DOT/FAA/AR-05/27

Office of Aviation Research
and Development
Washington, D.C. 20591

Real-Time Scheduling Analysis

November 2005

Final Report

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

1. Report No. DOT/FAA/AR-05/27	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle REAL-TIME SCHEDULING ANALYSIS		5. Report Date November 2005
		6. Performing Organization Code
7. Author(s) Joseph Leung and Hairong Zhao		8. Performing Organization Report No.
9. Performing Organization Name and Address Department of Computer Science New Jersey Institute of Technology Newark, NJ 07102		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. NJIT/WOW 01/C/AW/NJIT Amendment 2
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Office of Aviation Research and Development Washington, DC 20591		13. Type of Report and Period Covered
		14. Sponsoring Agency Code AIR-120

15. Supplementary Notes

The Federal Aviation Administration Airport and Aircraft Safety R&D Division COTR was Charles Kilgore.

16. Abstract

This project was concerned with scheduling analysis of real-time tasks. It consisted of two major tasks. The first task explored and reported the industry approaches to scheduling real-time tasks and the tools they used in the verification of temporal correctness. To carry out this task, a questionnaire was designed and sent to a number of industry people who are involved in developing software for real-time systems. Their responses were analyzed and conclusions were drawn.

The second task consisted of developing scheduling algorithms and temporal verification tools for a model of periodic, real-time tasks. An optimal scheduling algorithm, called Deadline-Monotonic-with-Limited-Priority-Levels, was developed for a system with a single processor and a limited number of priority levels. A procedure to determine if a given set of periodic, real-time tasks is feasible on one processor with *m* priority levels, where *m* is less than the number of tasks, was also developed.

Two heuristics for a multiprocessor system with a limited number of priority levels were given. Additionally, a conjecture on the processor utilization bound, *U(n)*, below which a set of unit-execution-time tasks is always schedulable was provided. While a complete proof of the conjecture has not been accomplished, it has been demonstrated that it is valid for several special cases.

17. Key Words Periodic, real-time tasks; Rate-monotonic and deadline-monotonic algorithms; Validation procedure; Real-time scheduling		18. Distribution Statement This document is available to the public through the National Technical Information Service (NTIS) Springfield, Virginia 22161.	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 135	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

A/D	Analog to digital controller
ATC	Air Traffic Control (System)
D/A	Digital to analog controller
DM	Deadline-Monotonic (Algorithm)
DM-LPL	Deadline-Monotonic-with-Limited-Priority-Level (Algorithm)
EDF	Earliest-Deadline-First (Algorithm)
FCFS	First-come-first-serve
FF	First-Fit (Algorithm)
FFDU	First-Fit-Decreasing-Utilization (Algorithm)
I/O	Input/Output (Activities)
NP	Nondeterministic Polynomial
RM	Rate-Monotonic (Algorithm)
RMFF	Rate-Monotonic-First-Fit (Algorithm)
RMNF	Rate-Monotonic-Next-Fit (Algorithm)
RTSA	Real-Time Scheduling Analysis
TS	Task system

EXECUTIVE SUMMARY

Real-time (computing, communication, and information) systems have become increasingly important in every day life. A real-time system is required to complete its work and deliver its services on a timely basis. Examples of real-time systems include digital control, command and control, signal processing, and telecommunication systems. Such systems provide important and useful services to society on a daily basis. For example, they control the engine and brakes on cars; regulate traffic lights; schedule and monitor the takeoff and landing of aircraft; enable aircraft control system; monitor and regulate bodily functions (e.g., blood pressure and pulse); and provide up-to-date financial information.

Many real-time systems are embedded in sensors and actuators and function as digital controllers. Typically, in these type of applications, signals arrive periodically at fixed periods. When the signals arrive, they must be processed before the arrival of the next batch of signals. Real-time tasks can be classified as hard real-time or soft real-time. Hard real-time tasks are those that require a strict adherence to deadline constraints, or else the consequence is disastrous. By contrast, soft real-time tasks are those that do not require a strict adherence to deadline constraints, but it is desirable to do so. Most real-time systems have a combination of both hard and soft real-time tasks.

Hard real-time tasks create another dimension in verifying their correctness. Not only do their logical correctness need to be verified (i.e., the program implements exactly the things it is supposed to do), their temporal correctness must also be verified (i.e., all deadlines are met). A hard real-time task must be both logically and temporally correct for it to be usable. Since a hard real-time task is executed periodically during its entire operational time and since the period of every task is very small compared to the duration of its operation, the schedule can be regarded as an infinite schedule for all practical purposes. Verifying the temporal correctness of an infinite schedule is a challenging problem since there are infinitely many deadlines to check. One of the main goals of this project is to develop tools to solve this verification problem.

