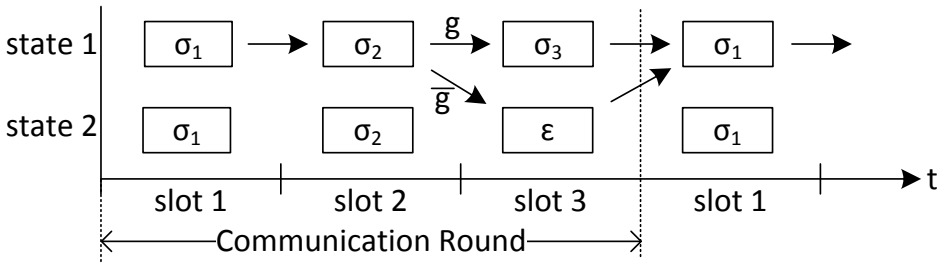


Figure 2.2: State-based representation of a TMR application

then resumes execution of the program at label L. The timespan of the countdown timer encodes the length of the slots. Instructions between future-halt blocks represent the actions for each slot.

Transmission of values is driven by create-send sequences. The branch(g, L) instruction enables the encoding of on-the-fly decisions. The guard g checks for specific conditions based on values of buffers, execution history, flags, etc. If the evaluated guard returns TRUE, the schedule will continue execution at the specified label L; otherwise, the schedule will continue with the next instruction. In the example, the third sensor verifies the transmitted values from the other sensors in the previous slots in the round. For simplicity, we assume that the station performed the corresponding receive instructions during the slot labeled L0.

Listing 2.1: Network Code program for sensor 1 in TMR example

```
L0:  future(3,L0);  % jump to L0 after 3 time units
     create(T1);    % create message T1 from local data
     send();        % send message T1
     halt();        % stall and wait for the timer
```

Listing 2.2: Network Code program for sensor 2 in TMR example

```
L0:  future(1,L1);  % jump to L1 after 1 time unit
     halt();        % stall and wait for the timer
L1:  future(2,L0)
     create(T2);
     send();
     halt();
```

Listing 2.3: Network Code program for sensor 3 in TMR example

```
L0:  future(2,L1);  % receive T1 and T2
```

```
        halt();
L1:     future(1, L0);
        if(T1=T2,L2);   % jump to L2 if T1=T2
        halt();
L2:     create(T3);
        send();
        halt();
```

The Network Code framework includes a powerful hardware-accelerated open-source platform that allows researchers to deploy and test their own solutions with tight timing guarantees in multi-segmented Ethernet networks [23]. However, while the framework offers tools to verify the schedules before runtime, developers must still write the program at low-level, requiring a good understanding of the underlying technology, and perform fine tuning of configuration parameters to fit particular application requirements.

Chapter 3

Scheduling of Overload-Tolerant Computation

Real-time applications have deadline constraints. The system should provision sufficient resources for the application to meet the deadlines, and use supply and demand bound functions to analyze the schedulability of workloads. The demand bound function determines the upper bound on the resources required by the application, while the supply-bound function specifies the lower bound on the resources supplied to the tasks. If the system provides fewer resources than required, the application will experience an overload. Most work concentrates on designing systems that cannot experience short periods of overloads.

This chapter presents analysis of resource provisioning for control applications that can tolerate overloads. It introduces analysis techniques for supply and demand bound functions that specifically consider overloads and delays in a periodic resource model. With this extended model, we address three problems: (1) determine the worst-case delay for a given resource demand and supply under a periodic resource model, (2) find a periodic resource supply for a given workload and worst-case tolerable delay, and (3) for a control system with a given robustness criterion, identify a periodic resource supply with a worst-case delay.

The remainder of this chapter is structured as follows: Section 3.1 presents an overview of the problem, and the motivating application of control systems is presented in Section 3.2. This produces a set of computational and tuning requirements, along with a specification on the maximum overload (i.e., delay) that can be tolerated. Section 3.3 presents the system model and shows how to characterize overloads using supply and demand bound functions. Section 3.4 presents methods on calculating a suitable resource

supply to meet the workload timing requirements in the presence of overloads. Section 3.5 demonstrates the use of the workflow for state feedback control of two plants. Section 3.6 presents some related work on schedulability analysis, specifically using supply and demand bound functions. Section 3.7 discusses what parameters in the system model affect the overloads.

3.1 Problem Statement

A supply bound function (*sbf*) and demand bound function (*dbf*) convert the timing requirements of the workload and the resource supply into a single timing requirement. Traditionally and informally, the *dbf* must stay below the *sbf* at all time intervals in order to avoid overloads. When permitting overloads, the *sbf* can remain below the *dbf* for bounded time intervals. Overloads then cause delays as the application must wait to receive sufficient resources. The worst-case delay δ^* is the maximum delay that tasks experience before they receive their requested resources.

In the context of control systems, a scheduling framework that supports overloads can help control engineers to design efficient and safe systems. A control system task-specification might include the worst-case tolerable delay in all time intervals. One can design an efficient resource supply to exploit the robustness of a given set of tasks to delays. Then, the following problem identifies the resource supply that the system designer needs to provide for the control application that permits overloads:

Goal: Given a control system workload $W = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a worst-case delay (δ^*), find the resource supply such that W is schedulable under the EDF scheduling policy and experiences a worst-case delay of at most δ^*.

A solution to this problem is applicable to hard, soft, and firm real-time systems. For a hard real-time system, δ^* must be zero. Soft real-time systems might specify some bound for δ^*. Firm real-time systems [63] may specify a δ^* with a probability of occurring.

3.2 Feedback Control Systems With Delays

Consider a physical system (plant) modeled as a linear time-invariant system of the form

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{3.1}$$

where t is the time variable, $x(t) \in \mathbb{R}^n$ is the state of the system, $u(t) \in \mathbb{R}^m$ is the control input applied via the actuators, and matrices A and B are of appropriate dimensions. This model is obtained from the physical processes governing the system under consideration. A typical objective is to choose the input $u(t)$ so that the system is *stable* in the following sense.

Definition 1 *The system (3.1) is said to be asymptotically stable if* $\lim_{t\to\infty} x(t) = 0$ *for any initial condition* $x(0)$*.*

When the full state $x(t)$ is measurable and the pair (A, B) satisfies an algebraic property known as *stabilizability* [26], it is possible to find a state-feedback control input of the form $u(t) = Kx(t)$ such that the system is asymptotically stable (where K is an appropriate $m \times n$ gain matrix). When the plant is controlled over a network, however, stabilization is complicated by issues such as sampling, delays and packet drops. There has been a large amount of research devoted to characterizing conditions under which stabilization is possible, for various assumptions on the system and the network [89, 44, 30, 48, 42, 73, 66]. This work follows the approach in [30], which presented a general and computationally efficient method to obtain bounds on the delays that can be tolerated by a given control system.

First, we assume that the plant state is sampled every p seconds to produce the state measurements $x(t_k)$, where $t_k = kp$ for $k \in \mathbb{N}$. These state measurements are then sent over the network to the controller (i.e., a computational resource), which calculates the control input $u(t_k) = Kx(t_k)$ and sends this value to the plant's actuators, where it is held until the next input is received. There is a delay μ_k incurred between measuring the plant's state at time t_k and applying the input $u(t_k)$. Thus, as in [30], the system evolves as follows:

$$\dot{x}(t) = Ax(t) + BKx(t_k), \;\; t \in [t_k + \mu_k, t_{k+1} + \mu_{k+1}).$$

Let μ^* be the maximum possible delay over the network (i.e., $\mu^* = \sup_{k\in\mathbb{N}} \mu_k$). The following result from [30] provides a method to determine whether the system will be stable with a given worst-case delay and feedback gain K.

Theorem 1 ([30]) *For a given scalar* η *and matrix* K*, if there exist matrices* $P > 0$*,* $T > 0$*,* N_i *and* M_i *(*$i = 1, 2, 3$*) of appropriate dimensions such that Equation 3.3 is true, then the system is asymptotically stable with the state feedback input* $u(t) = Kx(t_k)$*,* $t \in [t_k + \mu_k, t_{k+1} + \mu_{k+1})$*, as long as the sampling period* p *and worst-case delay* μ^* *satisfy*

$$p + \mu^* \leq \eta. \tag{3.2}$$

$$\begin{bmatrix} N_1 + N_1^T - M_1 A - A^T M_1^T & N_2^T - N_1 - A^T M_2^T - M_1 BK & N_3^T - A^T M_3^T + M_1 + P & \eta N_1 \\ * & -N_2 - N_2^T - M_2 BK - K^T B^T M_2^T & -N_3^T + M_2 - K^T B^T M_3^T & \eta N_2 \\ * & * & M_3 + M_3^T + \eta T & \eta N_3 \\ * & * & * & -\eta T \end{bmatrix} < 0. \tag{3.3}$$

For a square symmetric matrix P, the notation "$P > 0$" in the above theorem indicates that the matrix is *positive definite*. The matrix in (3.3) is symmetric, and to save space, the $*$ symbols are used as placeholders for the appropriate matrix elements. When η is a fixed constant, the above expression is a *Linear Matrix Inequality*, which can be solved efficiently for the unknown matrices P, T, N_i and M_i using convex programming software such as `CVX` [17, 41]. One can find the largest value of η for which the system will be stable by using bisection.

To relate the above characterization of stability to the characterization of overloads or delays, we note that the worst-case delay δ^* represents the longest length of time *after the end of any task's period* that would be required for the necessary computational resources to become available. Thus, from the perspective of the control system, the longest possible delay seen by a packet generated at time t_k would be $\mu^* = p + \delta^*$ (i.e., the length of one period plus the maximum additional time required to obtain the desired resources). Thus, once we find a worst-case value for η from Theorem 1, we can obtain an upper bound on δ^* from Equation (3.2) as

$$\delta^* = \eta - 2p.$$

Figure 3.1 outlines how developers can use the results of this work. After the control engineer designs the system, she computes the *dbf* of the application, and specifies the upper bound on the worst-case delay (for example, using the technique described above). Second, using our algorithms as specified in Section 3.4, the engineer finds a resource supply for the resource of interest (e.g., the processor). Third, the engineer analyzes the *sbf*, the *dbf*, and the control system to determine whether the found supply is sufficient for the system (e.g., worst-case delay remains below the specified bound). If the found supply fits the system, then in the fourth step, the engineer can use the supply to deploy the system; otherwise, the engineer can tweak the constraints on the supply generation and find a different supply.

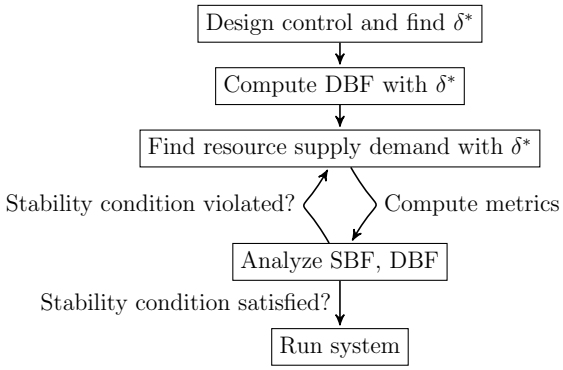


Figure 3.1: Proposed workflow to use resource supply

3.3 Overloads in Supply and Demand Curves

The system model consists of a periodic resource model and a periodic workload, consisting of a set of tasks (e.g., the tasks to process the feedback control signals). A task τ_i is characterized by a tuple (p_i, e_i) where p_i is the period and e_i is the worst-case execution time. We assume the deadline d_i of task i is equal to p_i. A set of tasks or a workload is characterized by a set of tuples: $\{(p_1, e_1), \ldots, (p_n, e_n)\}$. This work assumes that all n tasks in the system are fully preemptive and have a known tolerable delay. The hyperperiod of all tasks' periods forms the cycle at which the system repeats its behaviour. A periodic resource model indicates resource replenishment in each period. Given a periodic resource model $R(\lambda, \theta)$, tasks are allocated for θ time units in every interval $[k\lambda, (k+1)\lambda], k \in \mathbb{N}$. A scheduling model M consists a workload W, a resource model R, and a scheduling algorithm A. This work uses the EDF scheduling policy.

3.3.1 Supply and Demand Bound Functions

Supply and demand bound functions are used to determine schedulability under a particular scheduling policy. Supply and demand bound functions facilitate exact schedulability analysis during all time intervals, rather than sufficient analysis. A demand bound function is used to find the maximum resource demand during a time interval. On the other hand, a supply bound function is used to calculate the minimum resource supply during a time interval.

For a given workload, the dbf is the maximum possible resource demand in any time interval t. Obtained from [76], Equation 3.4 shows how to calculate the resource demand

for n tasks for the EDF scheduling scheme for a time interval (Fig. 3.2) of length t:

$$\mathrm{dbf}(W, EDF, t) = \sum_{\tau_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor e_i. \tag{3.4}$$

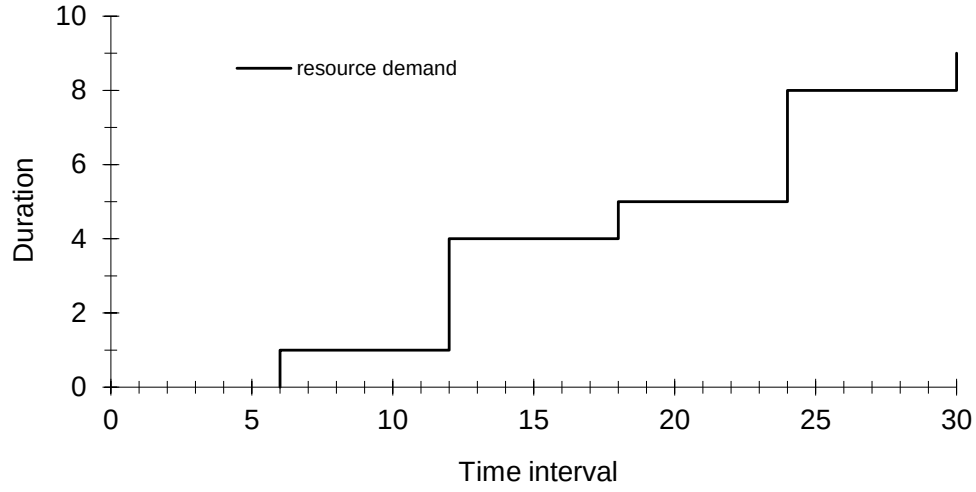


Figure 3.2: $\mathrm{dbf}(t)$ for $W = \{\tau_1(6,1), \tau_2(12,2)\}$

A periodic resource supply $R(\lambda, \theta)$ provides θ time units in every λ time units. The supply bound function calculates the minimum resource supply during any time interval t. Using Equation 4.1 from [76], it is possible to find the minimum resource supply during any time interval (Fig. 3.3) of length t as:

$$\mathrm{sbf}(t) = \begin{cases} (t - (k+1)(\lambda - \theta)) & \text{if } t \in [k_1, k_2] \\ \\ (k-1)\theta & \text{otherwise} \end{cases} \tag{3.5}$$

with $k_1 = (k+1)\lambda - 2\theta$, $k_2 = (k+1)\lambda - \theta$, and k as

$$k = \max(\lceil (t - (\lambda - \theta))/\lambda \rceil, 1).$$

Note that, the value of k is greater than or equal to 1.

Since a periodic resource $R(\lambda, \theta)$ guarantees a supply of at least θ time units in every interval $[k\lambda, (k+1)\lambda], k \in \mathbb{N}$, the model leaves open how the guaranteed θ time units are

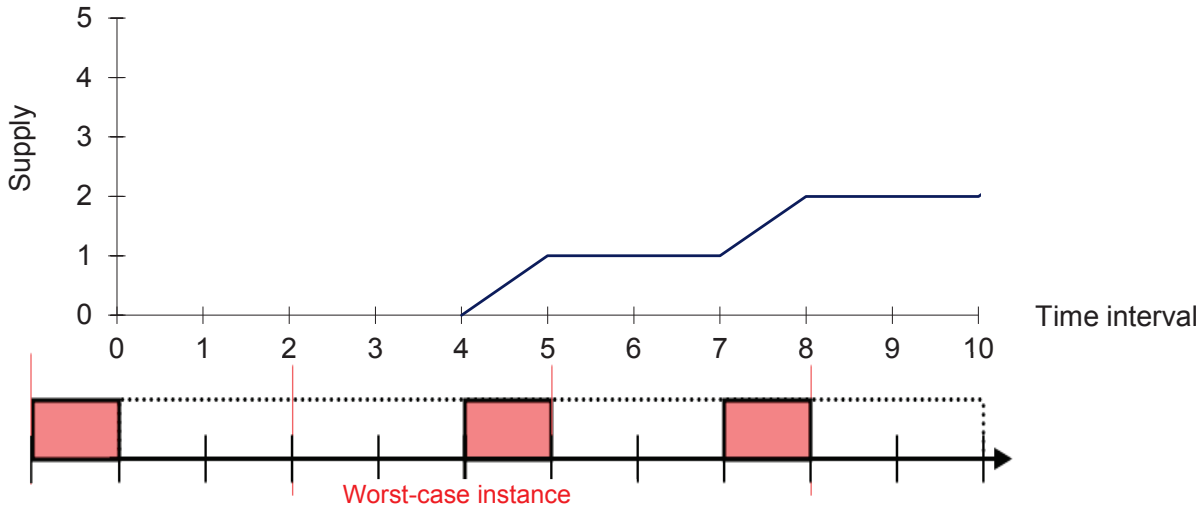


Figure 3.3: sbf(t) for $\lambda = 3$ and $\theta = 1$

distributed over a time interval of size λ. An instance of the periodic resource is a time trace of resource allocations that satisfies the guarantee (λ, θ).

For a given scheduling model $M(W, R, EDF)$, if the resource demand of W is no greater than the resource supply of R during any time interval, then M is schedulable. Therefore, M is schedulable if $\forall t : dbf(t) \leq sbf(t)$.

Example 1 *Consider a periodic resource supply* $R(3, 1)$ *and scheduling model* $M(W, R, EDF)$ *that has two tasks in the workload,* $W = \{\tau_1(6, 1), \tau_2(12, 2)\}$. *Fig. 3.4 shows the computed sbf and dbf. This workload is not schedulable with the given resource if the workload cannot tolerate any delay, because the supply is not always greater or equal than the demand.*

3.3.2 Characterizing Overloads

While previous work on the periodic resource model [78] only discusses resource supplies and demands for which the *sbf* is always less than the *dbf*, our work focuses on using the periodic resource model in systems for which the *dbf* can be greater than the *sbf* for some time intervals. Figure 3.5 shows such overloads. Figure 3.6 shows a more detailed view of a single overload.