This project consists of two major jobs. The first job was to explore and report the industry approaches to scheduling real-time tasks and the tools they use in the verification of temporal correctness. A questionnaire was developed and sent to a number of industry representatives who are involved in developing software for real-time systems. Based on their responses, some conclusions were drawn, which are described in this report.

The second job consisted of developing scheduling algorithms and temporal verification tools for a model of periodic, real-time tasks. An optimal scheduling algorithm, called Deadline-Monotonic-with-Limited-Priority-Levels, was developed for a system with a single processor and a limited number of priority levels. As a byproduct of the work on the Deadline-Monotonic-with-Limited-Priority-Levels algorithm, a procedure to determine if a given set of periodic, real-time tasks is feasible on one processor with *m* priority levels, where *m* is less than the number of tasks, was also developed.

This report begins with a summary of the industry survey results, then the three approaches that were used to schedule a real-time task system are discussed: (1) Clock-Driven, (2) Processor-Sharing, and (3) Priority-Driven. It was reasoned that the Priority-Driven approach is far

superior to the Clock-Driven and Processor-Sharing approaches. The report then reviews the literature on Priority-Driven scheduling algorithms, which can be divided into two categories: Dynamic-Priority and Fixed-Priority. While Dynamic-Priority scheduling algorithms are more effective than Fixed-Priority scheduling algorithms, they are rarely used in practice because of the overhead involved. Therefore, the report concentrates on Fixed-Priority scheduling algorithms.

The Deadline Monotonic algorithm is an optimal Fixed-Priority scheduling algorithm for one processor. Unfortunately, the algorithm assumes that the number of priorities is the same as the number of real-time tasks. In practice, one can only have a limited number of priorities, say *m*, supported by a system. Under this scenario, the Deadline Monotonic algorithm fails to be optimal, and as a result of the work to find an optimal scheduling algorithm, the Deadline-Monotonic-with-Limited-Priority-Levels algorithm was developed, along with a procedure to check if a given set of real-time tasks is feasible on one processor with *m* priority levels.

The same problem was explored for multiprocessor systems. It was demonstrated that finding an optimal assignment is strongly nondeterministic polynomial (NP)-hard, which is tantamount to showing that there is no efficient algorithm to solve this problem. Motivated by the computational complexity, several heuristics for solving this problem are suggested.

The minimum processor utilization, *U(n)*, for a set of *n* unit-execution-time tasks, was also studied. *U(n)* is the threshold for the total processor utilization of the *n* tasks, below which they are always schedulable. It is conjectured that

$$U(n) = \tfrac{1}{n-1} + \tfrac{1}{n} + \tfrac{1}{n+1} + \ldots + \tfrac{1}{2n-3} + \tfrac{1}{2n}\,.$$

Some special cases of this conjecture are proven, but a complete proof failed to be solvable.

Some other jobs were planned for this research effort (i.e., study of fault-tolerant issues, which are concerned with schedulability analysis when there are time losses due to transient hardware or software failures, and study of CPU scheduling coupled with I/O activities). However, due to time constraints, significant progress was not made in these areas.

1. INTRODUCTION.

1.1 PURPOSE AND BACKGROUND.

Real-time computing, communication, and information systems have become increasingly important in every day life. A real-time system is required to complete its work and deliver its services on a timely basis. Examples of real-time systems include digital control, command and control, signal processing, and telecommunication systems. Such systems provide important and useful services to society on a daily basis. For example, they control the engine and brakes on cars; regulate traffic lights; schedule and monitor the takeoff and landing of aircraft; enable aircraft control systems; monitor and regulate bodily functions (e.g., blood pressure and pulse); and provide up-to-date financial information.

Many real-time systems are embedded in sensors and actuators and function as digital controllers. An example of a digital controller, taken from Liu [1] is shown in figure 1-1. The term plant in the figure refers to a controlled system (such as an engine, a brake, an aircraft, or a patient), A/D refers to analog-to-digital converter, and D/A refers to digital-to-analog converter. The state of the plant is monitored by sensors and can be changed by actuators. The real-time (computing) system estimates from the sensor readings the state of the plant, *y(t)*, at time *t* and computes a controlled output, *u(t)*, based on the difference between the current state and the desired state (called reference input in the figure), *r(t)*. This computation is called the control-law computation in the figure. The output generated by the control-law computation activates the actuators, which bring the plant closer to the desired state.