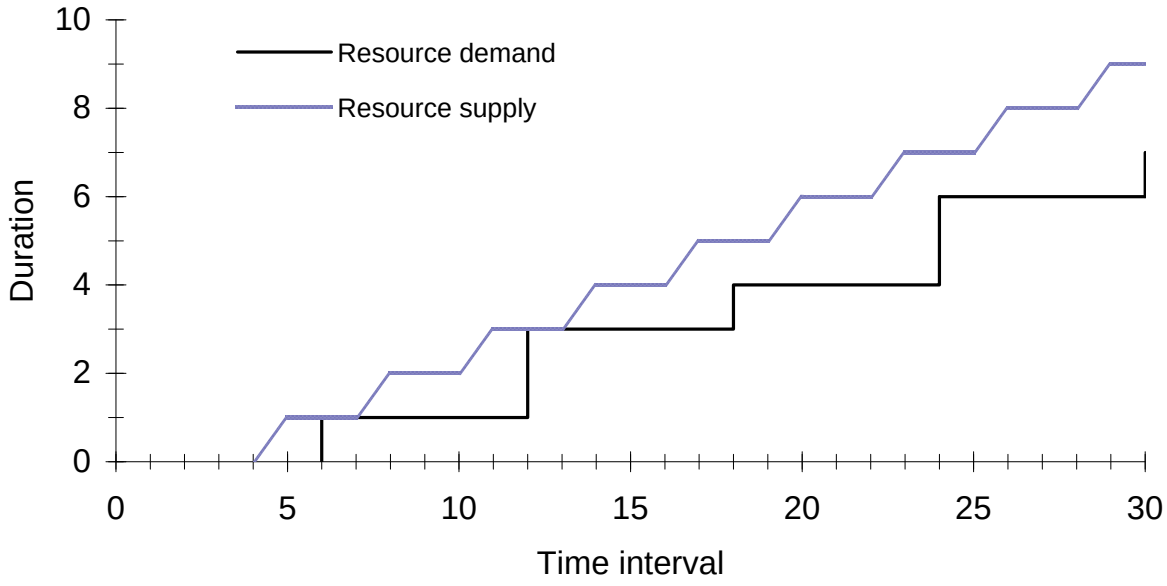


Figure 3.4: An example of *sbf* and *dbf* where $\forall t$: $\text{sbf}(t) \geq \text{dbf}(W, t, EDF)$

Definition 2 (Overload) *An overload is said to occur in a time interval of length t when* $\exists t > 0 :$ *dbf(t)* $\geq$ *sbf(t).*

Example 2 *Slightly changing the workload to* $W = \{\tau_1(6,1), \tau_2(12,2)\}$ *makes the system infeasible to schedule if the periodic resource supply is* $R(3,1)$. *Figure 3.5 shows the new* dbf. *The system is infeasible because in a time interval of* $t = 12$, *the system can experience an overload, since the* dbf *is greater than the* sbf.

An overload starts and ends at a specific time interval at which the *sbf* and the *dbf* intersect before the *dbf* becomes greater than the *sbf*. The points at which this intersection happen are called *points of interest*.

Definition 3 (Points of Interest) *A point t is an overload point* (t_o) *if*

$$\exists \pi > 0 : \forall \epsilon \in (0, \pi] \, \text{sbf}(t - \epsilon) \geq \text{dbf}(t - \epsilon) \\ \textit{and } \text{sbf}(t) < \text{dbf}(t). \tag{3.6}$$

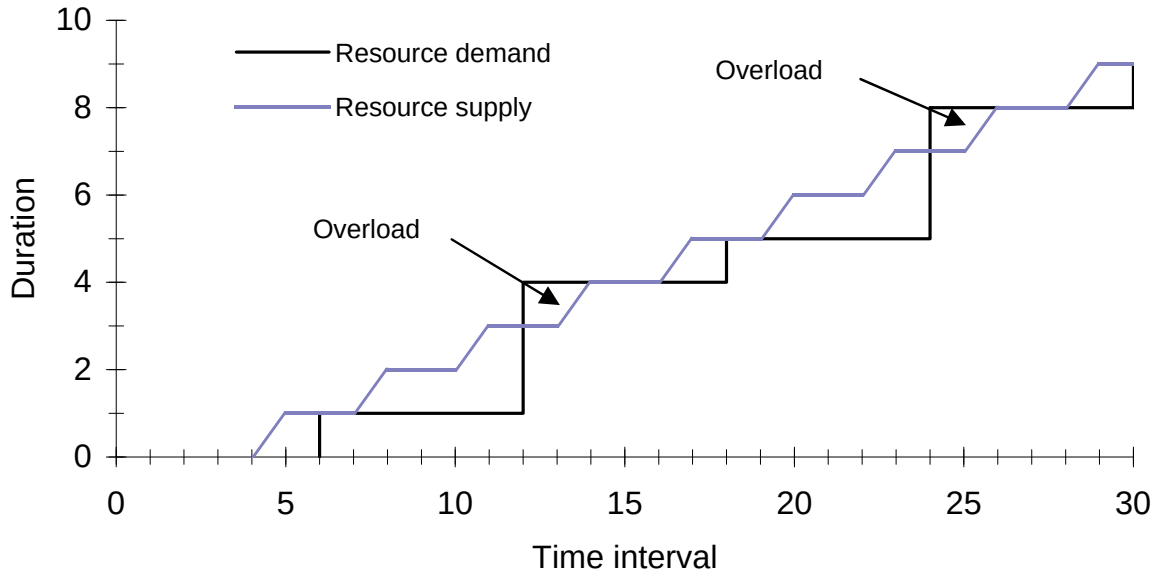


Figure 3.5: An example of an overload ($\exists t : \text{sbf}(t) < \text{dbf}(W, t, EDF)$)

A point t is a recovery point (t_r) if

$$\exists \pi > 0 : \forall \epsilon \in (0, \pi] \, \text{sbf}(t - \epsilon) < \text{dbf}(t - \epsilon) \\ \textit{and } \text{sbf}(t) = \text{dbf}(t). \tag{3.7}$$

In this work, an overload point t_o is the earliest possible integer point that Equation 3.6 satisfies. This overload point is associated with a recovery point t_r, which is the earliest integer point Equation 3.7 satisfies.

We use these points of interest to define the *duration* and *severity* of an overload. The first recovery point with a time interval greater than an overload point is the *associated* recovery point. Informally, this is the point at which the *sbf* catches up to the *dbf* again.

Definition 4 (Overload Duration) *For a given overload point t_o and its associated recovery point t_r such that $t_o \leq t_r$, the* duration *of an overload is $t_r - t_o$ when $t_o \geq t$, or $t - t_o$ when $t_o \leq t \leq t_r$, where t is the length of the time interval under consideration.*

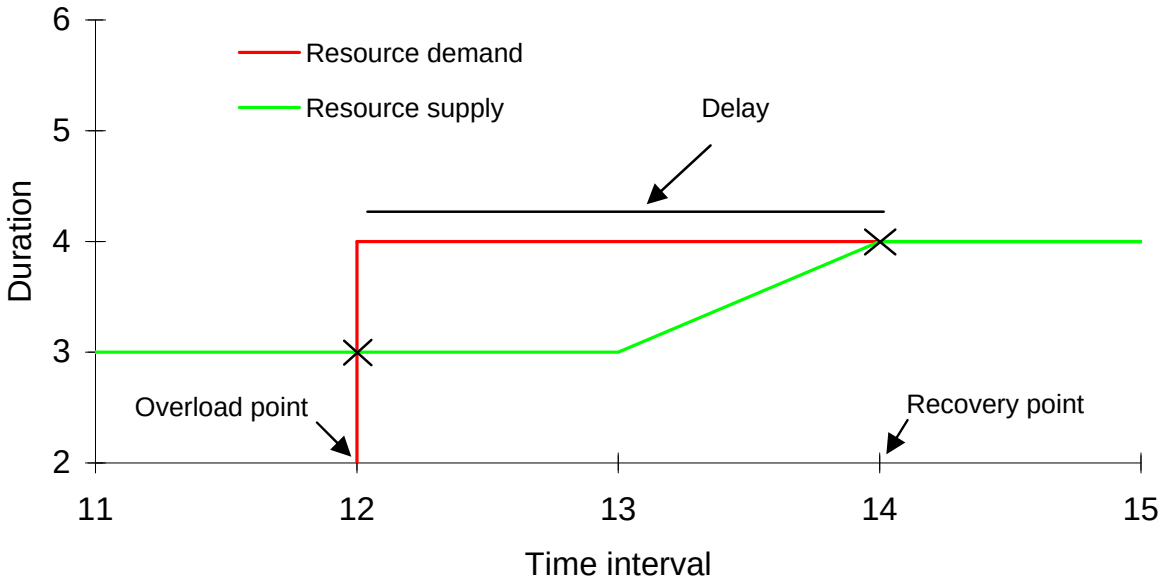


Figure 3.6: A detailed view of an overload shown in Figure 3.5

The duration of an overload is a useful metric when designing the system. The existence of a given overload point t_o and its associated recovery point t_r means there exist time intervals of length t_r in which the system may be overloaded. However, at the same time, for time intervals greater or equal to t_r, the system no longer experiences an overload. Thus the difference $t_r - t_o$ specifies the *delay* that tasks experience when waiting for their resources. However, if no such recovery point t_r exists, the delay is taken to be $t - t_o$, where t is the time interval under consideration.

Observation [Duration=Delay]: For a given overload point t_o and its associated recovery point t_r, the duration of the overload $t_r - t_o$ or $t - t_o$ characterizes the delay that tasks experience during the overload before receiving their demanded resource.

Definition 5 (Worst-case Delay) *For a scheduling model $M(W, R, EDF)$ with a periodic resource $R(\lambda, \theta)$ and a workload W, then under the EDF policy, the maximum of all overload durations will be the worst-case delay (δ^*) of any task.*

The following observations limit the locations of points of interest:

1. For an overload point t_o, the *dbf* can only exceed the *sbf* at points t where the *dbf* increases. The *dbf* only increases at $t = m \cdot p_i$ for some positive integer m and task period p_i. Thus, if the system contains overload points, then they must be values in the set $\{mp_i : m \in \mathbb{N}^+\}$.

2. For a recovery point t_r, the *sbf* can only be equal to or greater than the *dbf* at points t where the *sbf* is increasing, i.e., at $t = c\lambda + 2(\lambda - \theta) + r$, where c is a positive integer and $1 \leq r \leq \theta$. Thus, if the system contains recovery points, then they must be of this form.

We now show that when the resource utilization is equal to the workload utilization, the overload characteristics of the system are periodic. In the process, we characterize the length of the largest time interval that has to be considered to analyze the system. To do this, we define the function

$$f(t) = \text{sbf}(t) - \text{dbf}(t), \quad t \in \mathbb{R}_{\geq 0}. \tag{3.8}$$

Note that the values of t for which $f(t) < 0$ correspond exactly to time intervals where the system is experiencing an overload.

Theorem 2 *Consider a system with workload utilization* $U_W = \sum_i \frac{e_i}{p_i}$ *and resource utilization* $U_R = \frac{\theta}{\lambda}$. *If* $U_R = U_W$, *then after* $t = 2(\lambda - \theta)$, *the function* $f(t)$ *is periodic with period* $LCM(\lambda, p_1, \ldots, p_n)$, *i.e.,*

$$\begin{aligned} &f(2(\lambda - \theta) + t + kLCM(\lambda, p_1, \ldots, p_n)) \\ &\qquad = f(2(\lambda - \theta) + t), \quad \forall t \in \mathbb{R}_{\geq 0}, \forall k \in \mathbb{N}. \end{aligned}$$

Proof First, one can verify from Equations (3.4) and (4.1) that $\text{dbf}(t)$ and $\text{sbf}(t)$ satisfy

$$\begin{aligned} \text{dbf}(t + k\text{LCM}(p_1, \ldots, p_n)) &= \text{dbf}(t) + \\ &\quad k\text{LCM}(p_1, \ldots, p_n)U_W, \forall t \in \mathbb{R}_{\geq 0}, \forall k \in \mathbb{N} \\ \text{sbf}(2(\lambda - \theta) + t + k\lambda) &= \text{sbf}(2(\lambda - \theta) + t) + k\theta, \\ &\quad \forall t \in \mathbb{R}_{\geq 0}, \forall k \in \mathbb{N}. \end{aligned}$$

Using these identities, we obtain

$$\begin{aligned} f(2(\lambda-\theta)+t+k\mathrm{LCM}(\lambda,p_1,\ldots,p_n)) \\ &= \mathrm{sbf}(2(\lambda-\theta)+t+k\mathrm{LCM}(\lambda,p_1,\ldots,p_n)) \\ &\quad - \mathrm{dbf}(2(\lambda-\theta)+t+k\mathrm{LCM}(\lambda,p_1,\ldots,p_n)) \\ &= \mathrm{sbf}(2(\lambda-\theta)+t)+k\mathrm{LCM}(\lambda,p_1,\ldots,p_n)U_R \\ &\quad - \mathrm{dbf}(2(\lambda-\theta)+t)-k\mathrm{LCM}(\lambda,p_1,\ldots,p_n)U_W. \end{aligned}$$

When $U_R = U_W$, this expression becomes

$$\begin{aligned} f(2(\lambda-\theta)+t+k\mathrm{LCM}(\lambda,p_1,\ldots,p_n)) \\ &= \mathrm{sbf}(2(\lambda-\theta)+t)-\mathrm{dbf}(2(\lambda-\theta)+t) \\ &= f(2(\lambda-\theta)+t), \end{aligned}$$

which proves the theorem.

The function $f(t)$ fully captures the relative behavior of the supply bound function and the demand bound function, and the entire function $f(t)$ is characterized by its values in the interval $[0, 2(\lambda-\theta)+\mathrm{LCM}(\lambda,p_1,\ldots,p_n))$. Thus, we only need to analyze the system for intervals up to this length to determine schedulability. We will use this fact in the rest of the chapter.

Algorithm 1 Finding all overload points in intervals of length up to t

Output: Ordered list of overload points L_o

```
for every i → 1, ..., |W| do
    for every 1 ≤ m ≤ ⌊t/p_i⌋ do
        if mp_i satisfies (3.6) then
            L_o ← L_o ∪ mp_i
        end if
    end for
end for
```

3.3.3 Computing the Points of Interest

Overloads can only occur at the points where the *dbf* increases, because the *sbf* is a monotonically increasing function. A recovery can only occur at points where the *sbf*

increases and the *dbf* remains flat. Since overload points and recovery points are located at intersection points, it is possible to find these points by solving the equation sbf(t)= dbf(t):

$$\begin{cases} \sum_{\tau_i \in W} \lfloor \frac{t}{p_i} \rfloor e_i = (t - (k+1)(\lambda - \theta)) & \text{if } t \in [k_1, k_2] \\ \\ \sum_{\tau_i \in W} \lfloor \frac{t}{p_i} \rfloor e_i = (k-1)\theta & \text{otherwise.} \end{cases} \tag{3.9}$$

We use the algorithms (Algorithm 1 and Algorithm 2) to find the overload and recovery points to Equation 3.9. Algorithm 1 identifies all overload points in any interval of length t. Algorithm 2 computes the recovery points associated with each overload point. The algorithm uses Equation 3.7 to determine the points.

Algorithm 2 Finding all (t_o, t_r) pairs in intervals of length up to t

Output: List of (t_o, t_r) pairs L_r

```
for every i → 1, ..., |W| do
    for every tuple of consecutive t_o^i, t_o^j ∈ {L_o ∪ t} do
        if ∃t_r with t_o^i < t_r < t_o^j which satisfies (3.7) then
            L_r ← L_r ∪ ⟨t_o, t_r⟩
        end if
    end for
end for
```

Example 3 *Continuing from Example 1, three overload points (t_o) and three recovery points (t_r) exist in all time intervals t of length $0 < t \leq (LCM(6,12) + 2(3-1))$ as defined in Theorem 5.7. The overload and associated recovery points up to the hyperperiod are:* $(12, 14)$. *Hence, the worst-case delay is two units. Figure 3.6 shows a tuple of an overload and a recovery point where the worst-case delay occurs.*

A system enters into *continuous overload* if $\exists t_o : \forall t > t_o$ sbf(t) $<$ dbf(t). If the resource utilization is less than the workload utilization, the system will eventually experience continuous overload.

3.3.4 Schedulability Analysis with Overloads

Schedulability analysis is one of the key requirements in real-time systems. A hard real-time system will be schedulable if sbf(t)$\geq$ dbf(t) at any time interval t. However, the schedu-

lability condition $\text{sbf}(t) \geq \text{dbf}(t)$ in any time interval t is not applicable for soft real-time system that can tolerate overloads. Therefore in the following, the schedulability analysis condition for EDF is defined in the presence of overloads (Theorem 3), characterized by the maximum tolerable delay.

Theorem 3 *Given a system workload $W = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with tolerable δ^* and a given resource model, W is schedulable if and only if the resource demand in any time interval exceeds the resource supply during the same time interval for no more than δ^* consecutive time units. Furthermore, this only has to be checked for time intervals up to length $LCM(p_1, \ldots, p_n, \lambda) + 2(\lambda - \theta)$.*

Theorem 5.7 establishes the proof of Theorem 3, because $f(t) = \text{sbf}(t) - \text{dbf}(t)$ repeats after $\text{LCM}(p_1, \ldots, p_n, \lambda) + 2(\lambda - \theta)$.

Since searching up to $\text{LCM}(p_1, \ldots, p_n, \lambda) + 2(\lambda - \theta)$ for each possible λ and θ will lead to different bounds on time intervals to search for suitable resource supplies, the following over-approximated observation can be used as a bound to search for λ .

Observation [Search Interval] Considering possible overloads at every instance of task period and recovery at the distance of maximum tolerable delay δ^* from overload points, the time interval to search for valid and feasible λ and θ is no greater than $\text{LCM}(p_1, \ldots, p_n) + \delta^*$.

3.4 Finding an Efficient Resource Supply

For a given system specification consisting of a workload and a worst-case delay, the objective of the developer is to provision the system with an applicable resource supply. In [78], the authors show how to calculate θ under the *EDF* scheduling policy with a given demand and resource period λ. Since our approach permits overloads, the technique specified in [78] is inapplicable. Furthermore, our target is to find θ without a pre assumed λ.

Since many possible resource supplies exist for a given workload, our method of calculating an efficient resource supply uses a cost function to choose one resource supply among many. The resource period λ and the supply θ are the parameters of the cost function. Our approach not only focuses on increasing the system throughput by lowering the resource usage (corresponding to a small θ/λ), but also reducing the number of context switches (corresponding to a large λ). A larger λ will decrease the number of context

switches because the resource accounting mechanism in the operating system may preempt the workload less frequently.

Assuming tasks with periods equal to deadlines, periodic transient overloads ($U_W = U_R$), and a periodic resource model, we propose an efficient periodic resource supply model and calculate λ and θ using the following lines:

- the diagonal lines (e.g., lines 0, 1, 2, and 3 in Figure 3.7) that pass through the points where $sbf(t)$ increases,
- the horizontal lines (e.g., lines 4, 5, and 6 in Figure 3.7) that are parallel to the x-axis and pass through the $dbf(t)$ where $dbf(t) > 0$,
- the vertical lines (e.g., lines 7, 8, and 9 in Figure 3.7) that pass through the points where $dbf(t)$ might equal to $sbf(t)$ after an overload occurred at $t - \delta^*$.

The diagonal, horizontal, and vertical lines intersect (as shown in Figure 3.7) for a certain λ and the corresponding θ. The calculation of the lines of interest is as follows.