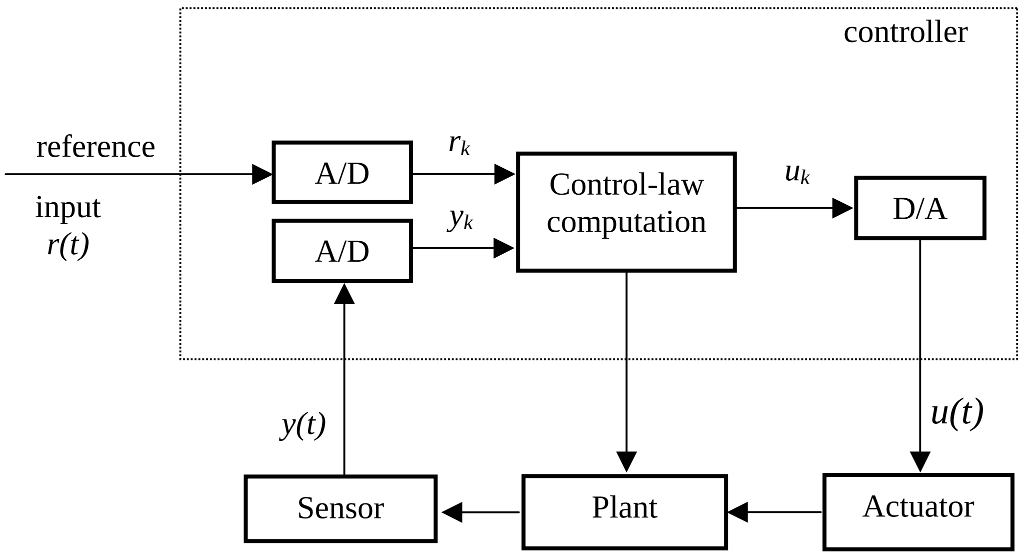

FIGURE 1-1. DIGITAL CONTROLLER

In figure 1-1, *r(t)* and *y(t)* are sampled periodically every sampling period, *T* units of time. Therefore, the control-law computation needs to be done periodically every *T* units of time. For each sampled data, the computation must be completed within *T* units of time, or else it will be erased by the next sampled data. Each computation is fairly deterministic in the sense that the maximum execution time can be estimated fairly accurately.

A plant typically has more than one state variable; e.g., the rotation speed and temperature of an engine. Therefore, it is controlled by multiple sensors and by multiple actuators. Because different state variables may have different dynamics, the sampling periods may be different. As an example, also taken from Liu [1], figure 1-2 shows the software structure of a flight controller. The plant is a helicopter, which has three velocity components: forward, side-slip, and altitude rates, which together are called collective in the figure. It also has three rotational (angular) velocities, referred to as roll, pitch, and yaw. The system uses three sampling rates: 180, 90, and 30 Hz; i.e., the sampling periods are 1/180, 1/90, and 1/30 seconds, respectively.

Do the following in each 1/180 –seconds cycle:

- Validate sensor data and select data source; in the presence of failures, reconfigure the system.
- Do the following 30-Hz avionics tasks, each every six cycles:
  - ➢ keyboard input and mode selection
  - ➢ data normalization and coordinate transformation
  - ➢ tracking reference update
- Do the following 30-Hz computations, each once every six cycles:
  - ➢ control laws of the outer pitch-control loop
  - ➢ control laws of the outer roll-control loop
  - ➢ control laws of the outer yaw- and collective-control loop
- Do the following 90-Hz computations once every two cycles, using outputs produced by 30-Hz computations and avionics tasks as input:
  - ➢ control laws of the inner pitch-control loop
  - ➢ control laws of the inner roll- and collective-control loop
- Compute the control laws of the inner yaw-control loop, using outputs produced by 90-Hz control-law computations as input.
- Output commands.
- Carry out built-in-test.
- Wait until the beginning of the next cycle.

FIGURE 1-2. SOFTWARE CONTROL STRUCTURE OF A FLIGHT CONTROLLER

The above controller controls only flight dynamics. The control system on board an aircraft is considerably more complex. It typically contains many other equally critical subsystems (e.g., air inlet, fuel, hydraulic, and anti-ice controllers) and many noncritical subsystems (e.g., compartment lighting and temperature controllers). So, in addition to the flight control-law computations, the system also computes the control laws of these subsystems.

Controllers in a complex monitor and control system are typically organized hierarchically. One or more digital controllers at the lowest level directly control the physical plant. Each output of a higher-level controller is a reference input of one or more lower-level controllers. One or more of the higher-level controller interfaces with the operator(s).

Figure 1-3, also taken from Liu [1], shows the hierarchy of flight control, avionics, and air traffic control (ATC) systems. The ATC system is at the highest level. It regulates the flow of flights to each destination airport. It does so by assigning to each aircraft an arrival time at each metering fix en route to the destination: The aircraft is supposed to arrive at the metering fix at the assigned arrival time. At any time while in flight, the assigned arrival time to the next metering fix is a reference input to the onboard flight management system. The flight management system chooses a time-referenced flight path that brings the aircraft to the next metering fix at the assigned arrival time. The cruise speed, turn radius, descend/ascend rates, and so forth required to follow the chosen time-referenced flight path are the reference inputs to the flight controller at the lowest level of the control hierarchy.