Calculating diagonal lines: The diagonal lines as shown in Figure 3.7 intercept the points where $sbf(t)$ increases at periodic intervals after an initial time-interval offset of $2(\lambda-\theta)$. The sbf increases by θ in each time interval of length λ after the initial offset. At each $t = 2(\lambda - \theta) + k(\lambda - \theta), k \in \mathbb{N}$, the slope of sbf (t) is 1 (this assumes a uniprocessor system). The diagonal lines are of interest because they represent all the points where the sbf increases. Equation 3.10 represents the set of all diagonal lines.

$$\{y_k(t) = (t - (k+2)(\lambda - \theta)), k \in \mathbb{N}, t \in \mathbb{N}, k \leq \text{LCM}(\{p_n\}) + \delta^*, t \leq \text{LCM}(\{p_n\}) + \delta^*\}. \tag{3.10}$$

Example 4 *Using the workload* $W = \{\tau_1(6,1), \tau_2(12,2)\}$ *presented in Example 1, the tolerable worst-case delay* $\delta^* = 2$*, and Equation 3.10, the following equations correspond to the four diagonal lines such that* $k = 0, \ldots, 3$ *which are no greater than the search interval* $LCM(p_1, \ldots, p_n) + \delta^* = 14$*.*

$$y_0(t) = t - 2(\lambda - \theta), \tag{3.11}$$
$$y_1(t) = t - 3(\lambda - \theta), \tag{3.12}$$
$$y_2(t) = t - 4(\lambda - \theta), \tag{3.13}$$
$$y_3(t) = t - 5(\lambda - \theta). \tag{3.14}$$

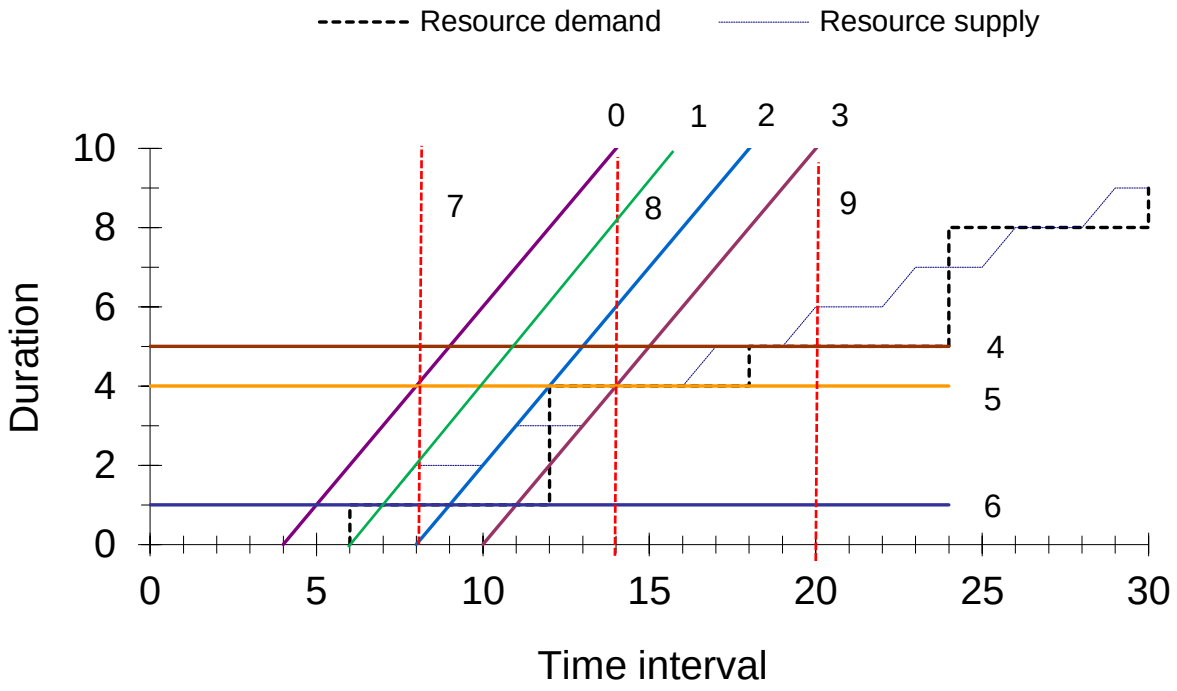


Figure 3.7: Finding intersection points for a given *dbf* and δ^*

Calculating horizontal lines: The horizontal lines shown in Figure 3.7 intercept the y-axis at the sum of the execution time units of a currently executing instance of a task and the execution time of all the preceding periodic instances of the current task and higher priority tasks. The y-intercept points are from the horizontal lines drawn on the $dbf(t)$ such that $dbf(t) > 0$. The horizontal lines are of interest because they contain the points where $sbf(t)$ may be equal to $dbf(t)$.

To devise an equation for representing the horizontal lines that intercept the y-axis, we assume a vector v containing the execution times of all tasks (i.e., $v = (e_1, e_2, \ldots, e_n)$). We also define C as a set of indices that refer to the possible number of instances of a task up to the search interval.

$$C = \left\{ (\alpha_1, \ldots, \alpha_n) \mid \forall p_i \in W, t = mp_i, m \in \mathbb{N}^+ : \alpha_i = \left\lfloor \frac{t}{p_i} \right\rfloor, t \leq \text{LCM}(p_1, \ldots, p_n) + \delta^* \right\}. \tag{3.15}$$

Equation 3.16 represents the set of horizontal lines that originate from y-intercept points.

$$D = \left\{ y \mid y = \alpha v^T, \alpha \in C \right\}. \tag{3.16}$$

(Continuing Example 4). In the following, the horizontal lines may represent the lines at the y-intercept points using Equation 3.16 for Example 4 until search interval (i.e., $t = 14$).

$$C = \{(1, 0), (2, 1)\},$$
$$D = \{y_0 = e_1, y_1 = 2e_1 + e_2\}.$$

Calculating vertical lines: Finally, the solid vertical lines shown in Figure 3.7 intercept the x-axis at the positive integer multiples of p_i of each task T_i. Since the time intervals p_i represent the time intervals where an overload might have occurred, the recovery points will be located δ^* distance to the overload points. Therefore, these vertical lines will be shifted right by an amount of the worst-case delay δ^* (e.g., lines 7 , 8, and 9) that may pass through the recovery points where $sbf(t) = dbf(t)$ when there is an overload at time interval $t - \delta^*$.

The dbf increases at time intervals of length $p_i\omega_i$ where $\omega_i \in \mathbb{N}^+$ for $\tau_i \in W$. Equation 3.17 represents the set of the vertical lines where $sbf(t) = dbf(t)$.

$$S = \left\{ x \mid x = p_i\omega_i + \delta^*, p_i \in W, \omega_i \in \mathbb{N}^+, \omega_i \leq \left\lfloor \frac{\text{LCM}(p_1, \ldots, p_n) + \delta^*}{p_i} \right\rfloor \right\}. \tag{3.17}$$

(Continuing Example 4). In the following, the horizontal lines represent the first few lines drawn at the x-intercept points where $sbf(t)$ might equal to $dbf(t)$ using Equation 3.17 for Example 4.

$$S = \{x_0 = p_1 + 2, x_1 = 2p_1 + 2, x_2 = p_2 + 2\},$$
$$S = \{x_0 = 8, x_1 = 14, x_2 = 20\}.$$

To find λ and θ, our method uses the intersection points of diagonal, horizontal, and vertical lines resulting from these equations. First, we solve Equation 3.10 with Equation 3.16. This yields,

$$t - (k + 2)(\lambda - \theta) = \alpha v^T.$$

Then we replace θ and t using $\sum \frac{e_i}{p_i} = \frac{\theta}{\lambda}$ (i.e., $U_W = U_R$) and Equation 3.17,

$$t = p_i\omega_i + \delta^*.$$

This yields,

$$\begin{aligned}
\lambda &= \theta - \left(\frac{\alpha v^T - t}{(k+2)}\right) \\
&= \sum \frac{e_i}{p_i}\lambda - \left(\frac{\alpha v^T - t}{(k+2)}\right) \\
&= \sum \frac{e_i}{p_i}\lambda - \left(\frac{\alpha v^T - p_i\omega_i - \delta^*}{(k+2)}\right)
\end{aligned}$$

$$\therefore \lambda = \frac{\alpha v^T - p_i\omega_i - \delta^*}{(k+2)(\sum \frac{e_i}{p_i} - 1)}. \tag{3.18}$$

Equation 3.18 represents a set of equations for different αv^T and $p_i\omega_i$. Therefore we may find many λ values that are suitable.

(Continuing Example 4). By combining Equations 3.11 − 3.14 with Equation 3.16 and replacing $\theta = \frac{1}{3}\lambda$ because $U_W = \frac{1}{3}$, we get the following set of equations which later are replaced by Equation 3.17 to deduce λ and θ.

$$\begin{aligned}
&\{\lambda_1 = \frac{3}{4}(t - \alpha v^T)\} && \text{(using Equation 3.11)}, \\
&\{\lambda_2 = \frac{1}{2}(t - \alpha v^T)\} && \text{(using Equation 3.12)}, \\
&\{\lambda_3 = \frac{3}{8}(t - \alpha v^T)\} && \text{(using Equation 3.13)}, \\
&\{\lambda_4 = \frac{3}{10}(t - \alpha v^T)\} && \text{(using Equation 3.14)}.
\end{aligned}$$

Using the possible values of ω_i, and α until $\text{LCM}(p_1, \ldots, p_n, \lambda) + 2(\lambda - \theta)$, we will get a set of ($\lambda$, θ). Using these assignments we can calculate δ^* using Algorithms 1 and 2,

and check for the validity of the resource supply with respect to the workload demand. Thus the algorithm based on our proposed resource supply calculation model finds a list of resource supplies that allow the worst-case delay to be less than or equal to the value δ^*. For the example, the diagonal, horizontal, and vertical lines intersect at $\lambda = 3$ and $\theta = 1$, which is a valid resource supply.

Algorithm 3 Finding resource supply

Input: number of tasks n, tasks execution time $(e_1, \ldots, e_n)$, tasks periods $(p_1, \ldots, p_n)$, tolerable delay δ^*, list $(\alpha_1, \ldots, \alpha_n)$, list $(\omega_1, \ldots, \omega_n)$

Output: list of valid resource supply $L(R)$

1: $U_W \leftarrow \sum_{i=1}^{n} \frac{e_i}{p_i}$, supplyList $L(R) = \{\phi\}, m = 1$
/* looping steps on the demand of each task (i.e., horizontal lines) */
2: **for** $i = 1 \ldots n$ **do**
3: **for** $m = 1 \ldots \lfloor \frac{\text{LCM}(p_1,\ldots,p_n)+\delta^*}{p_i} \rfloor$ **do**
/* looping steps on the duration of each task (i.e., vertical lines)*/
4: **for** $i = 1 \ldots n$ **do**
5: **for** $\omega_i = 1 \ldots \lfloor \frac{\text{LCM}(p_1,\ldots,p_n)+\delta^*}{p_i} \rfloor$ **do**
/* looping steps on the supply (i.e., diagonal lines) */
6: **for** $k = 1 \ldots \text{LCM}(p_1, \ldots, p_n) + \delta^*$ **do**
/* using Equation 3.18 */
7: $\lambda = \frac{\alpha v^T - p_i \omega_i - \delta^*}{(k+2)(\sum \frac{e_i}{p_i} - 1)}$
8: calculate $\theta = U_W \lambda$
9: calculate δ using Algorithm 1, 2
10: **if** $\delta^* \geq \delta$ **then**
11: $L(R) = L(R) \cup \{(\lambda, \theta)\}$
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: **end for**
17: **end for**
18: traverse list $L(R)$ for a suitable (λ, θ)

To find an efficient resource supply, the workflow is as follows for a given set of tasks: the utilization of the resource supply is kept the same as the workload utilization as transient overload occurs when $U_W = U_R$. The algorithm based on the supply calculation model searches for resource supplies that have recovery points at $t = \delta^* + \omega_i p_i$ (i.e., time intervals

that denote the recovery of overloads) and calculates a fitting resource period if one exists. However, a resource supply that contains the recovery point may still be unusable, because the supply might have a worse δ^* at a later or earlier part of the *sbf*. Therefore, the algorithm searches for different supplies and checks them based on the method described in Section 3.3.2. Figure 3.4 shows multiple resource supplies for the Example 1 workload with $\delta^* = 1$; e.g., $\lambda = 1.5$ (blue color line in Figure 3.4) and $\lambda = 1.12$ (red color line in Figure 3.4). Our framework then selects from these candidates the one that has the largest λ to reduce the number of context switches.

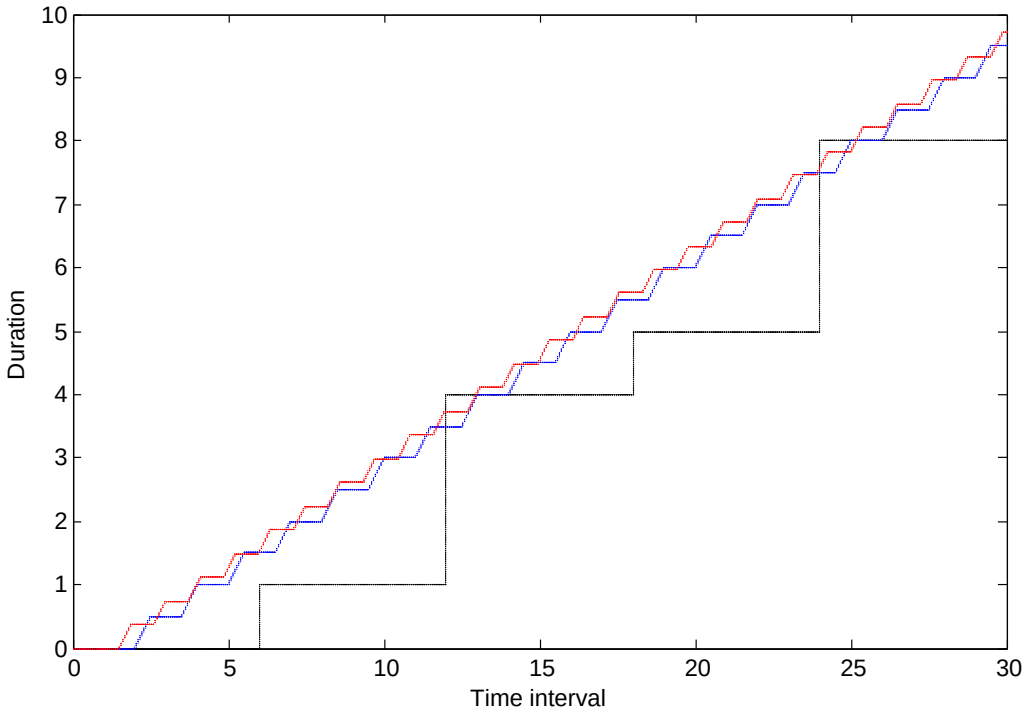


Figure 3.8: Two of the candidate solutions for Example 1 with input delay *one*

Theorem 4 *Given that dbf(t) is schedulable, the sbf(t) for $\lambda = \theta$ always satisfies the dbf(t) with a given tolerable delay δ^*.*

Proof The sbf(t) is a line with slope 1 for $\lambda = \theta$, which characterizes the maximum possible resource supply in a single processor system. Assume that there exists a case when the

assignment of $\lambda = 1$ and $\theta = 1$ do not satisfy dbf(t) with a given tolerable delay δ^*. Hence, the utilization of resource supply is $U_R = 1$ and $U_R < U_W$. This yields, $U_W > 1$ which is violates the assumption of a single processor system. Therefore, the sbf(t) for $\lambda = 1$ and $\theta = 1$ always satisfies the dbf(t) with a given tolerable delay δ^*.

Lemma 1 *Using the proposed resource supply calculation model (Eq. 3.10, 3.16, and 3.17), Algorithm 3 can find a valid resource supply, if there exists one.*

Proof In the proposed resource supply calculation model or Line 7 of Algorithm 3, the denominator of Equation 3.18 is only increased by the value $k \in \mathbb{N}$, because the utilization of the workload is constant and k is independent on the numerator of the equation. Therefore, the supply calculation proceeds by decreasing λ because of increasing k, which automatically decreases θ as $U_W = U_R$. The value k is incremented up to the search interval, LCM$(p_1, \ldots, p_n) + \delta^*$. With a small enough λ and θ, the *sbf* will become similar to a line as discussed in Theorem 4 such that the δ^* constraint is preserved. The proposed model will always find a valid resource supply, if one exists for the given inputs. In Algorithm 3, k is incremented in integers rather than real numbers. However, Algorithm 3 ensures that a suitable resource supply is found because k is incremented up to the search interval. Therefore, Algorithm 3 does not necessarily find an optimal resource supply but finds at least a suitable resource supply.

Corollary 1 *The algorithm based on the proposed resource supply calculation model to find a valid resource supply is sound.*

Proof The proposed algorithm to find valid resource supplies is sound, because they are checked for validity using supply and demand bound functions as defined in Line 9 and 10 of Algorithm 3.

Feasibility analysis refers to whether a task set is feasible under a resource model for a given scheduling policy. Feasibility ensures that there exists a resource supply that can satisfy the requirements of the tasks. Theorem 5 denotes that the output of the proposed resource supply calculation algorithm is feasible under a periodic resource model for workloads with bounded overloads.

Theorem 5 *Given a system workload $W = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with tolerable δ^*, the output of the proposed algorithm is feasible if and only if there exists a resource supply such that the resource demand in any time interval exceeds the resource supply during the same time interval for no more than δ^* consecutive units of time.*

Proof Theorem 4 establishes the proof of Theorem 5, because there always exist a resource supply such that maximum delay is bounded by δ^*.

The following observations are used to derive the complexity of the search algorithm that finds a valid resource supply.

- Vertical lines pass through multiples of tasks periods up to $\text{LCM}(p_1, \ldots, p_n) + \delta^*$.
- Horizontal lines pass through the summation of tasks execution time up to $\text{LCM}(p_1, \ldots, p_n) + \delta^*$.
- Diagonal lines depend on k which increases up to $\text{LCM}(p_1, \ldots, p_n) + \delta^*$. The value k is an integer value as defined in the $\text{sbf}(t)$ equation.

The complexity of the proposed algorithm as shown in Algorithm 3 to find suitable resource supplies with respect to the number of tasks and the worst-case delay is $O(n^3(\text{LCM}(p_1, \ldots, p_n) + \delta^*)^5)$. In Algorithm 3, the upper bound of **for** loop in Line 2 is $O(n)$, in Line 3 is $O(\text{LCM}(p_1, \ldots, p_n) + \delta^*)$, in Line 4 is $O(n)$, in Line 5 is $O(\text{LCM}(p_1, \ldots, p_n) + \delta^*)$, and in Line 6 is $O(\text{LCM}(p_1, \ldots, p_n) + \delta^*)$. The upper bound in Line 7 is $O(n(\text{LCM}(p_1, \ldots, p_n) + \delta^*)^2)$ because of using Algorithm 1 and 2.

3.5 Experimental Analysis of A Control System

We developed a MATLAB-based application called *sbFinder* based on the results shown in this work. To demonstrate the utility of our technique for designing the resource supply in the context of a control system, we consider the problem of simultaneously stabilizing two plants with a single computational resource. The first plant, denoted by Ω_1, is an inverted pendulum mounted on a cart, and is given by the following linearized dynamical system [62]:

$$\dot{x}(t) = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.1818 & 2.6727 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -0.4545 & 31.1818 & 0 \end{bmatrix}}_{A} x(t) + \underbrace{\begin{bmatrix} 0 \\ 1.8182 \\ 0 \\ 4.5455 \end{bmatrix}}_{B} u(t). \tag{3.19}$$

The state feedback controller for this system is designed (under nominal delay-free conditions) as

$$u(t) = \underbrace{\begin{bmatrix} 2.6939 & 3.5571 & -23.5576 & -4.462 \end{bmatrix}}_{K} x(t).$$

The plant is sampled every $p_1 = 0.02$ seconds. Using Theorem 1, the maximum value of η is 0.0732. The worst-case delay in this system then is $\delta_1 = \eta - 2p_1 = 0.032$ seconds.

The second plant, denoted Ω_2, is a chemical distillation column modeled as a linearized system with 8 states and 4 inputs; the exact model can be found in [79], and we omit the details here to save space. The nominal state feedback controller is designed to place the closed loop poles at $-1, -1.1, -1.2, -1.3, -1.4, -1.5, -1.6, -1.7$, and this produces the control gain K. The plant is sampled every $p_2 = 0.15$ seconds. Once again applying Theorem 1, the maximum value of η is 0.3340, and thus the maximum delay for obtaining computational resources that can be tolerated by this control system is $\delta_2 = \eta - 2p_2 = 0.034$ seconds.

The computation of both control inputs is done by a single processor. Thus, to maintain stability, the processor must guarantee that the worst-case delay for calculating any control input is $\delta^* = \min(\delta_1, \delta_2) = 0.032$ seconds. We take our unit of time to be 0.01 seconds, and assume that the processor can compute the control input for either plant within this length of time. Thus, in terms of this time-unit, the period and execution time for the first plant are $p_1 = 2$ and $e_1 = 1$, respectively, and the period and execution time for the second plant are $p_2 = 15$ and $e_2 = 1$, respectively. The maximum tolerable delay δ^* for both systems is $\delta^* = 3$.