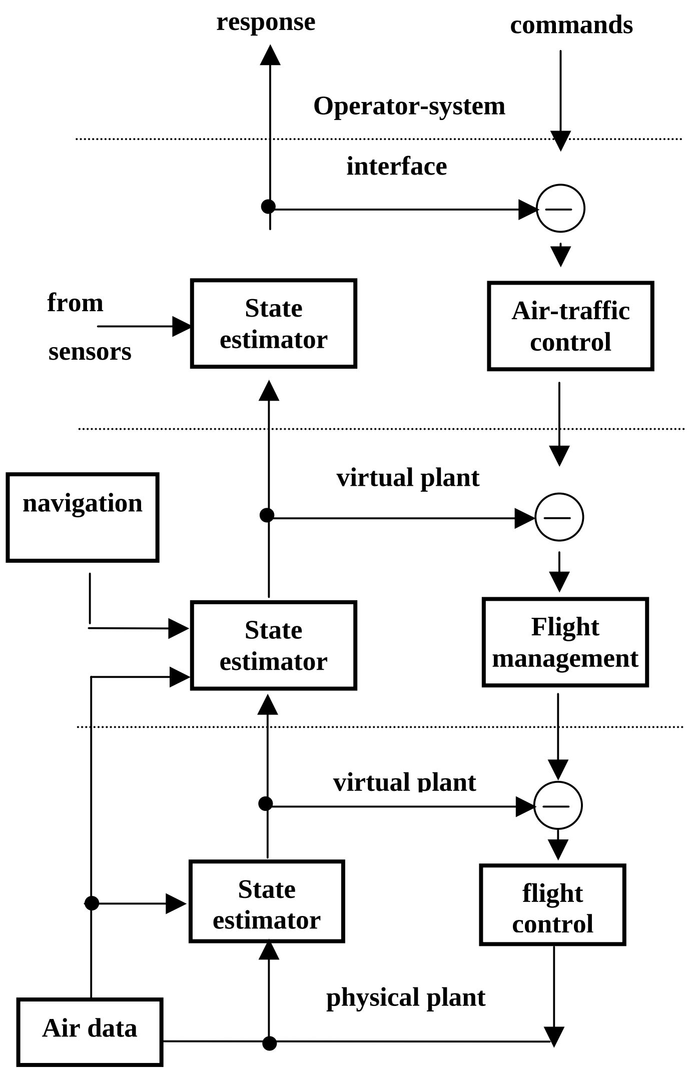

FIGURE 1-3. AIR TRAFFIC AND FLIGHT CONTROL HIERARCHY

Real-time tasks can be classified as hard or soft. Hard real-time tasks are those that require a strict adherence to deadline constraints, or else the consequence is disastrous. An example of a hard real-time task is the flight controller shown in figure 1-2. By contrast, soft real-time tasks are those that do not require a strict adherence to deadline constraints, but it is desirable to do so. An example of a soft real-time task is the controller that controls the compartment lighting and temperature in an aircraft. Most real-time systems have a combination of both hard and soft real-time tasks.

Hard real-time tasks create another dimension in validating their correctness. Not only do their logical correctness need to be verified (i.e., the program implements exactly the things it is supposed to), their temporal correctness must also be verified (i.e., all deadlines are met). A hard real-time task must be both logically and temporally correct for it to be usable. Since a hard real-time task is executed periodically during its entire operational time and since the period of every task is very small compared to the duration of its operation, the schedule can be regarded as an infinite schedule for all practical purposes. Verifying the temporal correctness of an infinite schedule is a challenging problem since there are infinitely many deadlines to check. One of the main goals of this project is to develop tools to help solve this verification problem. Since the main concern of this report is hard real-time tasks, the term hard real-time task will simply be called a real-time task throughout this report.

This project consisted of two major jobs. The first job was to explore and report the industry approaches to scheduling real-time tasks and the tools they use in the verification of temporal correctness. A questionnaire was developed and sent to a number of industry representatives who are involved in developing software for real-time systems. Based on their responses, some conclusions were drawn, these are described in this report.

The second job consisted of developing scheduling algorithms and temporal verification tools for a model of periodic, real-time tasks. An optimal scheduling algorithm, called Deadline-Monotonic-with-Limited-Priority-Levels (DM-LPL), was developed for a system with a single processor and a limited number of priority levels. As a byproduct of the DM-LPL algorithm, a procedure to determine if a given set of periodic, real-time tasks is feasible on one processor with m priority levels, where m is less than the number of tasks, was also developed.

A periodic, real-time task, T_i, is characterized by the quadruple (s_i, e_i, d_i, p_i), where s_i is the initial request time, e_i is the execution time, d_i is the relative deadline, and p_i is the period. In this characterization, T_i makes an initial request at time s_i, and thereafter at times $s_i + kp_i$, $k = 1, 2, \ldots$ The k-th request requires e_i units of execution time, and it must be completed no later than the deadline $s_i+(k\text{-}1)p_i + d_i$. A real-time task system consists of n periodic, real-time tasks, and is denoted by $TS = (\{T_i, \{s_i\},\{e_i\}, \{d_i\}, \{p_i\})$.