With this workload and delay specifications, our proposed model produces a list of 138 solutions that are valid. The list contains 132 different assignments of resource supply that have the minimum utilization. Traversing the list for the maximum resource period yields $\lambda = 2.5$, $\theta = 1.4167$, $\delta^* = 1.8$, and utilization = 0.5667.

3.6 Related Work

Shin and Lee present schedulability analysis [76] based on the *sbf* [77] and the *dbf* [12, 58] for the compositional real-time scheduling framework. This framework can be used to establish global (system level) timing properties by composing individual timing properties. The authors present schedulability conditions for the standard Liu and Layland periodic resource task model and propose a periodic resource model under EDF and RM scheduling that allows the composition of multiple timing requirements into a single timing requirement. In the related work [76], the authors analyze schedulability of a bounded delay resource partition model in terms of the *sbf* and the *dbf*. Deducing a single timing requirement out of multiple timing properties creates some new challenges which have been solved in a number of subsequent works [74, 77]. As a variant of the system model

from [76], Shin *et al.* also propose algorithms that define optimal interfaces for the subsystems which may share resources. Integrating subsystems into a system having optimal interfaces motivates the development of adaptive and reconfigurable systems.

An important aspect of using the *sbf* and the *dbf* is the optimized use of the resources. Easwaran *et al.* [35] show that selecting a particular resource model that minimizes the collective resource requirements facilitate systems to change components on the fly. Lee *et al.* propose an optimization framework for maximizing the QoS under K random failures on schedulability. The authors use Langarian duality [57] for distributed computation that results in optimal solutions.

Mok, Feng, and Chen [64] introduce the concept of a supply function to measure the minimum amount of computing resources provided to a static partition. Wandler and Thiele [83] propose the concept of interface-based design that uses real-time calculus and modular performance analysis to compute the supply curves. Lipari and Bini [59] derive a set of supply functions that are feasible to schedule an application. Later Bini et al. propose an optimization framework [15] to select the minimum bandwidth required of a EDF task set. These works use the fraction of computing resource supplied by the processor and the initial delay of the resource to ensure that minimum bandwidth is given to the workload demand, but do not consider delays due to the transient overloads that the tasks may tolerate and the existence of a periodic resource model (a special class of supply functions) that Shin and Lee [75] introduce.

Devi and Anderson [33] introduce tardiness bounds under global EDF scheduling on a multiprocessor for soft real-time systems. However, the tardiness bounds are not derived in terms of the supply and demand bound functions for a compositional framework as discussed in [76]. Kumar et al. [55] propose a model to compute the resource with a given delay bound from a stream of jobs characterized by an input arrival trace. However, the arrival jobs are not specified with a certain delay bound that we assume in this work to characterize application-specific tasks that can tolerate overloads or delays. Moreover, the delay is calculated in terms of time rather than the time intervals that we follow because we attempt to calculate the delay from the supply and demand bound as defined in [76, 77] which are functions of time intervals. Buttazzo et al. [40] introduce elastic scheduling that allows to vary the period of a task based on its flexibility specified in the task model. This model inherently allows to tolerate overloads to a certain amount but does not use the concept of supply and demand bound functions we use to compute an efficient periodic resource model towards building a compositional system. Hence our work is in-line with the other work in the literature but differs in finding an efficient resource model due to the time-interval analysis of supply and demand bound functions for systems that can tolerate bounded transient overloads.

3.7 Discussion

Compositionality. A compositional framework can combine different specifications and build up a schedule that satisfies the workload demands. The system calculates the most suitable resource supply for each of the specification's demands. Each resource supply turns into the workload demand while using the compositional framework. Earlier work on the periodic resource model [76] assumes no overloads or delays in the workload. Using our work to extend [76], it is possible to deal with workloads with bounded transient overloads. Definition 6 describes the composition method in the context of transient overloads.

Definition 6 (Composition method) *Given a number of scheduling models $M_1 \dots M_n$, a compositional scheduling model $M_P(W_P, R_P, EDF)$ can be derived by mapping the resource model of a child scheduling model $R_i(\lambda_i, \theta_i)$ to its periodic task $\tau_i(p_i, e_i)$ and including any new tasks (τ_i') such that $W_P = \{\tau_1(\lambda_1, \theta_1), \dots, \tau_n(\lambda_n, \theta_n)\} \cup \{\tau_i'(p_i', e_i')\}$.*

Relation between overload duration and overload severity. In this dissertation, the *severity* of the overload is defined as $dbf(t_o) - sbf(t_o)$. The duration and the severity of an overload are related.

1. A large overload severity implies a long overload duration and thus a long delay.
2. A long overload duration (and thus a long delay) does not necessarily imply a large overhead severity.

These two observations originate from the fact that the best possible resource supply is $R(1, 1)$ in which the system receives all resources in a uniprocessor system. In such a scenario, the slope of the *sbf* is 1. Thus, the overload delay is always at least equal to the overload severity. Hence, a large overload severity implies a long overload duration.

On the other hand, the worst possible resource supply is $R(x, 1)$ where $x \to 0$. In this scenario, the slope of the *sbf* is close to 0. Thus, even a small overload severity can result in a long overload duration.

Multiprocessor systems. The way to deal with multiprocessor systems is to use partitioning algorithms [13]. EDF is not guaranteed to be optimal for multiprocessor systems, although it is optimal for uniprocessor systems. Therefore, prior to running the EDF

scheduling policy for uniprocessor systems, a partitioning algorithm can distribute the tasks statically to different processors.

Table 3.1 presents a comparative analysis between two types of multiprocessor scheduling: partitioned and global. From the analysis, it is certain that partitioned scheduling is a good choice for multiprocessor systems. Our uniprocessor analysis can be used upon a multiprocessor platform after partitioning a set of tasks into processors using an algorithm [13].

Table 3.1: Partitioned Scheduling vs Global Scheduling

Aspect	**Partitioned scheduling**	**Global scheduling**
Schedulability	The total utilization is at most m upon a m-processor platform in the hard real-time domain after partitioning tasks into processors. Partitioning overhead exists to assign tasks to processors.	Global scheduling (i.e., G-EDF) ensures tardiness bounds for soft real-time applications as long as the total system utilization is at most m [14]. Global scheduling tests are too pessimistic [14].
Implementation complexity	Low implementation complexity compared to global scheduling. Heavy utilization tasks (utilization higher than 0.5) strongly affect task partitioning heuristics [14].	Task partitioning complexity does not exist. Global schedulers, specially those are optimal, are difficult to implement in practice.
Predictability	Highly predictable if new tasks are added into the system. May suffer more delays to execute newly admitted tasks in the assigned processor than another processor.	Newly admitted tasks can run on any of the processors which is available based on priorities. Less predictable on the migrations of dynamically added tasks.
Isolation	High isolation because tasks are partitioned into processors and do not migrate.	Cluster disciplines can provide isolation and also provide migration. Isolation is limited due to allowing migration.
OS overhead	OS scheduling and release overhead is low [14]. Overhead due to task partitioning under heavy utilization tasks is high.	Scheduling and release overhead for clustered disciplines is similar to partitioned scheme [14]. OS scheduling and release overhead is high due to the cost of frequent cache line migrations and heavy bouncing [14].
Cache overhead	For light tasks (utilization between .001 and .1), weighted schedulability is better than global disciplines with respect to cache overhead [14]. For heavy tasks, weighted schedulability is worse than global disciplines with respect to cache overhead [14].	Cache overhead is low with respect to the ratio of schedulable task sets for global schemes under heavy tasks [14]. Migration may cause cache misses and thus extra delays [14].
Resource sharing	Less synchronization required compared to global schemes when sharing resources. In case of having a high-available resource, schedulability is very poor unless task utilizations are high [18].	In case of having a high-available resource, schedulability is better than partitioned scheme [18]. Synchronization overhead and errors may become a bottleneck and synchronization maintenance may become complex.

Chapter 4

Scheduling of Multi-Mode Communication

A key problem in designing multi-mode real-time systems is the generation of schedules to reduce the complexities of transforming the model semantics to code. Moreover, distributed multi-mode applications are prone to suffer from delays incurred during mode changes. We therefore aim to generate communication schedules that have low average mode-change delay for multi-mode real-time distributed applications.

This dissertation discusses the use of optimization constraints associated with timing requirements to generate state-based schedules for multi-mode communication systems, and illustrates the workflow for generating schedules from specifications through a real-time video monitoring case-study. Experiments in the case-study demonstrate that schedules generated using the proposed method reduce the average mode-change delay in relation to a randomized algorithm and the well-known EDF scheduling algorithm.

Current trends in distributed systems push the boundaries of existing real-time networks. Increased number and complexity of distributed devices and integration of multiple real-time domains with traditional computers in a single network are quickly making legacy fieldbuses obsolete due to their limited bandwidth and incompatible protocols.

Increasing interest in RTE, for example, provides strong evidence of this trend [32]. In recent years, both industry and academia have reported experimental evidence for hard real-time communication on top of Ethernet infrastructure [4, 3]. A common characteristic among the proposed solutions is the use of enhanced devices with specific modules for TDMA arbitration. To provide real-time guarantees, TDMA networks require careful planning of time-critical communication tasks, which must be scheduled and verified in

advance. In practice, static TDMA schedules are complex to design, and lead to inefficient bandwidth utilization, since they reserve time slots to handle the worst case, even though it rarely occurs.

Generating real-time schedules for efficient utilization of the available bandwidth, with data dependencies and conditional execution is a challenging and relevant problem for next-generation distributed systems. This work proposes a novel methodology for generating state-based communication schedules from a high-level specification of the distributed components, moving the complexity of schedule design away from the developer. Starting from a state machine description of the distributed tasks, the proposed workflow generates abstract representations of state-based schedules for any feasible system, which can then be mapped to executable entities. This chapter walks through the individual steps using an example case-study based on real-time video streaming over an Ethernet network, which is a recurrent application in the automotive domain. The results demonstrate that the generated schedules meet the real-time guarantees, while minimizing the average delay to switch from one operational mode to another with respect to a set of valid schedules generated using both EDF and random mapping of messages to time slots. The analysis from the particular case-study can be easily generalized to any network based on TDMA.

The remainder of the chapter is structured as follows: Section 4.1 describes formal definitions of state-based scheduling, and introduces relevant terminology for the schedule generation workflow. Section 4.2 illustrates the steps for the generation of state-based schedules from component-level specifications using a video streaming case-study. Section 4.3 provides a brief overview of related work. Section 4.4 briefly discusses the advantages of state-based schedules in transmitting mixed critical traffic consisting of best-effort and real-time.

4.1 Formal Definitions

A state-based schedule is an abstract representation of communication systems based on TDMA with on-the-fly decisions. A TDMA communication system consists of a set of stations that exchange messages through a broadcast network. The concepts of state-based schedules are discussed in Chapter 2.

TDMA schedules divide time into non-overlapped slots and rounds. Each slot is defined by a start point and a slot length. A scheduler performs a message-to-slot mapping based on the timing requirements for the system. A *linear schedule* maps a slot to either only one message or leaves it empty. A communication round refers to a sequence of messages that repeats endlessly as the system executes.

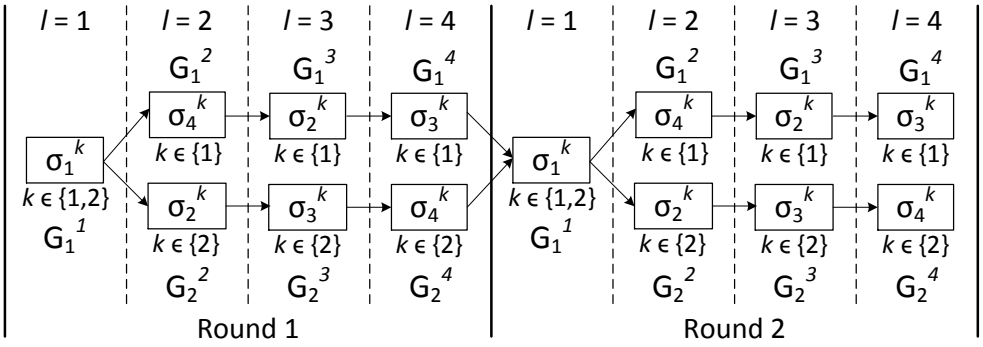


Figure 4.1: Overlapped message assignments in slot 1 form group G_1^1

An *operational mode* describes a system state and specifies the messages that the system needs to communicate when running in the associated mode. Each operational mode is associated to a predefined linear schedule that meets the timing requirements for the messages. State-based schedules encode a set of linear schedules associated to different modes using a global-time base. As a result, different messages from different modes can be mapped to the same slot, but only one mode can be active at any slot during runtime. The schedules can encode guarded transitions that allow the system to change from one mode to another between consecutive slots within a single communication round.

Let us consider a set of messages $\{\sigma_m\}$ where $m \in \mathbb{N}$. Each message σ_m has an associated period π_m, transmission time e_m, and deadline equal to period, all represented as entire multiples of an atomic time unit γ. Let us also consider a set V representing the operational modes, each one associated to a linear schedule. Considering a slot length of γ, the communication round will have LCM$\{\pi_m\}$ slots, where LCM$\{\pi_m\}$ is the Least-Common-Multiple of periods of all the messages, i.e., the *hyperperiod* of the system. We say that the state-based schedule has an *overlap* in slot $l = 1, \ldots,$ LCM$\{\pi_m\}$ when different modes map the same message to that slot. Overlapped messages at slot l can be combined into a *group* G_x^l where $1 \leq x \leq |V|$. A state-based schedule is a tree-like structure where each branch represents a *transition* from a group in slot l to one or more groups in slot $l + 1$.

A valid state-based schedule is one that meets the timing requirements for each linear schedule and the possible transitions between them. A particular system can have multiple valid schedules, and then different number of overlaps and groups. Figure 4.1 and Figure 4.2 show an example of two different schedules for a system with two modes and LCM$\{\pi_m\} = 4$. For illustrative purposes, let us assume that both schedules are valid for the particular system. The notation σ_i^k indicates that mode k maps σ_i to the corresponding slot. The

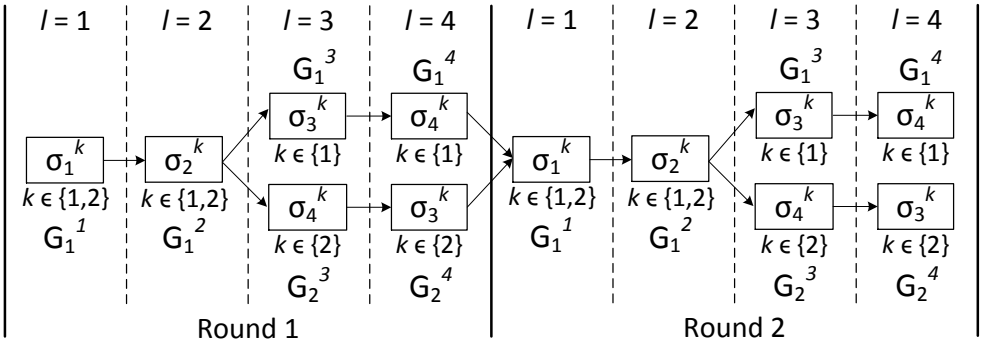


Figure 4.2: An alternative schedule with an additional overlap in slot 2

schedule in Figure 4.1 has an overlap for σ_1 in $l = 1$, which can be represented in group G_1^1. Figure 4.2 shows an alternative schedule with an additional overlap for σ_2 in slot 2, which is represented in G_1^2.

A mode-change in a state-based schedule is a timed event that triggers a guarded transition to move from an old mode $v_s \in G_x^l$ to a new mode v_d in the next slot at the cost of a mode-change delay $\delta(v_s, v_d)$ defined as:

$$\delta(v_s, v_d) = \begin{cases} 0 & \text{if } v_d \in G_x^l \\ \\ \text{LCM}\{\pi_m\} - l & \text{otherwise} \end{cases} \tag{4.1}$$

where $\text{LCM}\{\pi_m\} - l$ represents the time until the next communication round starts. The mode-change can occur immediately if a transition exists between mode v_s and v_d. If not, the mode-change occurs when a new communication round starts. Reducing the number of transitions decreases the number of groups which lowers the average mode-change delay (Theorem 6).

Theorem 6 *Given a set of valid state-based schedules for a particular system, the schedule with the fewest number of groups in a communication round will minimize the average mode-change delay $\overline{\delta}$ for a uniform probability of mode-changes.*

Proof Our proof is based on contradiction. Let us consider two valid schedules S and S' for a particular system, with S' having fewer groups per communication round. Let us assume that $\overline{\delta}_{S'} > \overline{\delta}_S$. The value $|V - G_x^l|$ represents the number of modes that do not belong to G_x^l. Since a maximum number of $|G_x^l|$ modes can request a mode-change to the

modes that do not belong to G_x^l, the average mode-change delay $\overline{\Delta}$ for all slots for all possible groups at runtime is:

Since the system executes all the transitions with uniform probabilities, then the average mode-change delay $\overline{\Delta}$ for all slots for all possible groups at runtime is:

$$\overline{\Delta} = \frac{\sum_l \sum_x (\text{LCM}\{\pi_m\} - l)(|V - G_x^l|)|G_x^l|}{\Gamma|\bigcup_l(\bigcup_x G_x^l)|}.$$

The average delay in the state-based schedule $\overline{\delta}_{S'}$ for a uniform possibility of mode changes is less than $\overline{\delta}_S$, because S' has fewer groups than that in S and therefore the value $|V - G_x^l|$ is less than that of in S. This contradicts that the average mode-change delay in S is lower than the average mode-change delay in S'.

This following section describes a workflow for schedule generation that uses the concept of groups to address the problem of message-to-slot mapping for state-based schedules, such that the resulting valid schedule minimizes the average mode-change delay.

4.2 Schedule Generation Workflow

This section walks through an example case-study to illustrate the necessary steps to generate state-based schedules from a component-level description.

4.2.1 Workflow Overview

Figure 4.3 illustrates the steps for the proposed workflow to generate state-based schedules. The designer specifies a component-level description consisting of state machine descriptions and timing requirements for each task generating messages. Given that the system is correctly described and feasible, the proposed workflow will generate a valid schedule that minimizes the average mode-change delay.

The first step in the workflow is an analysis that combines the states of the individual components and obtain the operational modes of the system. This step also considers a feasibility test to verify that it is possible to meet the timing requirements for all the operational modes. If any of the operational mode is unfeasible (e.g., due to bandwidth

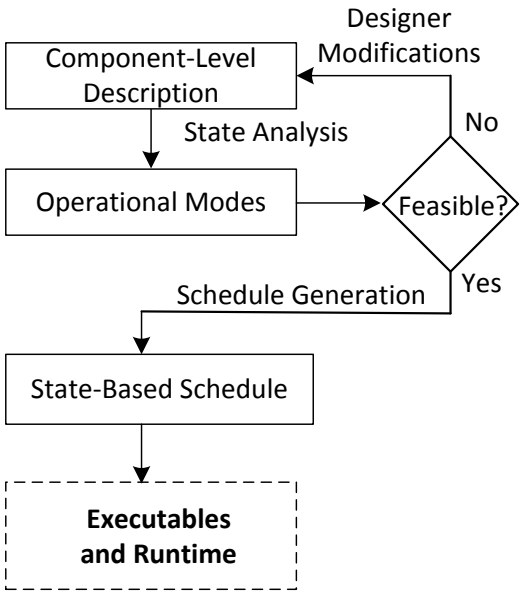


Figure 4.3: Proposed workflow to generate state-based schedules

limitations), then the designer must modify the system specifications. For feasible systems, the next step is the generation of a valid schedule that minimizes the average mode-change delay.