A schedule S for a real-time task system TS is said to be valid if the deadline of each request of each task is met. Since the schedule is infinite, checking if the schedule is valid is a nontrivial problem. TS is feasible if there is a valid schedule for it. TS is schedulable by a particular scheduling algorithm if the scheduling algorithm produces a valid schedule for it. A scheduling algorithm is said to be optimal if every feasible task system is schedulable by the scheduling algorithm.

1.2 REPORT OVERVIEW.

This report begins with a summary of the industry survey results. Three approaches that have been used to schedule a real-time task system are (1) Clock-Driven, (2) Processor-Sharing, and (3) Priority-Driven. It was reasoned that the Priority-Driven approach was far superior to the Clock-Driven and Processor-Sharing approaches. The report then reviews the literature on Priority-Driven scheduling algorithms, which can be divided into two categories: Dynamic-Priority and Fixed-Priority. While Dynamic-Priority scheduling algorithms are more effective than Fixed-Priority scheduling algorithms, they are rarely used in practice because of the overhead involved. Therefore, the report concentrates on Fixed-Priority scheduling algorithms.

The remaining portions of the report focus on specific scheduling model. Leung and Whitehead have shown that the Deadline Monotonic algorithm is an optimal Fixed-Priority scheduling algorithm for one processor. Unfortunately, the algorithm assumes that the number of priorities is the same as the number of real-time tasks. In practice, one can only have a limited number of priorities, say *m*, supported by a system. Under this scenario, the Deadline Monotonic (DM) algorithm fails to be optimal, and the optimal scheduling algorithm DM-LPL was developed, along with a separate procedure to check if a given set of real-time tasks is feasible on one processor with *m* priority levels.

The same problem is explored for multiprocessor systems. It is demonstrated that finding an optimal assignment is strongly nondeterministic polynomial (NP)-hard, which is tantamount to showing that there is no efficient algorithm to solve this problem. A problem *Q* is NP-hard if all problems in the NP-class are reducible to *Q*. Motivated by the computational complexity, several heuristics for solving this problem are suggested.

The minimum processor utilization, $U(n)$, for a set of *n* unit-execution-time tasks is also studied. $U(n)$ is the threshold for the total processor utilization of the *n* tasks, below which they are always schedulable. It is conjectured that

$$U(n) = \frac{1}{n-1} + \frac{1}{n} + \frac{1}{n+1} + \ldots + \frac{1}{2n-3} + \frac{1}{2n}$$

Some special cases of this conjecture are proven, but time was not available to perform the complete proof.

Some other tasks were planned for this research effort (i.e., study of fault-tolerant issues, which are concerned with schedulability analysis when there are time losses due to transient hardware or software failures, and study of central processing unit (CPU) scheduling coupled with input/output (I/O) activities). However, due to time constraints, significant progress was not made in these areas. These remain topics to be considered in future research.

1.3 USING THIS REPORT.

There are a number of potential uses for this report. Since this research task falls more into the category of basic research than many of the Federal Aviation Administration (FAA) Software and Digital System Safety Project research and development initiatives, some explanation of how various readers may use the report is provided. The intended audience for this report is certification authorities, industry representatives, and researchers. A brief summary of how each audience can use this report is listed below.

- Certification Authorities. Certification authorities will primarily benefit from the summary of the industry survey, the tutorial of the different scheduling approaches, and the research results (sections 2, 3, and 5). Additionally, certification authorities might desire to browse section 4 and appendix B for information purposes, realizing that these sections are research-focused and will require significant work before they can be implemented in an actual aircraft project.

- Industry Representative. The industry can benefit from this entire report but should realize that section 4 and appendix B are at a research stage. The proofs will require verification by a qualified independent entity before they can be implemented in an actual aviation project. It is also likely that the industry would need to develop tools to help implement the algorithms of this report into a usable format.

- Researchers. As mentioned before and discussed throughout this report, this research effort is really the beginning of what needs to be done before implementing the algorithms. Researchers will likely benefit most from section 4 and appendix B, and will likely want to build upon these in additional work. Section 5 provides specific information about where the future research needs to go.

2. SURVEY OF INDUSTRY APPROACHES.

A Real-Time Scheduling Analysis (RTSA) Questionnaire was sent out to industry representatives who are involved in developing software for real-time systems. The questionnaire is shown in appendix A. Fifteen questionnaires were returned and are tabulated in appendix A.

Of the 15 respondents, the majority (12) of them work for avionics or engine control developers or aircraft or engine manufacturers. The majority either verify/test real-time scheduling performance, perform RTSA on aviation system projects, or develop real-time operating systems (RTOS) that support RTSA. On the whole, the respondents had the appropriate background to answer this questionnaire. The responses and questions are summarized below.