The previous steps generate an abstract representation describing the message-to-slot mapping for the system. These descriptions can then be translated into a programming language to generate executable abstractions. Previous work illustrates this process by mapping schedules designed by hand for simple applications to the Network Code framework, which offers a domain-specific programming language with conditional branching capabilities and a powerful hardware environment for real-time communication over Ethernet [23].

The rest of this section provides details for each step in the workflow using an example case-study based on the demonstration setup described in [23]. The application considers the real-time streaming of multiple video sources on top of Ethernet, which can change the resolution according to specific operational conditions for efficient utilization of the bandwidth.

Figure 4.4: Environment for the example application

4.2.2 Application Example and Assumptions

Let us consider an embedded video monitoring system for mining trucks. Drivers need some kind of monitoring system for increased security when sharing the road with smaller vehicles and people because of the dimensions of these trucks (typically over 7 mt. high). The system uses four video cameras, each one transmitting a stream of the surroundings of each wheel to displays located in the driver's cabin. All devices connect through a real-time capable Ethernet network operating at 1[Gbit/s] [23].

The cameras operate in two states that differ in the number of Frame-Per-Second (FPS) : Standard Quality (SQ) and High Quality (HQ). The cameras operate in SQ by default. Each wheel includes a sensor that detects proximity to surrounding objects. When a sensor activates, the cameras switch to HQ. An additional condition is that the sensors activate according to the movement of the truck: front sensors only activate if the truck is moving forward, and rear sensors only activate if the truck is moving backwards.

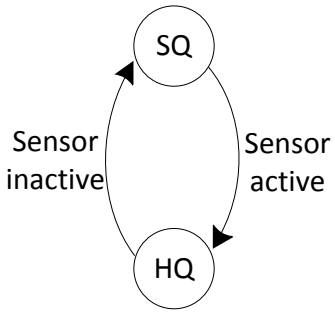
For the analysis we only consider the scheduling of the video streams, and assume that all distributed components are synchronized to a global clock reference. In practice, the

model specification must also consider the scheduling of periodical sensor readings and synchronization messages, which require much less bandwidth than the video data [23].

4.2.3 Component-Level Description

Figure 4.5 shows a state machine representation and transition table for the camera attached to the front right wheel. The inputs Front Right (FR), Front Left (FL), Rear Right (RR), and Rear Left (RL) represent the status of the sensors attached to each wheel; and s_k and s_{k+1} represent the current and next state, respectively. The designer must provide a similar representation for each camera, together with the timing requirements for each state (addressed in Section 4.2.4).

Alternatively, the designer can use high-level languages such as Architecture Analysis and Design Language (AADL) [56] or Unified Modelling Language (UML) profile for Modelling and Analysis of Real-Time and Embedded Systems (MARTE) [81] to specify the system behavior, and use a parser to extract the transition tables and timing specifications.



(a) State Machine

FCR, FCL, RCR, RCL	s_k	s_{k+1}
1, 0, 0, 0	X	HQ
1, 1, 0, 0	X	HQ
others	X	SQ

(b) Transition Table

Figure 4.5: Representation of the states for each camera

4.2.4 State Analysis

The algorithm presented in [47] allows us to compute the operational modes of the system as the cross products of the state machines for the individual components. In the case-

Table 4.1: Timing requirements for different video qualities

	Nominal [ms]		Normalized	
FPS	Comp. Time	Period	Comp. Time	Period
15	7600	66700	1	8
30	7600	33350	1	4
60	7600	16720	1	2

study, the condition that front and rear cameras cannot operate at HQ at the same time leaves only seven possible operational modes out of the sixteen possible.

Table 5.2 shows the parameters of interest to perform feasibility analysis for the case-study. These parameters must be encoded together with the transition tables. The nominal transmission time represents the time required to transmit a single video frame of 640x480 pixels, with a pixel-width of 32 bits, over a 1[Gbit/s] link. Since the maximum payload for a standard Ethernet frame is 1500 bytes, video frames are transmitted as a sequence of Ethernet frames. The reported time accounts for the overhead related to Ethernet headers and Inter-Frame Gap (IFG). For simplicity, we omit the propagation latency and additional processing in the path between cameras and displays, which will depend on the physical configuration of the network. In practice, designers must provide a worst-case value for these parameters and consider them in the total transmission time for each message. The table also shows the periods associated to different FPS. Considering an atomic unit for the schedule equal to the minimum transmission time for a message, we normalize the timing specifications to this time unit, and floor the normalized period. This processing over-estimates the actual requirements, but allows us to represent all the timing as multiple integers of the time unit.

For the system to be feasible, the total channel utilization for each operational mode cannot be greater than the available bandwidth for scheduled traffic. This is:

$$U(v_k) = \sum_{\sigma_m \in v_k} \frac{c_m}{\pi_m} \leq \frac{B}{L} \tag{4.2}$$

where B is the bandwidth assigned to scheduled messages, and L is the link capacity, with $B \leq L$.

Table 4.2 summarizes the utilization test for all operational mode when setting the SQ mode to 15[FPS], and the HQ mode to either 30[FPS] or 60[FPS]. Considering that all link capacity is available for the video streams, the feasibility test will be $U(v_k) \leq 1$. We see if the HQ mode is set to 60[FPS], there are two modes that fail the feasibility test (bold

Table 4.2: Feasible modes with messages specifications

ID	Mode	Utilization (SQ=15[FPS])	
	(FCR, FCL, RCR, RCL)	HQ=30[FPS]	HQ= 60[FPS]
1	(SQ,SQ,SQ,SQ)	0.5	0.5
2	(SQ,SQ,SQ,HQ)	0.625	0.875
3	(SQ,SQ,HQ,SQ)	0.625	0.875
4	(SQ,SQ,HQ,HQ)	0.75	**1.25**
5	(SQ,HQ,SQ,SQ)	0.625	0.875
6	(HQ,SQ,SQ,SQ)	0.625	0.875
7	(HQ,HQ,SQ,SQ)	0.75	**1.25**

numbers), and then the designer must either need to change the specifications or modify the system. In this case, reducing the quality of the HQ mode to 30[FPS] makes the system feasible.

4.2.5 Schedule Generation

To find an optimal state-based schedule with respect to minimizing the number of groups, this work uses Integer Linear Programming (ILP) to find optimal assignments of messages to slots. Minimizing the number of groups provides the optimal assignments of messages to slots, which are based on the constraints on timing requirements of messages of each mode and the characteristics of state-based schedules.

In addition to the parameters of the system model, the ILP model uses some variables to find the optimal assignments of messages to slots. The variable α_m^k represents the number of instances for every σ_m^k until the hyperperiod such that $\alpha_m^k = \frac{|\Gamma^k|}{\pi_m^k}$ where $\Gamma^k = \{1, \ldots, \text{LCM}\{\pi_m^k\}\}$. The variable x_{ml}^k represents the usage of a time slot for $m \in \mathbb{N}$, $l \in \Gamma^k$, and $k \in V$. Therefore,

$$x_{ml}^k = \begin{cases} 1 & \text{if message } \sigma_m^k \text{ is allocated to slot } l \text{ in mode } k \\ 0 & \text{otherwise.} \end{cases}$$

The number of groups in a state-based schedule will increase if different messages of different modes are allocated to the same slot and the previous slots. The variable s_{ml}^k is

used to reduce the possibilities of such assignments of messages to slots. The value of s^k_{ml} is set to 1 if a message σ^k_m of mode k is allocated to a slot l, and there exists the same σ_m message of any other modes that is allocated to the same slot l. Therefore,

$$s^k_{ml} = \begin{cases} 1 & \text{if } x^k_{ml} = 1 \wedge \\ & \exists x^v_{ul}, st : u \neq m \wedge v \neq k \wedge x^v_{ul} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The variable w^k_{ml} is used to determine whether different messages of other modes are allocated to the same slot l if a message σ_m of mode k is already allocated to slot l. The ILP model shown below minimizes the number of groups by calculating the assignments of required time slots to messages for each mode k in the Γ^k. Since the decrease in number of groups not only depend on allocating different messages of other modes in the current slot but also in the earlier time slots, the objective function assigns weights such that the overlaps can occur earlier than later.

$$\begin{aligned} \min \quad & \sum_{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma^k, \mathrm{k} \in \mathrm{V}} s^k_{ml} \times (\Gamma^k - l) + \sum_{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma^k, \mathrm{k} \in \mathrm{V}} x^k_{ml} \times l. \\ st. \quad & \mathbf{C}_1\{\forall \mathrm{m} \in \mathrm{N}, \mathrm{k} \in \mathrm{V}\} : \\ & \sum_{l=g\pi^k_m+1}^{g\pi^k_m+\pi^k_m} x^k_{ml} \geq c^k_m \ , g = 0, \ldots \alpha_m - 1; \\ & \mathbf{C}_2\{\forall l \in \Gamma^k, k \in \mathrm{V}\} : \\ & \sum x^k_{ml} \leq 1, \\ & \mathbf{C}_3\{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma^k, \\ & \mathrm{k} \in \mathrm{V}, \mathrm{u} \in \mathrm{N} - \{m\}, \mathrm{v} \in \mathrm{V} - \{k\}\} : \\ & w^k_{ml} \geq x^v_{ul}, \\ & \mathbf{C}_4\{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma^k, \mathrm{k} \in \mathrm{V}\} : \\ & s^k_{ml} \geq x^k_{ml} + w^k_{ml} - 1, \\ & \mathbf{C}_5\{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma^k, \mathrm{k} \in \mathrm{V}\} : \\ & s^k_{ml} \geq 0, \end{aligned}$$

In the ILP model, a number of constraints represent the characteristics of the optimized schedule. Constraint C_1 specifies that all messages at least get the computation units in

their periods. Constraint C_2 and C_5 are bounds to guarantee x^k_{ml} and s^k_{ml} being binary. Constraint C_3 and C_4 compute s^k_{ml}. These constraints manage to find optimal assignments of x^k_{ml} that the decomposition method (Definition 7) uses to construct a state-based schedule.

Definition 7 (Decomposition method) *Given all schedulable and reachable groups at any time slot, a decomposition method derives all groups in the next time slot from x^k_{ml} followed by a labelled transition (i.e., guard).*

Using the decomposition method as defined above, it is possible to generate an optimized schedule either based on provisioning empty slots to transmit best-effort traffic or allowing messages to execute more than their timing requirements. In the latter, the schedule has fewer groups and therefore faster average mode-change delay, but lower bandwidth for best-effort traffic. This dissertation presents the use of best-effort traffic to gain the full advantage of using state-based schedules. According to Definition 7, we define a method (Algorithm 4) to construct a state-based schedule from the values of x^k_{ml} based on provisioning empty slots. Using x^k_{ml}, the method finds all groups from $l = 1$ onwards, along the slots, up to slot $\left|\Gamma^k\right|$, where a group contains modes that either have overlapped messages or nothing (ϵ) to transmit. A group G initially contains all messages and uses the decomposition method to create groups G^k_{ml}. Figure 4.6 shows the state-based schedule for the case-study prioritizing best-effort traffic.

To demonstrate that the solution to the described optimization problem assigns the messages to slots efficiently, we use random assignments of x^k_{ml} that also meet the timing requirements of messages of each mode. Figure 4.7 shows the difference in the mode-change delays between the 10000 randomized schedules and the optimized schedule for a hyperperiod of 4, 8, 12, 16, 20, 24, 28, and 32 time slots using the case-study through changing the period of the SQ message. The average mode-change delay for the generated schedule using optimized x^k_{ml} is also significantly less (Figure 4.7) than the delay in the schedules generated using either randomized or EDF-based x^k_{ml}.

We implemented a set of open-source scripts for each step of the workflow. Users can download these scripts [2] and verify the results for the case-study included as an example, or start with new system specifications. The current version of the workflow scripts are based on Matlab and use the AMPL/GUROBI optimization problem solver.

Function 4 schedGen(k, l, m, x^k_{ml})

Input: All k modes, all l slots, all m messages, all x^k_{ml} assignments
Output: All G^k_{ml} groups

```
    /* find all the overlaps and form groups for each slot */
 1: for each slot l do
 2:   for each mode k do
 3:     if k is the first mode and x^k_ml = 1 then
 4:       form a new group G^k_ml
 5:     else
 6:       check if it is required to form a new group
 7:       set isRequired=true
 8:       for each group G^k_ml do
 9:         if mode k belongs to both group G^k_ml and group G^k_m(l-1) then
10:           set isRequired=false
11:           break;
12:         end if
13:       end for
14:       if isRequired==false then
15:         add mode k to the last group created
16:       else
17:         form a new group G^k_ml
18:       end if
19:     end if
20:   end for
21: end for
    /* create branches using the formed groups */
22: for each slot l do
23:   for each group G^k_ml do
24:     check membership of G^k_ml with all groups in the previous slot
25:     create a link between group G^k_ml and the overlapped group G^k_m(l-1)
26:   end for
27: end for
```

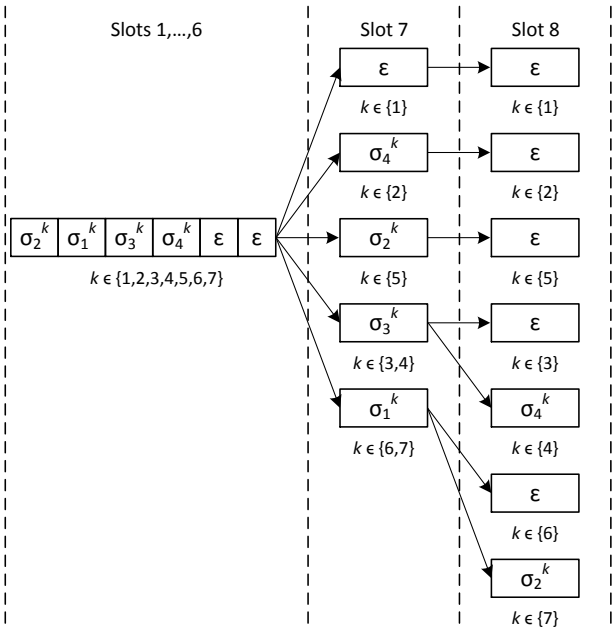


Figure 4.6: An optimized best-effort oriented schedule using decomposition

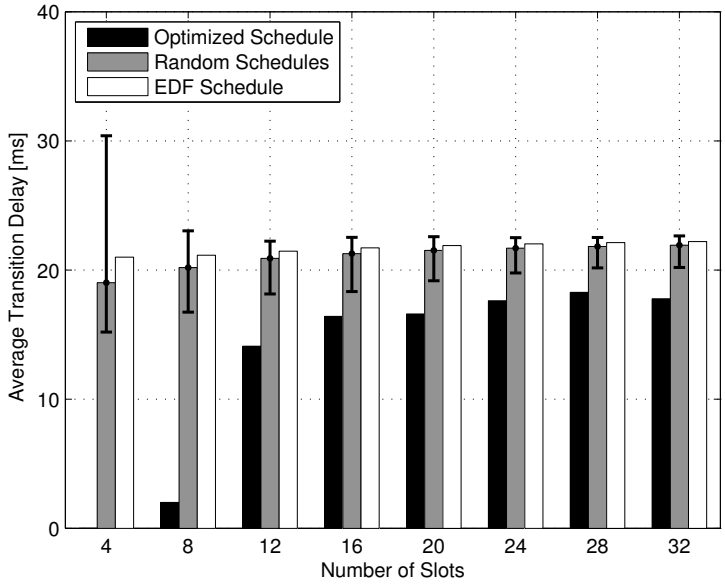


Figure 4.7: Average mode-change delay in generated schedules using randomized, EDF-based, and optimized x_{ml}^k assignments

4.3 Related Work

Traditional real-time networking protocols allow limited control to the applications over the communication behaviour at runtime. For example, developers must assign message priorities statically on a CAN bus to ensure that the priorities are unique [16]. FlexRay [39] follows a static TDMA approach with a specific slot for dynamic traffic at the end of each round. Stations must wait for that specific slot to transmit dynamic messages, and its timing is not guaranteed.

Some recent work explore the concepts of state-based schedules, but they lack any generation technique of schedules from high-level specifications. For example, in [86], the authors demonstrate the advantage of state-based scheduling for control systems, but using a small scale system, and the schedules are not necessarily optimized. Moreover, the work uses timed automata [5] to express the schedule using regular expressions. This work uses the notion of slots and communication rounds to reduce complexity for analysis and verification. In [69], the authors propose mechanisms to synthesize clocked graphs [70] with the Network Code framework, but avoids generation and optimization of state-based schedules.

4.4 Discussion

Complexity Characterization. The problem of minimizing the number of groups is an optimization problem to find optimal assignments x^k_{ml} of all possible m messages to all possible l slots in all possible k modes. The constraints of this problem include timing requirements of messages. For the sake of complexity characterization, the decision version (D) of the problem can be defined as, "Given all messages with timing requirements and a non-negative integer z, decide whether there exist assignments to all slots in all modes such that the number of groups is no greater than z". The problem D is in NP, because given x^k_{ml} assignments of all m messages to all l slots in all m modes with timing specifications and a non-negative integer z value, it is possible to check whether the input is valid or not by verifying timing requirements and calculating overlaps. Moreover, a well-known NP-complete problem, SAT can be used to show that D is NP-hard. An instance of SAT is identical to an instance of D and vice versa. The reduction can be done in polynomial time. Our problem of minimizing the number of groups requires finding a minimum z value and therefore the problem is harder than D.

Mixed-Criticality Communication. A mixed-criticality system must support the communication of both best-effort and time-critical data in the same network. Traditional

TDMA systems provide hard real-time guarantees at the cost of high penalization in the effective bandwidth utilization, since the schedules assign slots to specific stations even when they do not have new data to transmit. In general, static TDMA schedules are designed to handle the worst-case scenario, even though the worst-case rarely occur in practice. The state-based scheduling approach described in this chapter enables a better utilization of empty slots which can be used to communicate best-effort traffic, providing a more efficient utilization of the bandwidth with respect to static TDMA, while keeping the timing guarantees for real-time messages. The execution hardware for the Network Code framework provides mechanisms to leverage this property. The enhanced switches propagate best-effort frames from stations with traditional interfaces using any gap between real-time messages, without requiring any configuration step [23].

Chapter 5

Coscheduling of Computation and Communication

With the growing demand in real-time systems, safety (i.e., hard real-time) and non-safety (i.e., soft real-time) critical components are integrated together. However, mixing critical tasks with non-critical tasks in adaptive and reconfigurable embedded systems creates challenges to guarantee that the safety-critical units complete their tasks correctly on time.

Separation of concerns is required to decrease complexity of a system and also to guarantee safety. To leverage separation of concerns, isolation among computation components and communication medium is essential. We achieve separation of concerns using component interfaces for communication. Using a walk-through study, we also show the steps to generate communication schedules using component interfaces. The component interfaces provide safety through isolation and the generated communication schedules provide high performance because of efficient control on communication and low average mode-change delay.

This chapter presents generation of state-based schedules using component interfaces to provide separation of computation and communication. Specifications of component interfaces are computed using local information of tasks. Tasks execution inside each component remain isolated from communication because of generating its schedules using the derived component interfaces.

A demand bound function (dbf) and supply bound function (sbf) [76] analyze timing requirements of the workload (i.e., tasks) and the resource supply to guarantee that sufficient amount of resource is available to satisfy the demand. The dbf(t) refers to the

maximum resource demand during a time interval t by the tasks and the sbf(t) determines the minimum resource supply during t. For hard real-time systems, dbf(t) is no greater than the sbf(t). However, this condition does not need to hold [11] for systems that can tolerate bounded amount of delays.

The sufficient resource supply for a component with workload specifications converts into the demand to find a resource supply with additional workload. To satisfy the demand of a component workload, its interface has a particular resource supply derived using sbf(t) and dbf(t). These interfaces are particularly useful for compositionality of multiple components [76].