Question A.2.1 asked, "What type of events are typically used to trigger time-critical functions of your real-time system (e.g., interrupts, data message queries, data refresh rates, (pilot) user input, change of state, certain conditions, display refresh rates, etc.)?" Interrupts were mentioned by ten respondents as the main events that are typically used to trigger the time-critical functions of their real-time systems. This fits well with the DM algorithm, which is essentially interrupt-driven.

Question A.2.2 asked, "What are typical performance requirements that your real-time system must meet?" The majority of the respondents mentioned that critical tasks must meet hard real-time deadline constraints. The response times mentioned are from a few milliseconds to hundreds of milliseconds. This justifies the study of scheduling analysis of hard real-time tasks, which is the main topic in this project.

Question A.2.3 asked, "Where are your performance requirements for time-critical functions typically defined (e.g., system requirements or interface control documents, software requirements document)?" System requirements, interface requirements, and software requirements are the most popular responses. It appears that performance requirements for time-critical functions are typically defined in those documents.

Question A.2.4 asked, "How do you distinguish time-critical functions from other functions in your application?" The majority of the respondents answer that time-critical functions are explicitly stated in the requirement.

Question A.2.5 asked, "Do your time-critical functions have dependencies on hardware or shared hardware devices (central processing unit, memory, data buses, I/O ports, queues, etc.) with other software functions of your application or other applications resident in the system? If yes, please explain." The answers were mixed. Some say that there are no dependencies, while others say that there are. The results are inconclusive.

Question A.2.6 asked, "What are some mechanisms that your application (software and hardware) uses to ensure that "time-critical triggers" get handled at the appropriate priority and gain the relevant resources to ensure that your performance requirements are achieved?" Priority levels assigned to interrupts were mentioned by several people as the mechanism used to schedule time-critical tasks. Some people mentioned that the computers they use have only one priority level, such as the PowerPC. This fits well with the model that the system has a limited number (m) of priority levels (as proposed for this research project). In this case, m = 1.

Question A.2.7 asked, "What type of reviews, analyses and testing does your team use to ensure that time-critical functions will satisfy their performance requirements, especially in worst-case condition scenarios?" The majority of respondents mentioned that they can obtain worst-case execution time by analyzing the code. As for validation, most respondents use emulator or some ad hoc approach to test. This is risky because an emulator can only show that it will work most of the time. It does not show that it will work all of the time. There is a definite need for a formal validation procedure which gives a guarantee that it will work all the time.

Question A.3.1 asked, "What approaches to message passing have your projects utilized?" The answers are so different that it was difficult to draw any conclusions. It seems that message passing mechanism is a function of the hardware and operating systems used by the organization. This explains the diverse answers.

Question A.3.2 asked, "Do your messages communicate with each other? If yes, please explain how." The majority answered that messages do not communicate with each other.

Question A.4.1 asked, "What type of processors have you used for your systems?" The majority of respondents currently use Intel processors; however, PowerPC seems to be gaining momentum. A small number of respondents use Motorola or TI chips. By far, the largest number use Intel family of processors.

Question A.4.2 asked, "Have you found any peculiarities with any of the processors that affect the real-time scheduling analysis? If yes, please explain the peculiarities and how they were addressed." The answers to this question varied significantly. Some pointed out that the lack of multiple interrupt priorities in PowerPC makes it difficult to schedule real-time tasks. This confirms the hypothesis of this research effort that more priority levels make scheduling easier. Some mentioned that the cache memory makes it difficult to analyze the worst-case running time, since the execution time depends on the hit ratio of the cache. Some mentioned that the pipeline processor also makes it difficult to estimate the worst-case running time.

Question A.4.3 asked, "Do your systems use a single processor or multiple processors? If multiple processors, how is the system functionality distributed and handled across processors?" Nine respondents said that they use a single processor while six respondents said that they use multiple processors. One respondent mentioned that they use both single and multiple processors. It seems that they are about evenly divided, with the single processor having a slight edge.

Question A.5.1 asked, "What scheduling algorithms/approaches have you used to schedule your system tasks at run time? Please match the algorithm (e.g., preemptive priority, round robin, etc.) with the system type (e.g., display, communication, navigation, etc.)." The majority responded that they use pre-emptive priority scheduling algorithm, of which the DM algorithm is a member. One respondent mentioned that they use Rate Monotonic (RM) Analysis.

Question A.5.2 asked, “If you used priority scheduling, how many priorities levels were assigned? How was priority inversion avoided? How did the number of priority levels compare to the number of processes?” The majority of the respondents said that they used 3 to 20 levels. One can conclude that the number of priority levels is relatively small, compared to the number of real-time tasks in the system.

Question A.5.3 asked, “What kind of scheduling problems have you encountered in multitasking systems and how were they addressed?” A fair number of respondents did not comment on this question. Therefore, it was not possible to draw any valid conclusions.