As discussed in Chapter 3, it is shown how to find efficient resource supplies using *sbf* and *dbf* for workloads with a tolerable system delay, which is experienced by any task at a given time. This avoids over-provisioning of resource supply because it allows the sbf(t) to go below the dbf(t) for a bounded amount of delay, which is added to each component rather than its tasks.

A state-of-the-art research in safety-critical real-time systems is the analysis of coscheduling of computational units and communication medium because of the necessity of separation of concerns and design optimization. Separation of concerns requires isolation of components and the communication medium to reduce the domino effect. The domino effect is a chain reaction that occurs when a small change causes a similar change nearby, which will then cause another similar change, and so on in linear sequence. Design optimization requires suitable scheduling policies to use for computation and communication, and schedules of which are necessarily not the same. This chapter discusses the synthesis of communication and computation schedules.

The remainder of the chapter is structured as follows: Section 5.1 describes the problem statement and the system model is presented in Section 5.2. Section 5.3 discusses coscheduling of computation and communication. As part of proposed coscheduling of computation and communication, Section 5.4 illustrates the steps to specify and design of component interfaces using the video streaming case-study. Section 5.5 discusses the generation of communication schedules using the derived component interfaces. Section 5.6 provides an overview of related work. Finally, Section 5.7 discusses the advantages of proposed approach.

5.1 Problem Statement

Distributed real-time systems require reliable networking solutions for the exchange of time-critical data in addition to the processors. The Network Code framework [38] is one of the real-time Ethernet-based networking solutions that supports a language to describe state-based schedules, which are TDMA-based but have the ability to make on-the-fly decisions. This on-the-fly decision making capability in schedules facilitates low average delay while changing modes. Faster mode-change also depends on how the messages are scheduled. In state-based schedules, mode-change delay will become less if more messages of different states can be allocated to a time slot. The assignments of messages is an optimization to generate a state-based communication schedule with a less number of branches. State-based schedules also allow to transmit best-effort traffic if no real-time traffic is present. The generation of such communication schedules is discussed in Chapter 4, but coscheduling of computation and communication is not discussed. Therefore, the delays introduced by computational units affect the scheduling of communication. This work aims to interconnect computational units through derived interfaces and generated communication schedules, even in the presence of delays introduced in components.

> ***Goal:*** *"Given a set of component interfaces with period, duration, a worst-case delay, generate a multi-mode communication schedule that has low average mode-change delay".*

5.2 System Model

The system model consists of a periodic resource model and periodic workload for each of the computational components connected through a shared communication medium. For each computational component, the periodic workload has a set of tasks and a set of messages for communication. The specifications of tasks and messages are derived from a physical model. In this work, we assume tasks and messages have implicit deadlines, and therefore deadline is equal to period. Tasks of a component are scheduled using EDF. Each component has a maximum tolerable delay that can be experienced by any of its tasks at a given time. For each component, the tasks in the system repeats its behaviour after the hyperperiod and the resource model replenishes available supply of resources in each period, which is the demand of computational component interface. The utilization of the workload (U_W) and the resource supply (U_R) are equal for the minimum resource usage

where transient overloads occur periodically, because the overload becomes permanent if $U_W > U_R$ or temporary if $U_W < U_R$.

Computational units (i.e., components) are characterized by the following:

- $\{C_j\}$ = a set of components where $j \in \mathbb{N}^+$
- $\{\tau_i^j\}$ = a set of tasks of component j where $i \in \mathbb{N}^+$
- p_i^j = period of task i of component j
- e_i^j = execution time of task i of component j
- (p_i^j, e_i^j) = characteristics of task i of component j
- W_j = workload of component j consisting of a set of tasks
- δ_j^* = worst-case tolerable delay of component j
- $R_j(\lambda_j, \theta_j) = \theta_j$ resource supply in every λ_j period for component j
- $I_j(\lambda_j, \theta_j, \delta_j)$ = interface of component j characterized by period λ_j, duration θ_j, and worst-case delay δ_j

and a state-based communication schedule is characterized by the following as discussed in Chapter 4:

- V = set of all communication modes
- $\{\sigma_m^k\}$ = a set of messages in communication mode k where $m \in \mathbb{N}^+$
- c_m^k = communication time of message σ_m in mode k
- π_m^k = period of each message σ_i in mode k
- (π_m^k, c_m^k) = characteristics of message m in communication mode k
- α_m instances for every message σ_m up to the end of communication round
- $v_s \in V$ denotes the current mode
- $V_d \subseteq V$ denotes the set of possible destination modes

- E is a set of tuples $\langle v_s, g_x, \lambda, v_d \rangle$ representing transitions from mode v_s to mode $v_d \in V_d$. The guard g_x is an enabling condition and λ is a set of updates on clock values. The set of transitions must be free of cycles
- B is the bandwidth assigned to scheduled messages
- L is the link capacity, with $B \leq L$.

Figure 5.1 shows an example model for two computational components with interfaces R_1 and R_2 connected through a shared bus. Each component has two tasks τ_1 and τ_2.

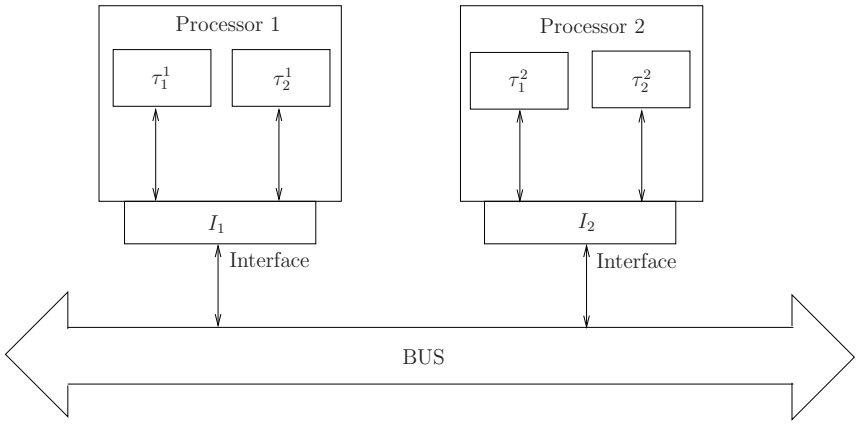


Figure 5.1: An example model of isolation between two computational components and the shared communication medium

5.3 Scheduling Computation and Communication

Scheduling computation has been extensively studied for real-time systems. Amongst them, the mostly used scheduling algorithms are EDF and RM. In [22], an extensive comparison between EDF and RM is shown. EDF is preferred over RM for better system utilization. Moreover, EDF performs reasonably well in transient overload situations. Transient overload occurs when the system needs more computing resource than available in order to be able to complete all tasks before their deadlines. Predicting Worst-Case Execution Time (WCET) is hard and sometimes inefficient, which might lead to transient overloads. Therefore, it is more realistic to design systems that can tolerate transient overloads. This work uses EDF to schedule tasks of components of the system.

As discussed in Chapter 3, an overload can be characterized as the delay that a component can suffer [11]. A periodic transient overload occurs for a certain amount of time and is repeated after the hyperperiod. In this work, we consider the periodic transient overloads which can be recovered. This delay is calculated as the difference when the overload occurs and it becomes resolved. The worst-case delay is the maximum delay that the system can suffer in the hyperperiod.

Component interfaces facilitate connecting computation and communication in a distributed system. These interfaces abstract the timing requirements of the workload of components. Using the supply and demand bound function analysis, interfaces can be derived, which provide the worst-case timing demand of all tasks in a component. We get the period and duration of an interface from the periodic resource supply that meets the demand of the workload in a component. Interfaces have an additional timing property, which is the worst-case delay that the component can experience. For example, a specification of an interface is $(3, 1, 2)$ where period, execution time, and worst-case delay are 3,1, and 2 respectively.

Coscheduling processors and network is challenging because of different specifications and requirements for each of them. If a processor scheduling is centralized and the network scheduling is distributed, the scheduling techniques are different. Moreover, a processor scheduling may not necessarily work well for the network. Therefore, it is a state-of-the-art challenge to synthesize scheduling techniques for computation and communication without violating the timing guarantees.

Different communication modes are usual in a distributed system. Switching modes at run time can cause delays in the system. We define mode-change delay as the time required to switch from one mode to another. One of the major advantages of using state-based schedules is the low average mode-change delay [8]. A key advantage of coscheduling EDF and state-based schedules is the low average mode-change delay for communication.

This chapter discusses interconnection of EDF with state-based scheduling with an assumption that overloads can occur in components which cause delays in communication. This work also aims to provide isolation of components through its interfaces and allows to interconnect them using state-based communication schedules for better bandwidth utilization, increased safety, and mixed-critical data transmission.

5.4 Specification and Design of Component Interfaces

This work introduces a model-driven approach to generate state-based schedules from interface specifications of components. The interface specifications are derived from high-level specifications, which we assume to write in a design language such as AADL [56]. The designer must provide an architectural description of the components in the system, state-machine representations of the types of communication with their interactions, and timing specifications.

The designer (1) specifies the details of components (i.e., tasks, resources, and their timing demand), transmission states and their interactions with timing requirements, (2) performs schedulability analysis and uses mechanisms to find out interfaces specifications from components information, (3) uses mechanisms not only to perform reachability and schedulability analysis, but also generates state-based schedules using components interfaces based on the application design and constraints.

A smart networked video monitoring system for collision avoidance is considered as a case-study (similar to the one discussed in Chapter 4), which is effective in different transportation systems. A mining truck is an example of such system. Mining trucks are typically over 7 meters high and need a monitoring system to avoid collisions when running with small vehicles. The monitoring system consists of four cameras connected to wheels to transmit videos of surroundings. The cameras are labelled as front camera right (FCR), front camera left (FCL), rear camera right (RCR), and rear camera left (RCL). All cameras are connected through 1[Gbit/s] Ethernet link.

5.4.1 Specification and Design

Correct specification of the components and the communication is necessary to build an efficient real-time system. Engineers can follow guidelines of a common standard to specify requirements such as AADL, or can have their own standard to write specifications. Users specify high-level abstract constructs to represent software components (processes, threads, data), hardware components (processors, memory, buses), and their interactions. Real-time properties such as period, deadline, and execution time associated to processes are mapped to hardware components. Having a unified model to specify both hardware and software components simplifies architectural representation of complex systems.

Computation specifications. Specifications of components depend on the tasks that they perform and characteristics of applications. In the case study, four sensors in the wheels are embedded along with the cameras. The cameras transmit two types of videos:

Table 5.1: Timing requirements for computation of video components

	Nominal [ms]		Normalized	
Task	Comp. Time	Period	Comp. Time	Period
τ_d	5	32	1	6
τ_c	10	64	2	12

standard quality (SQ) and high quality (HQ). When a sensor detects an object, the attached camera transmits HQ video. Sensors activate according to the movement of the truck: front sensors only activate if the vehicle is moving forward, and rear sensors only activate if the vehicle is moving backwards. We assume that the sensors perform a computation task to detect objects and the direction of the vehicle (τ_d), and the cameras perform a computation task to buffer and manage video (τ_c). Therefore, each video component has two computation tasks to schedule under the EDF scheduling policy. Considering deadline is equal to the period and the components can tolerate delays, Table 5.1 shows the timing specifications of computation tasks for each of the video components. In specifying requirements, we consider that the rate of task τ_c (30 frames per second) is higher than task τ_d to ensure that enough data is available. Assuming an atomic unit for the schedule equal to the minimum computation time for a task, we normalize the timing specifications to this time unit, and floor the normalized period. The normalization may overestimate the actual requirement, but allows us to represent all the timing requirements as multiple integers of the time unit. We assume that the timing specifications allow video components to tolerate a delay of 2 time units in the worst-case, which is within the tolerable delay range of video broadcasting for a QoS as discussed in [29]. This case-study assumes to transmit the raw video, leaving the use of encoder (e.g. MPEG-2) and different frames (i.e., I, P, B) as discussed in [25] for the future work. Instead of determining the tolerable delay based on a particular QoS, a control component can have a maximum delay tolerance depending on stability metrics as discussed in [11].

The display component is located near the vehicle driver to receive data periodically from the network. We assume that the display component has a single task that executes faster than other tasks and therefore the computation time and period are set to 2 and 4 time units respectively. We assume that the display component can tolerate a delay of 1 time unit.

Communication specifications. Specifications of communication define its operational requirements for an application. For the case study, the specifications of the cameras

Table 5.2: Timing requirements for communication

	Nominal [μs]		Normalized	
FPS	Comm. Time	Period	Comm. Time	Period
15	7600	66700	6	48
30	7600	33350	6	24

Table 5.3: Feasible modes with messages specifications

ID	Mode (FCR, FCL, RCR, RCL)	Utilization
1	(SQ,SQ,SQ,SQ)	0.5
2	(SQ,SQ,SQ,HQ)	0.625
3	(SQ,SQ,HQ,SQ)	0.625
4	(SQ,SQ,HQ,HQ)	0.75
5	(SQ,HQ,SQ,SQ)	0.625
6	(HQ,SQ,SQ,SQ)	0.625
7	(HQ,HQ,SQ,SQ)	0.75

transmission characteristics are taken from [8]. Each of the four cameras operate in two states based on the FPS. The SQ frames are transmitted by default and cameras switch to transmitting HQ frames once an object is detected. Moreover, either front cameras or rear cameras can transmit HQ frames after an object is detected.

The possible configurations of the communication is calculated as the cross products of the state machines. The timing requirements of different types of video transmission (i.e., HQ and SQ) as shown in Table 5.2 are calculated using pixel specification (640×480) of frames [8] and network bandwidth (1Gbit/s). To find the feasible communication modes without considering computation delays, schedulability analysis can be performed using Equation 4.2 as defined in Chapter 4. Table 5.3 shows feasible communication modes for the video case-study where $\frac{B}{L} = 1$.

5.4.2 Finding Component Interfaces

A component has a workload consisting of tasks with timing specifications and can use an interface to specify the resource requirement. Under the EDF scheduling policy, a suitable

resource supply (i.e., period and duration) is calculated using the characteristics of supply and demand bound functions [11]. To ensure finding an efficient resource supply R_j for a component j, the utilization of the resource supply is kept the same as the workload utilization. Figure 5.2 shows the workflow to find suitable resource supplies. The algorithm based on the supply calculation model [11] searches for resource supplies that have recovery points at δ_j^* distance from possible overload points (periods of tasks) and validates calculated resource supplies using supply and demand bound functions. After getting a set of valid supplies, one of them is chosen as an efficient resource supply based on application characteristics. An efficient resource supply becomes the first two parameters in the specification of an interface I. The delay that the component experiences because of using the suitable resource supply becomes the third parameter of the interface specification.

A suitable resource supply meets the specifications of the workload and avoids over-provisioning of resources. For example, if the display component has the workload $(3, 1)$, then a resource supply $(4, 2)$ does not meet specification because the worst-case delay is greater than 1. However, the resource supply $(2, 1)$ meets the specification and avoids over-provisioning of resources compared to $(1, 1)$. For each of the components located in the wheels, the interface specification is $(3, 1, 2)$, where 3 is the period, 1 is the duration, and 2 is the worst-case delay that the component can experience.

5.5 Generation of State-Based Schedules

To generate state-based schedules, the first step is to extract internal state representations of communication for each of the components as discussed in the previous sections. We refer the cross product of the states that are reachable as communication modes. A schedulability test is required to find communication modes that can meet timing requirements. To characterize a good state-based schedule, we use the properties of state-based schedules as defined in Chapter 4.

Component interfaces provide additional information in generating state-based schedules for communication. In particular, delays in computation have an impact on communication and its schedules. Because, delays in computation cause the data to become available late for transmission and perhaps require the component to operate in a different communication mode to meet the timing requirements.

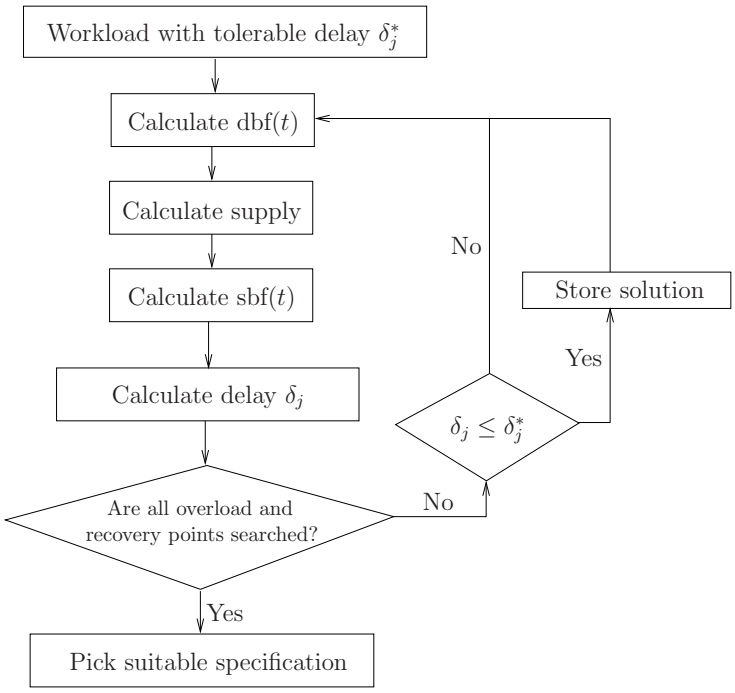


Figure 5.2: Finding specification for component interfaces

5.5.1 Optimization Model

The problem of finding an optimal state-based schedule with respect to minimizing the number of groups can be solved using ILP. To do this, the model has a constraint on computation time of each message to obtain at least the required time slots. A boolean variable x_{ml}^k is set to 1 if a message m is allocated to a slot l in mode k, and 0 otherwise. A computation delay that results a message m in mode k to be transmitted late is characterized using δ_m^k. We discuss mechanisms to choose to a delay value in Section 5.5.3. The number of overlaps for a slot l can be increased if same message m is allocated to the previous slots in all modes. A constraint using a variable s_{ml}^k enforces this. To ensure the occurrence of overlaps earlier than later, a weight (l) is multiplied to each slot assignment x_{ml}^k in the objective function. Minimizing the sum of values of variables s_{ml}^k and x_{ml}^k which are multiplied by weights provides the minimum number of groups in the schedule. The objective of using weights to assign x_{ml}^k as early as possible and vice versa for s_{ml}^k. We minimize the number of groups to find out the assignments of messages to slots to generate a state-based schedule that has low average mode-change delay.

- x_{ml}^{k} coefficient determines the usage of a time slot for $m \in N$, $l \in \{1, \ldots, \text{LCM}\{\pi_m^k\}\}$ and $k \in V$, where N represents the set of messages and V represent the number of modes. These coefficients are defined as follows:

$$x_{ml}^{k} = \begin{cases} 1 & \text{if message } \sigma_m \text{ uses the slot in mode } k \\ \\ 0 & \text{otherwise.} \end{cases}$$

- s_{ml}^{k} coefficient determines the overlaps. The variable s_{ml}^{k} is set to 1 if a message m is allocated a slot l in mode k, and other messages except m are allocated at the same slot l in modes except k.

$$s_{ml}^{k} = \begin{cases} 1 & \text{if } x_{ml}^{k} = 1 \wedge \\ & \exists x_{ul}^{v}, st : u \neq m \wedge v \neq k \wedge x_{ul}^{v} = 1 \\ \\ 0 & \text{otherwise.} \end{cases}$$

- w_{ml}^{k} determines overlaps. This determines whether different messages of other modes are allocated to the same slot l if a message m of mode k is already allocated to slot l.

- δ_m^k is an input delay value for message m in mode k.

- $\Gamma^k = \{1, \ldots, \text{LCM}\{\pi_m^k\}\}$ is the set of all slots until the hyperperiod of mode k.

$$\min \quad \sum_{\forall \mathrm{m}\in \mathrm{N}, l\in\Gamma_m^k, \mathrm{k}\in \mathrm{V}} s_{ml}^k \times (\Gamma_m^k - l) + \sum_{\forall \mathrm{m}\in \mathrm{N}, l\in\Gamma_m^k, \mathrm{k}\in \mathrm{V}} x_{ml}^k \times l.$$

$$st. \quad \mathbf{C}_1\{\forall \mathrm{m} \in \mathrm{N}, \mathrm{k} \in \mathrm{V}\} :$$

$$\sum_{l=g\pi_m^k+1}^{g\pi_m^k+\pi_m^k} x_{ml}^k \geq c_m^k + \delta_m^k \, , g = 0, \ldots \, \alpha_m - 1;$$

$$\mathbf{C}_2\{\forall l \in \Gamma_m^k, k \in \mathrm{V}\} :$$

$$\sum x_{ml}^k \leq 1,$$

$$\mathbf{C}_3\{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma_m^k,$$

$$\mathrm{k} \in \mathrm{V}, \mathrm{u} \in \mathrm{N} - \{m\}, \mathrm{v} \in \mathrm{V} - \{k\}\} :$$

$$w_{ml}^k \geq x_{ul}^v,$$

$$\mathbf{C}_4\{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma_m^k, \mathrm{k} \in \mathrm{V}\} :$$

$$s_{ml}^k \geq x_{ml}^k + w_{ml}^k - 1,$$

$$\mathbf{C}_5\{\forall \mathrm{m} \in \mathrm{N}, l \in \Gamma_m^k, \mathrm{k} \in \mathrm{V}\} :$$

$$s_{ml}^k \geq 0,$$

Constraint C_1 specifies that all messages at least get the computation units in their periods even though a delay occurs. Constraint C_2 specifies that no two messages are assigned to the same slot at the same mode. Constraint C_3 enforces overlaps. Constraint C_4 and C_5 enforces s_{ml}^k to be non-negative and binary. The objective function includes x_{ml}^k to assign overlaps as early as possible in the hyperperiod for less number of branches in the generated state-based schedule.

5.5.2 Determining Valid Schedules

Computational delays may cause the optimization model unable to find a state-based communication schedule where messages can meet the timing constraints during the hyperperiod. If a valid schedule is absent for a given set of messages with timing constraints, the optimization model reports on infeasibility. Therefore, it is sufficient to use the optimization model for schedulability analysis of a set of messages that can experience computation delays.

Definition 8 (Schedulability with delays) *A given set of communication messages* $\{m\}$ *in a set of modes* $\{k\}$ *up to slot* $l = 1 \ldots, \mathrm{LCM}\{\pi_{\mathrm{m}}^{\mathrm{k}}\}$ *will be schedulable for a given delay specification if the optimization model returns an assignment of* x_{ml}^{k} *for a given set of communication messages* $\{m\}$ *in all modes* $\{k\}$ *upto* $l = 1 \ldots, \mathrm{LCM}\{\pi_{\mathrm{m}}^{\mathrm{k}}\}$.

5.5.3 Determining Delays

Interfaces provide specifications of maximum tolerable delays of components. Thus, if δ^* is the maximum delay and δ_r is a delay that components may experience, then δ_r will be no greater than δ^*. Since state-based communication schedules have the ability to make decisions at runtime, it is possible to switch between schedules at run time for different delays. Considering a set of delays as *dynamic* possible values of δ_r no greater than the maximum tolerable delay may provide a greater range of schedulability with an increase of number of generated schedules. In contrast, a system may use the maximum of tolerable delays of all components as the *static* value of δ_r to generate a state-based schedule. Using the static value of δ_r may reduce the schedulability of operational modes but provides guarantees that the schedule is valid for all dynamic values of δ_r.

Dynamic delays

To consider dynamic values of delays, we assume a set Δ_r of finite number of non-negative delays up to the maximum delay (Equation 5.1). Therefore, if δ_t is the delay that a component experiences at run time, then $\delta_t \leq \delta_r$, where $\delta_t \in \mathbb{R}_{\geq 0}$. Different δ_r values may be feasible to generate a state-based schedule for components running in a communication mode. Figure 5.3 shows an example of different δ_r values for which a state-based communication schedule can be generated for component C_1 and C_2 in mode k. We can reduce the cardinality of Δ_r by choosing a local maximum of δ_r as shown in Figure 5.3 to generate state-based schedules for a less number of δ_r values.

$$\Delta_r = \{\delta_r \mid (\delta_r < \delta^*) \cup (\delta_r = \delta^*), \delta_r \in \mathbb{N}_{\geq 0}\}\,. \tag{5.1}$$

Example 5 *Consider two components with a worst-case tolerable delay of 3 time units for the communication specification as shown in Table 5.2 and 5.3. Under the assumption of dynamic delays,* δ_r *can be set to 0, 1, 2, and 3. State-based schedules are generated for the* δ_r *values that the optimizer can use to produce a feasible solution. Thus, if components are*

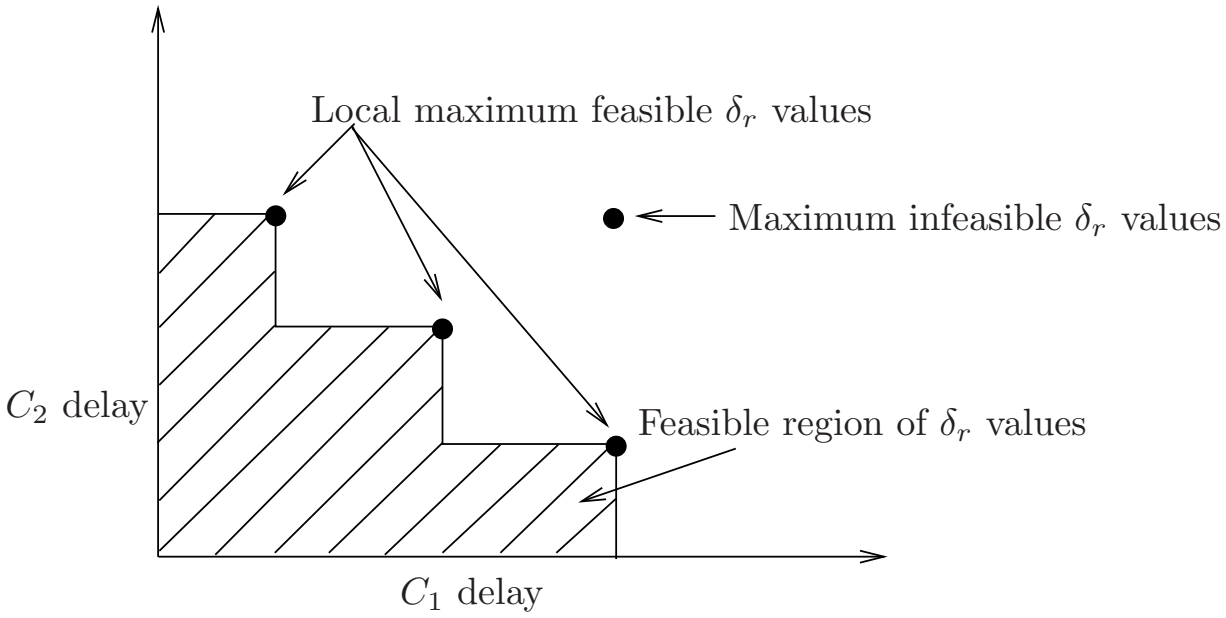


Figure 5.3: Analysis on δ_r values in mode k

running in mode 7 and experience a delay $\delta_t \leq 2$, it can still operate in the same mode. However, if $2 < \delta_t \leq 3$, then the components can switch to other feasible modes such as mode 1 to continue operation using valid schedules.

Static delay

Each component j may experience a delay δ_j^* in the worst-case, which not necessarily the maximum delay among all components in the system. Under the static delay assumption, the value of δ_r is the maximum of all delays that components can experience in the worst-case (Equation 5.2).

$$\delta_r = \max_j \delta_j^*. \tag{5.2}$$

In the case-study, the worst-case delay that the components can suffer in each hyperperiod is 2 time units. Under the assumption of dynamic delays, the δ_r values are non-negative values up to 2 time units. In contrast, the δ_r value is 2 time units under the assumption of static delay.

5.5.4 Construction of State-based Schedules

Optimization model provides assignments of messages to slots for each of the communication mode if a state-based schedule can be generated. To construct the schedule that has messages timing requirements met within the hyperperiod, groups are formed using overlaps. However, to guarantee timing requirements upon a mode change, separate groups are formed for messages scheduled in the current slot but not in the previous slot. The decomposition method as discussed in Chapter 4 uses the optimal assignments of x_{ml}^{k} (messages to slots) to construct a state-based schedule.

5.5.5 Experimental Evaluation

For the case-study, we consider the static delay assumption in choosing δ_r, because the optimizer can output valid assignments of messages to slots. That is, the schedule generated for the static value of δ_r is applicable to dynamic values of δ_r. Moreover, the static (i.e., worst-case) delay makes use of the communication bandwidth no less than dynamic delays. This yields robust experimental analysis.

Mode-change delay analysis

To demonstrate that our optimizer assigns the slots to messages efficiently with respect to minimizing the number of groups in the generated schedule, we use random assignments of x_{ml}^{k} that also meet the timing requirements of messages in each state. For the video-streaming case study with the maximum delay of 2 time units, Table 5.4 shows the difference in the average mode-change delay for the generated schedule using optimization constraints and the best-case of 10000 times generated randomized but valid schedules. The average mode-change delay in the generated optimized schedule is found significantly better than the randomized schedules. Moreover, to construct a state-based schedule, we also assign messages to slots based on deadlines as used in the EDF scheduling policy, which also results higher average mode-change delay than the optimized schedule.

Scalability analysis

Scalability is important because it represents the magnitude of a system. A large-scale system may have significant volume of specifications of different components. Although the schedule is generated off-line, our optimization model can find a solution quickly for

Table 5.4: Average mode-change delay analysis

Types of assignments	Delay [in ms]
Optimized	27.77
Randomized	33.13 (Best=30.62, Worst=35.03)
EDF-assigned	29.45

Table 5.5: Optimizer execution information for the video case-study

Modeling language	AMPL
Optimization solver	Gurobi 4.6.1
Time elapsed	308 (in seconds)
Objective	6253
Simplex iterations	1430739
Branch-and-cut nodes	6371

a reasonable number of slots. This also motivates to use a different range of δ_r values in generating state-based schedules. Table 5.5 shows the details including timing to solve the optimization problem of the video case-study that has 48 time slots in a hyperperiod using a well-known mathematical programming language, A Mathematical Programming Language (AMPL) and the solver GUROBI [1]. In this case-study, two of the modes (ID 4 and 7) utilize the communication medium 100% when the worst-case delay is 2 time units. Despite this, the solver produces results in about 5 minutes, which indicates high scalability of the proposed solution.

5.6 Related Work

Separation of concerns has been gaining increasing attention because of the safety requirements in many real-time systems [85]. Separation of concerns has already been discussed in some research [49] but not precisely in separating computation and communication. Several protocols such as Giotto [43] attempt to reduce the resource interdependency by separating the resource usage in layers. Giotto separates the value and execution domain. In Giotto, program execution is independent of the reading and writing of values. A

TTP [51] or state-based [37] schedule separates the communication from computation so that sending and receiving messages are independent of the tasks running on the stations. In PEACOD [7], the authors provide a framework for specifying resource consumptions for small pieces of code to provide determinism.

Separation of concerns allows to have different scheduling policies exist together. A computational component may have different tasks scheduled under multiple scheduling policies such as EDF and RM. Timing requirements of tasks scheduled under different scheduling policies are converted into a single requirement. In [77], the authors propose a compositional real-time scheduling framework for real-time systems which can be used to establish global (system level) timing properties of a component from individual timing properties of tasks running on a resource. The authors present schedulability conditions for a periodic task model and propose a periodic resource model under EDF and RM scheduling. This periodic resource model can compute a single timing demand from multiple timing requirements using supply and demand bound functions. In another related work [76], the authors analyze compositional schedulability of a bounded delay resource partition model. In [76], the authors propose algorithms to define optimal interfaces for the subsystems which may share resources. Integrating subsystems into a system having optimal interfaces provides isolation in developing adaptive and reconfigurable systems. In [84], the authors propose a compositional analysis framework that uses real-time calculus and assume/guarantee interfaces.

Traditional real-time communication protocols allow limited control to the applications over the communication behaviour at runtime. For example, application developers have to assign message priorities statically on a CAN bus to ensure that the priorities are unique [16]. FlexRay [39] follows a TDMA approach, assigning an specific slot at the end of each round for dynamic and arbitrary traffic. However, stations must always wait for that specific slot to transmit dynamic messages, and the timing of the messages transmitted during that slot is not guaranteed. Using state-based communication schedules [38], stations can make decisions at runtime and timing of the messages transmitted during that slot can be guaranteed. This provides flexibility as well as predictability in message transmissions.

A recent work [8] discusses a workflow for generating state-based schedules from high-level specifications, but avoids considering computational specifications such as task delays. Since the system model in [8] assumes that messages cannot be delayed, the generation technique will not apply for connected computational components that are delay-tolerant. This work therefore aims to synthesize computation and communication through isolation and generation of communication schedules from derived component interfaces.

5.7 Discussion

One of the key advantages of using state-based communication schedules is predictability, which is required in safety-critical systems. Predictable communication scheduling can have deterministic behaviour. Behaviour of a communication scheduling depends on not only functionality but also architectural and execution properties such as resource consumption and timing. Defining all possible behaviours prior communication is necessary to guarantee determinism. In generating state-based schedules, all possible behaviours are analyzed and specified *a priori*. This avoids non-determinism and makes the execution of communication predictable. Using component interfaces, the determinism in communication scheduling holds even in the presence of delays.

Lemma 2 *Given a system with a number of components connected through a communication medium using interfaces and state-based schedules, the communication delay is bounded.*

Proof The amount of delays that a component experiences is bounded, transient, and periodic in nature, because of Theorem 2 in Chapter 3. Consider a component j with workload utilization $U_W^j = \sum_i \frac{e_i^j}{p_i^j}$ and resource utilization $U_R^j = \frac{\theta^j}{\lambda^j}$, where task i executes for e_i^j in every p_i^j and resource R_j provides θ_j units of time in every λ_j. If $U_R^j = U_W^j$, then after $t = 2(\lambda_j - \theta_j)$, the function $f(t)$ that represents overloads using sbf(t) and dbf(t) is periodic with period LCM$(\lambda, p_1^j, \ldots, p_n^j)$, i.e.,

$$f(2(\lambda_j - \theta_j) + t + y\text{LCM}(\lambda, p_1^j, \ldots, p_n^j)) \\ = f(2(\lambda_j - \theta_j) + t), \quad \forall t \in \mathbb{R}_{\geq 0}, \forall y \in \mathbb{N}.$$

However, the delays that components can experience in the worst-case are within the delay tolerance, because this is guaranteed through calculating an efficient resource supply [11]. Therefore, the communication delay depends on the mode-change delay in the worst-case. Since, the communication schedule repeats after the hyperperiod, the communication delay is bounded to the hyperperiod of all messages transmitted in the network.

This chapter illustrates the design and schedule generation process for multi-mode communication using component interfaces. A major reason in generating an optimized state-based schedule is to allow the system to switch between communication modes at run time, while meeting the real-time requirements of all messages. This ability of switching modes facilitates low average mode-change delays.

Chapter 6

Conclusions

Current real-time systems are inherently complex and built with heavy over-provisioning of resources to compromise between safety and functionality, because transient overloads can lead to failure due to missing deadlines. For example, the first flight of the space shuttle was delayed because of a transient overload during system initialization on one of the processors dedicated to the control of the shuttle [21]. However, balancing safety and functionality requirements is required to maximize the performance of a system. Some systems such as control systems can tolerate transient overloads (i.e., bounded delays) and thus would not need such high over-provisioning. This will then allow lower overall resource usage or running more functionality. A complex real-time system is, in fact, a multi-mode system that facilitates high functionality, but also demands guarantees on safety requirements.

Model-driven development is complex for time-critical systems not only because of meeting deadlines at runtime, but also the effectiveness of generated schedules that represent the runtime behaviour. A well understood approach of model-driven development is to create a system-level design and generate schedules specific to the execution framework. In the system model, an architecture represents both processor and network. Therefore, in this dissertation, we analyzed both processor and network scheduling techniques that are suitable for a complex system design.

A well-known scheduling policy for processors is EDF. This dissertation presented a comprehensive analysis to characterize overloads under the EDF scheduling policy using overload points and recovery points for systems experiencing transient overloads. To understand the impact of overloads, we defined overload metrics such as the worst-case delay and the worst-case severity. Using the analysis of overloads, we proposed an efficient resource supply model for a given workload and a tolerable worst-case delay. Control en-

gineers can use the framework for feedback control systems, which was demonstrated by simultaneously stabilizing two plants with a single computational resource.

A suitable scheduling policy for networks is state-based scheduling. In this thesis, we presented a workflow with an illustration of a real-time video streaming case-study to implement higher level abstractions through generating state-based schedules that facilitate conditional executions at runtime. We also demonstrated that the generated schedules using constraints through a linear optimization solver are better than schedules generated using EDF and a randomized algorithm because of lower average mode-change delays.

The complexity of real-time systems is increasing and therefore safety is becoming a major challenge and concern in next-generation distributed systems. Several standards in different domains require evidence to certify that a system is safe. In these circumstances, separation of concerns is an effective design methodology to increase safety because of reducing complexity and dependency. This dissertation argued to use EDF for processor scheduling and state-based schedules for network scheduling to achieve both performance and safety through separation of concerns. This dissertation presented a workflow to implement higher level component abstractions using interfaces and generate state-based schedules that facilitate conditional executions at run time. Therefore, the work in this thesis can be used for safety through isolation and also to provide high performance due to efficient control on communication and low average mode-change delay.

Chapter 7

Future Work

This thesis discusses the characterization of transient overloads under EDF scheduling. However, overloads are also relevant to analyze using supply and demand bound functions for other scheduling policies. For these different scheduling policies it is also necessary to derive a resource supply model to reduce overprovisioning of resources for workloads that can tolerate overloads. Therefore, the analysis of overloads and its impacts on providing resource supplies is also useful in case a different scheduling policy than EDF is preferred.

In this thesis, transient overloads are characterized as bounded delays for control systems, but this observation can also be applied to other real-time systems that experience bounded transient overloads. An example is a multi-mode car component running infotainment applications where delays can occur while changing modes. An efficient resource supply can be derived for the multi-mode component to reduce overprovisioning of resources that guarantee meeting timing requirements of tasks.

This dissertation presents analysis of transient overloads under the preemptive EDF scheduling policy for a uniprocessor or partitioned multiprocessor system. However, the problem statement is also applicable to using other scheduling schemes that are static priority-based, global, or non-preemptive in multicore or manycore systems. Therefore, extensions of overload analysis to these areas can also become potential future work.

The proposed resource supply model can be extended to handle different types of tasks such as sporadic tasks. Sporadic tasks have a minimum inter-arrival time. Therefore, the *dbf* for sporadic tasks is different from periodic tasks. For sporadic tasks, $\text{dbf}(W, t, \text{EDF}) = \sum_{\tau_i \in W} \max(0, (\left\lfloor \frac{t-d_i}{p_i} \right\rfloor + 1)e_i)$. In the worst case, sporadic tasks can arrive periodically based on their inter-arrival time and our model can still handle such workloads.

The resource supply searching algorithm presented in this dissertation increases the value of k in integers rather than real numbers to reduce the time complexity of finding a suitable resource supply. However, due to the increase of k in integers rather than real numbers, the algorithm cannot guarantee to find an optimal resource supply. Finding an optimal resource supply can be considered as a potential future work to increase the efficiency of resource usage.