Question A.5.4 asked, “Have you used real-time operating systems to support your schedule guarantees? If yes, what kind of operating systems have you used and what kind of scheduling challenges have you encountered?” Most respondents said that they did not use real-time operating systems to support their schedule guarantees. For the few who said they did, they used in-house proprietary systems. It seems that there is a learning curve here. If the tools are made available to them free of charge, they may in fact use these tools in the future.

Question A.5.5 asked, “Do you verify what data gets dumped, due to priority settings and functions getting preempted? If yes, how does it affect your system?” Most respondents replied No or N/A. Therefore, there was insufficient data to draw any conclusions.

Question A.5.6 asked, “Do you use tools to assist in the real-time scheduling analysis? If yes, what kind of tools? How are the outputs of these tools verified?” Most respondents replied No or that they use emulators and simulators. This can create problems since these are not rigorous and formal analyses.

Question A.5.7 asked, “What trends in commercial aviation systems do you think will challenge the current scheduling approaches (i.e., may lead to the need for new scheduling algorithms)?” Some said that multiple thread real-time deadline scheduling analysis will be the future trends that challenge the current scheduling approaches. Some said that the desire to reuse, the desire to inherit confidence from reuse, and the desire to use nondevelopmental items will be the major challenges to the current scheduling approaches. These comments point to the importance of a theory of scheduling on multiple processors.

Question A.6.1 asked, “After system development, do you verify that deadlines are met and scheduling analysis assumptions are correct? If yes, please explain how.” The majority of the respondents said that they verified that deadlines are met and scheduling analysis assumptions are correct after system development.

Question A.6.2 asked, “In what areas of timing verification or validation have you encountered problems and how were they addressed?” The answers were so diverse that it was difficult to draw any conclusions. It seems that the problems encountered is highly dependent on the specific problems and the hardware or software used in the company.

Question A.7.1 asked, “Does your testing allow for faults? If yes, please explain.” Most respondents said that their testing allow for faults. This is mostly handled by injecting faults into the system and checking to see how the system responds to the faults. Responses indicated that no worst-case analysis is done; i.e., it is mostly done in an ad hoc manner.

Question A.8.1 asked, "In your opinion, what are the major issues regarding RTSA and its verification?" Responses varied significantly and are summarized below.

- Confirmation of timing issues under all foreseeable circumstances is a major issue regarding RTSA and its verification.
- Testing is difficult when modifications and/or changes are made.
- The tools are very expensive and not always available.
- The analysis tends to be intuitive and lacks formal analysis.
- The operating systems are so general that they are of little use in dealing with real-time systems.

From the responses of the questionnaire, the following conclusions were drawn:

- There is a need for scheduling analysis and verification in the avionics industry.
- The current practice is by ad hoc methods. Tools are seldom used either because they are expensive and not available, or the operating systems are for general purpose and not usable for real-time systems.
- The trend is towards multiprocessor systems.
- Software developers do test for fault tolerance, but the main method used is by means of fault injection which is rather ad hoc.

It was concluded that developing defined approaches and algorithms for scheduling, deadline verification, and fault tolerance will significantly help the avionics industry. Furthermore, these theories should be implemented into a software tool suite that can be made available to anyone who desires to use it. As more and more people use these tools (which may need to be qualified), future systems will be less error-prone and easy to maintain and modify.

3. TUTORIAL ON DIFFERENT SCHEDULING APPROACHES.

Whether a set of real-time tasks can meet all their deadlines depends on the characteristics of the tasks (e.g., periods and execution times) and the scheduling algorithms used. Scheduling algorithms can be classified as pre-emptive and non-pre-emptive. In non-pre-emptive scheduling, a task once started must be executed to completion without any interruptions. By contrast, pre-emptive scheduling permits suspension of a task before it completes, to allow for execution of another more critical task. The suspended task can resume execution later on from the point of suspension. While pre-emptive scheduling incurs more system overhead (e.g., context switching time due to pre-emptions) than non-pre-emptive scheduling, it has the advantage that processor utilization (the percentage time that the processor is executing tasks) is significantly higher than non-pre-emptive scheduling. For this reason, most of the scheduling algorithms presented in the literature are pre-emptive scheduling algorithms.

There are three major approaches in designing pre-emptive scheduling algorithms for real-time tasks: Clock-Driven, Processor-Sharing, and Priority-Driven. Each approach is described below.

The Clock-Driven approach is the oldest method used to schedule real-time tasks. In this method, a schedule is handcrafted and stored in memory before the system is put in operation. At run time, tasks are scheduled according to the scheduling table. After the scheduler dispatches a task, it will set the hardware timer to generate an interrupt at the next task switching time. The scheduler will then go to sleep until the timer expires. This process is repeated throughout the whole operation.