The hyperperiod grows exponentially as a function of the longest period, number of tasks, and co-primeness of the period of the tasks [20]. Task period selection to minimize the hyperperiod is a potential future work related to this thesis. In general, a shorter hyperperiod is preferable over a longer hyperperiod, because the shorter the hyperperiod the lower the complexity of searching a valid and feasible resource supply.

A hierarchical scheduling in each component is also possible where different specifications in each level are combined together to get a single timing requirement. The resource supply turns into demand in each level and continues until no workload specification is left to combine. A compositional framework uses hierarchical scheduling to reduce complexity by abstracting subcomponents in a component. Therefore, using compositionality can reduce the number of interfaces.

This thesis presents state-based scheduling based on Ethernet, but results are also relevant for real-time communication in general. State-based scheduling is applicable to other technologies operating at layers one and two of the OSI model. Therefore, if industry shifts the interest to a different communication media in the future, we will be able to port the developed cores for time-triggered coordination by simply replacing the Ethernet core with the proper interfacing to the underlying MAC.

In this dissertation, the experimental analysis section of generating state-based schedules assumes uniform distribution of mode changes. An effective extension would be to apply different distributions on mode changes following practical models. However, this requires to find mode-change patterns which highly depend on the types of applications. Several statistical models can also be applied using stochastic processes.

The scope of this work considered as modelling and design, and therefore we avoid the details on hardware infrastructure for the efficient and reliable execution of generated schedules. Although we have a framework [2] implemented to generate state-based schedules from a high-level specification, a gap remains to run them in the existing hardware and try different hardware platforms. This additional work will greatly improve the usability of the system, and the results would be relevant for the real-time and embedded software domains.

List of Symbols

$\{C_j\} =$ a set of components where $j \in \mathbb{N}^+$

$\{\tau_i^j\} =$ a set of tasks of component j where $i \in \mathbb{N}^+$

$p_i^j =$ period of task i of component j

$e_i^j =$ execution time of task i of component j

$(p_i^j, e_i^j) =$ characteristics of task i of component j

$W_j =$ workload of a component consisting of a set of tasks

$\delta_j^* =$ maximum tolerable delay of component j

$R_j(\lambda_j, \theta_j) = \theta_j$ resource supply in every λ_j period for component j

$I_j(\lambda_j, \theta_j, \delta_j) =$ interface of component j characterized by period λ_j, duration θ_j, and worst-case delay δ_j

$V =$ set of all communication modes

$\{\sigma_m^k\} =$ set of messages in communication mode k where $m \in \mathbb{N}^+$

$c_m^k =$ communication time of message σ_m in mode k

$\pi_m^k =$ period of each message σ_i in mode k

$(\pi_m^k, c_m^k) =$ characteristics of message i in communication mode k

$\alpha_m =$ number of instances for every message σ_i up to the end of communication round

$v_s =$ the current mode

$V_d =$ the set of possible destination modes

$B =$ the bandwidth assigned to scheduled messages

$L =$ the link capacity, with $B \leq L$

Acronyms

AADL Architecture Analysis and Design Language.

AMPL A Mathematical Programming Language.

CAN Controller Area Network.

CO Cardiac Output.

EDF Earliest Deadline First.

ET-messages Event-Triggered Messages.

FPS Frame-Per-Second.

IFG Inter-Frame Gap.

ILP Integer Linear Programming.

LAP Left Atrial Pressure.

LEF Largest Error First.

LLF Least Laxity First.

LLREF Largest Local Remaining Execution Time First.

MAC Medium Access Control.

MARTE Modelling and Analysis of Real-Time and Embedded Systems.

MEF-TOD Maximum-Error-First with Try-Once-Discard.

NCS Networked Control System.

PAP Pulmonary Artery Pressure.

PCWP Pulmonary Capillary Wedge Pressure.

PFair Proportionate Fair.

PVR Pulmonary Vascular Resistance.

QoS Quality of Service.

RM Rate Monotonic.

RTE Real-Time Ethernet.

TDMA Time Division Multiple Access.

TMR Triple Modular Redundancy.

TT-messages Time-Triggered Messages.

TTEthernet Time-Triggered Ethernet.

TTP Time-Triggered Protocol.

UML Unified Modelling Language.

WCET Worst-Case Execution Time.

References

[1] AMPL—A Mathematical Programming Language. www.ampl.com. Visited November. 2014.

[2] Open-source scripts. http://www.mathworks.com/matlabcentral/fileexchange/44716. Visited November. 2014.

[3] Profinet. http://www.profibus.com. Visited November. 2014.

[4] Time-Triggered Ethernet. http://www.tttech.com. Visited November. 2014.

[5] R. Alur and G. Weiss. Regular Specifications of Resource Requirements for Embedded Control Software. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 159–168, Washington, DC, USA, 2008.

[6] M. Anand, S. Fischmeister, Y. Hur, J. Kim, and I. Lee. Generating Reliable Code from Hybrid Systems Models. *IEEE Transactions on Computers*, 59(9):1281–1294, 2010.

[7] M. Anand, S. Fischmeister, and I. Lee. Resource Scopes: Toward Language Support for Compositional Determinism. In *Proceedings the 12th IEEE International Symposium on Object/component/service-oriented Real-time Distributed Computing (ISORC)*, pages 295–304, Tokyo, Japan, March 2009.

[8] A. Azim, G. Carvajal, R. Pellizzoni, and S. Fischmeister. Generation of Communication Schedules for Multi-Mode Distributed Real-Time Applications. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 293:1–293:6, Dresden, Germany, March 2014.

[9] A. Azim and S. Fischmeister. Rollback to reduce mode-change delays. https://bitbucket.org/aazim/mode-change/downloads. Visited November. 2014.

[10] A. Azim and S. Fischmeister. Resolving State Inconsistency in Distributed Fault-Tolerant Real-Time Dynamic TDMA Architectures. In *International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, Toulouse, France, September 2011.

[11] A. Azim, S. Sundaram, and S. Fischmeister. An Efficient Periodic Resource Supply Model for Workloads with Transient Overloads. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 249–258, Paris, France, July 2013.

[12] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized Multiframe Tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[13] S. Baruah and N. Fisher. The Partitioned Multiprocessor Scheduling of Deadline-Constrained Sporadic Task Systems. *IEEE Transactions on Computers*, 55:918–923, 2006.

[14] A. Bastoni, B.B. Brandenburg, and J.H. Anderson. An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 14–24, San Diego, USA, Nov 2010.

[15] E. Bini, G. Buttazzo, and Y. Wu. Selecting the Minimum Consumed Bandwidth of an EDF Task Set. In *Proc of 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, Washington, D.C, USA, December 2009.

[16] Bosch. *CAN Specification, Version 2.* Robert Bosch GmbH, September 1991.

[17] S. P. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[18] B.B. Brandenburg, J.M. Calandrino, A. Block, H. Leontyev, and J.H. Anderson. Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin? In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 342–353, St. Louis, USA, April 2008.

[19] M.S. Branicky, S.M. Phillips, and W. Zhang. Scheduling and Feedback Co-Design for Networked Control Systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, pages 1211–1217, Las Vegas, USA, December 2002.

[20] V. Brocal, P. Balbastre, and R. Ballester. Task Period Selection to Minimize Hyper-period. In *IEEE Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, Toulouse, France, September 2011.

[21] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[22] G. C. Buttazzo. Rate Monotonic vs. EDF: Judgment Day. *Real-Time Systems*, 29(1):5–26, January 2005.

[23] G. Carvajal, M. Figueroa, R. Trausmuth, and S. Fischmeister. Atacama: An Open FPGA-based Platform for Mixed-Criticality Communication in Multi-segmented Ethernet Networks. In *Proc. of the 21st IEEE Int. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 121–128, Seattle, USA, 2013.

[24] G. Carvajal and S. Fischmeister. A TDMA Ethernet Switch for Dynamic Real-Time Communication. In *Proc. of the 18th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 119–126, Charlotte, USA, May 2010.

[25] S. Chakraborty, T. Mitra, A. Roychoudhury, L. Thiele, U.D. Bordoloi, and C. Derdiyok. Cache-Aware Timing Analysis of Streaming Applications. In *19th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 159–168, Pisa, Italy, July 2007.

[26] C.T. Chen. *Linear Systems, Theory and Design*. Oxford University Press, 1999.

[27] X. Chen, A. Azim, X. Liu, and S. Fischmeister. CSS: Conditional State-based Scheduling for Networked Control Systems. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 78–87, Seoul, Korea, August 2012.

[28] X. Chen, A. Azim, X. Liu, S. Fischmeister, and J. Ma. DTS: Dynamic TDMA scheduling for Networked Control Systems. *Journal of Systems Architecture*, 60(2):194–205, 2014.

[29] Y. Chen, T. Farley, and N. Ye. QoS Requirements of Network Applications on the Internet. *Information Knowledge Systems Management*, 4(1):55–76, January 2004.

[30] Q.-L. Han D. Yue and C. Peng. State Feedback Controller Design for Networked Control Systems. *IEEE Transactions on Circuits and Systems – II: Express Briefs*, 51(11):640–644, November 2004.

[31] R. I. Davis and A. Burns. A Survey of Hard Real-time Scheduling for Multiprocessor Systems. *ACM Computing Surveys*, 43(4):1–44, 2011.

[32] J.-D. Decotignie. The Many Faces of Industrial Ethernet. *IEEE Industrial Electronics Magazine*, 3(1):8 –19, March 2009.

[33] U.M.C Devi and J.H Anderson. Tardiness Bounds under Global EDF Scheduling on a Multiprocessor. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 330–341, Miami, USA, December 2005.

[34] A. Easwaran, M. Anand, and I. Lee. Compositional Analysis Framework Using EDP Resource Models. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS)*, pages 129–138, Washington, D.C., USA, 2007.

[35] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental Schedulability Analysis of Hierarchical Real-time Components. In *R. Shelton, ORMSC/970609, Open Engineering*, pages 272–281. ACM Press, 2006.

[36] S. Fischmeister and A. Azim. Design Choices for High-Confidence Distributed Real-time Software. In *Proc. of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 327–342, Crete, Greece, October 2010.

[37] S. Fischmeister, O. Sokolsky, and I. Lee. A Verifiable Language for Programming Communication Schedules. *IEEE Transactions on Computers*, 56(11):1505–1519, November 2007.

[38] S. Fischmeister, R. Trausmuth, and I. Lee. Hardware Acceleration for Conditional State-Based Communication Scheduling on Real-Time Ethernet. *IEEE Transactions on Industrial Informatics*, 5(3):325–337, 2009.

[39] FlexRay Consortium. *FlexRay Communications System — Protocol Specification*, June 2004. Version 2.0.

[40] G. C. Buttazzo and M. Caccamo and L. Abeni. Elastic Scheduling for Flexible Workload Management. *IEEE Transactions on Computers*, 51:289–302, 2002.

[41] M. Grant and S. Boyd. CVX: Matlab Software for Disciplined Convex Programming, version 1.21. http://cvxr.com/cvx/, April 2011.

[42] V. Gupta, A. F. Dana, J. Hespanha, R. M. Murray, and B. Hassibi. Data Transmission Over Networks For Estimation and Control. *IEEE Transactions on Automatic Control*, 54(8):1807–1819, August 2009.

[43] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A Time-Triggered Language for Embedded Programming. *Proceedings of the IEEE*, 91(1):84–99, January 2003.

[44] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu. A Survey of Recent Results in Networked Control Systems. *Proceedings of the IEEE*, 95(1):138–162, January 2007.

[45] S.H. Hong. Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems. *IEEE Transactions on Control Systems Technology*, 3(2):225–230, 1995.

[46] S.H. Hong and W.H. Kim. Bandwidth Allocation Scheme in CAN Protocol. *IEEE Proceedings Control Theory and Applications*, 147(1):37–44, 2000.

[47] S. C. Hsieh. Product Construction of Finite-State Machines. In *Proc. of the World Congress on Engineering and Computer Science*, pages 141–143, San Francisco, USA, 2010.

[48] O. C. Imer, S. Yuksel, and T. Basar. Optimal Control of LTI Systems Over Unreliable Communication Links. *Automatica*, 42(9):1429–1439, September 2006.

[49] D. Jackson and E. Kang. Separation of Concerns for Dependable Software Design. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER, pages 173–176, New Mexico, USA, 2010.

[50] K. Ji and W. Kim. Stochastic Optimal Control and Network Co-design for Networked Control Systems. *Proceedings of International Journal of Control Automation and Systems*, 5(5):515, 2007.

[51] H. Kopetz. *Real-time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[52] H. Kopetz. The Rationale for Time-Triggered Ethernet. In *Real-Time Systems Symposium (RTSS)*, pages 3–11, Barcelona, Spain, 2008.

[53] H. Kopetz, G. Bauer, and S. Poledna. Tolerating Arbitrary Node Failures in the Time-Triggered Architecture. *SAE 2001 World Congress, March 2001, Detroit, MI, USA*, March 2001.

[54] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a Federated to An Integrated Architecture for Dependable Embedded Real-time Systems. In *Technische Univ. Wien, Vienna, Austria. Rep. 22*, 2004.

[55] P. Kumar, J. Chen, L. Thiele, A. Schranzhofer, and G.C. Buttazzo. Real-Time Analysis of Servers for General Job Arrivals. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 251–258, Toyama, Japan, August 2011.

[56] G. Lasnier, T. Robert, L. Pautet, and F. Kordon. Behavioral Modular Description of Fault-Tolerant Distributed Systems with AADL Behavioral Annex. In *Conference on New Technologies of Distributed Systems*, pages 17–24, Tozeur, Tunusia, 2010.

[57] J. Lee, I. Shin, and A. Easwaran. Online Robust Optimization Framework for QoS Guarantees in Distributed Soft Real-time Systems. In *Proceedings of the ACM international conference on Embedded software*, EMSOFT, pages 89–98, Arizona, USA, 2010.

[58] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 261–270, San Jose, USA, 1987.

[59] G. Lipari and E. Bini. Resource Partitioning Among Real-Time Applications. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 151–158, Porto, Portugal, 2003.

[60] J. W. S. Liu. *Real-Time Systems*. Prentice Hall publishers, 2000.

[61] M. Lluesma, A. Cervin, P. Balbastre, I. Ripoll, and A. Crespo. Jitter Evaluation of Real-Time Control Systems. In *Proceedings of 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Application*, pages 257–260, Sydney, Australia, 2006.

[62] B. Messner and D. Tilbury. Control Tutorials for MATLAB and Simulink. http://www.engin.umich.edu/group/ctm/examples/pend/invpen.html,Visited November. 2014.

[63] A. Mok. Firm Real-time Systems. *ACM Computing Surveys*, 28, 1996.

[64] A. K. Mok and A. Feng. Towards Compositionality in Real-Time Resource Partitioning Based on Regularity Bounds. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 129–138, 2001.

[65] A. Oliveira, A. Azim, S. Fischmeister, R. Marau, and L. Almeida. D-RES: Correct Transitive Distributed Service Sharing. In *Proc. of the Work-in-Progress Session of the*

Conference on Emerging Technologies and Factory Automation (ETFA), Barcelona, Spain, 2014.

[66] M. Pajic, S. Sundaram, G. J. Pappas, and R. Mangharam. The Wireless Control Network: A New Approach For Control Over Networks. *IEEE Transactions on Automatic Control*, 56(10):2305–2318, October 2011.

[67] L. T. X. Phan, S. Chakraborty, and P. S. Thiagarajan. A Multi-Mode Real-Time Calculus. In *Proc. of the 29th IEEE Real-Time Systems Symposium (RTSS)*, Barcelona, Spain, 2008.

[68] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing Analysis of the FlexRay Communication Protocol. *Real-Time Systems*, 39:205–235, August 2008.

[69] D. Potop-Butucaru, A. Azim, and S. Fischmeister. Semantics-Preserving Implementation of Synchronous Specifications Over Dynamic TDMA Distributed Architectures. In *Proc. of the International Conference on Embedded Software (EMSOFT)*, pages 199–208, Scottsdale, Arizona, USA, October 2010.

[70] D. Potop-Butucaru, R. de Simone, Y. Sorel, and J. Talpin. Clock-driven Distributed Real-time Implementation of Endochronous Synchronous Programs. In *Proceedings of the 7th ACM International Conference on Embedded Software (EMSOFT)*, pages 147–156, Grenoble, France, 2009.

[71] H. Rehbinder and M. Sanfridson. Integration of Off-line Scheduling and Optimal Control. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 137–143, Stockholm, Sweden, 2000.

[72] X. Ren, S. Li, Z. Wang, M. Yuan, and Y. Sun. A QoS Management Scheme for Paralleled Networked Control Systems with CAN Bus. In *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 842–847, Busan, Korea, 2004.

[73] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry. Foundations of Control and Estimation over Lossy Networks. *Proc. of the IEEE*, 95:163–187, 2007.

[74] I. Shin, M. Behnam, T. Nolte, and M. Nolin. Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 209–220, Barcelona, Spain, 2008.

[75] I. Shin and I. Lee. Periodic Resource Model for Compositional Real-time Guarantees. In *Real-Time Systems Symposium (RTSS)*, pages 2–13, Cancun, Mexico, 2003.

[76] I. Shin and I. Lee. Compositional Real-Time Scheduling Framework. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 57–67, Lisbon, Portugal, 2004.

[77] I. Shin and I. Lee. Compositional Real-time Scheduling Framework with Periodic Model. *ACM Transactions Embedded Computing Systems*, 7:1–39, May 2008.

[78] I. Shin and I. Lee. Periodic Resource Model for Compositional Real-time Guarantees, 2010. Technical Report.

[79] S. Skogestad and I. Postlewaite. *Multivariable Feedback Control*. Wiley, 1996.

[80] M. Velasco, J.M. Fuertes, C. Lin, P. Marti, and S. Brandt. A Control Approach to Bandwidth Management in Networked Control Systems. In *Proceedings of the 30th Annual Conference of IEEE Industrial Electronics Society (IECON)*, pages 2343–2348, Busan, Korea, 2004.

[81] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet. A Co-design Approach for Embedded System Modeling and Code Generation with UML and MARTE. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 226–231, Nice, France, 2009.

[82] G.C. Walsh and H. Ye. Scheduling of Networked Control Systems. *IEEE Control Systems Magazine*, 21(1):57–65, 2001.

[83] E. Wandeler and L. Thiele. Real-time Interfaces for Interface-based Design of Real-Time Systems with Fixed Priority Scheduling. In *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT)*, pages 80–89, New Jersey, USA, 2005.

[84] E. Wandeler and L. Thiele. Interface-Based Design of Real-Time Systems with Hierarchical Scheduling. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 243–252, San Jose, USA, April 2006.

[85] A. Wassyng, M. Lawford, and T. Maibaum. Separating Safety and Control Systems to Reduce Complexity. In *Conquering Complexity*, pages 85–102. Springer, 2012.

[86] G. Weiss, S. Fischmeister, M. Anand, and R. Alur. Specification and Analysis of Network Resource Requirements of Control Systems. In *In Proc. of the 12th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 381–395, San Fransisco, USA, April 2009.

[87] F. Xia, X. Dai, Z. Wang, and Y. Sun. Feedback Based Network Scheduling of Networked Control Systems. In *Proceedings of International Conference on Control and Automation(ICCA)*, pages 1231–1236, Budapest, Hungary, 2005.

[88] J. Yépez, P. Martí, and J.M. Fuertes. Control Loop Scheduling Paradigm in Distributed Control Systems. In *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 1441–1446, Virginia, USA, 2003.

[89] W. Zhang, M. S. Branicky, and S. M. Phillips. Stability of Networked Control Systems. *IEEE Control Systems Magazine*, 21(1):84–99, 2001.