The Clock-Driven approach has several disadvantages that render it undesirable to use: (1) it requires a fair amount of memory to store the scheduling table; (2) a slight change in task parameters (e.g., execution time and period) requires a complete change of the scheduling table, which can be very time-consuming, and (3) this approach is not adaptive to any change at run time. For example, if a system fault occurs or a task runs for less (or more) time than predicted, it is not clear how the scheduling decisions can be adapted to respond to the change.

The Processor-Sharing approach is to assign a fraction of a processor to each task, depending on the utilization factor (execution time divided by period) of the task. Since a processor cannot be used by more than one task at the same time, the processor sharing is approximated by dividing a time interval into smaller time slices and giving each task an amount proportional to the fraction of processor assigned to the task. For example, if a task is assigned 0.35 of a processor, then the task would receive 35 percent of the time slices in the time interval.

The time slice has to be made very small to obtain a close approximation of processor sharing. But when the time slice is very small, a significant amount of time will be spent in context switching. This is a major drawback of the Processor-Sharing approach.

In the Priority-Driven approach, each task is assigned a priority. At run time, the ready task that has the highest priority will receive the processor for execution. Priorities can be assigned at run time (Dynamic-Priority) or fixed at the beginning before the operation starts (Fixed-Priority). Fixed-Priority scheduling algorithms incur far less system overhead (context switching time) than Dynamic-Priority scheduling algorithms, since the scheduler does not need to determine the priority of a task at run time. Furthermore, Fixed-Priority scheduling algorithms can be implemented at the hardware level by attaching the priority of a task to the hardware interrupt level. On the other hand, processor utilization under Fixed-Priority scheduling algorithms is usually not as high as Dynamic-Priority scheduling algorithms. It is known that Fixed-Priority scheduling algorithms may yield a processor utilization as low as 70 percent, while Dynamic-Priority scheduling algorithms may yield a processor utilization as high as 100 percent.

One of the most well-known Dynamic-Priority scheduling algorithm is the Earliest-Deadline-First (EDF) algorithm, which assigns the highest priority to the task whose deadline is closest to the current time. It is known that EDF is optimal for one processor [2 and 3], in the sense that any set of tasks that can be feasibly scheduled by any Dynamic-Priority scheduling algorithms can also be feasibly scheduled by EDF. However, EDF is not optimal for two or more processors [4]. At the present time, no scheduling algorithm is known to be optimal for two or more processors.

The two most well known Fixed-Priority scheduling algorithms are the Rate-Monotonic (RM) and Deadline-Monotonic (DM) algorithms [3 and 5]. RM assigns the highest priority to the task with the smallest period (or equivalently, the highest request rate), while DM assigns the highest priority to the task with the smallest relative deadline. It should be noted that DM and RM are identical if the relative deadline of each task is identical to its period. Leung and Whitehead [5] have shown that DM is optimal for one processor, in the sense that any set of tasks that can be feasibly scheduled by any Fixed-Priority scheduling algorithms can also be feasibly scheduled by DM. Liu and Layland [3] have shown that RM is optimal when the relative deadline of each task coincides with its period; it fails to be optimal if the deadline of some task is not identical to its period. Both DM and RM fail to be optimal for two or more processors [6]. At the present time, no scheduling algorithm is known to be optimal for two or more processors.

The following documents are specifically recommended to further describe the scheduling approaches, and also see references 1-6.

- S. Natarajan, ed., (1995), *Imprecise and Approximate Computation*, Kluwer, Boston.
- A.M. Van Tilborg and G.M. Koob, eds., *Foundations of Real-Time Computing: Scheduling and Resources Management*, Kluwer, Boston.

4. DESCRIPTION OF SCHEDULING MODEL.

The research topics in this project were planned to be (1) priority assignment, (2) multiprocessor scheduling, (3) fault tolerant issue, and (4) I/O activities. However, because of time limitations, only the first two topics were studied. In this report, the scheduling model for the first two topics is defined.

A periodic, real-time task, T_i, is characterized by a quadruple (s_i, e_i, d_i, and p_i), where s_i is the initial request time, e_i is the execution time, d_i is the relative deadline, and p_i is the period. In this characterization, T_i makes an initial request at time s_i, and thereafter at times $s_i + kp_i$, $k = 1, 2, \ldots$ The k^{th} request requires e_i units of execution time and it must be completed no later than the deadline $s_i + (k\text{-}1)p_i + d_i$. A real-time task system consists of n periodic, real-time tasks, and is denoted by $TS = (\{ T_i \}, \{ s_i \}, \{ e_i \}, \{ d_i \}, \{ p_i \})$.

A schedule S for a real-time task system TS is said to be valid if the deadline of each request of each task is met. Since the schedule is infinite, checking if the schedule is valid is a non-trivial problem. TS is said to be feasible if there is a valid schedule for it. TS is said to be schedulable by a particular scheduling algorithm if the scheduling algorithm produces a valid schedule for it. A scheduling algorithm is said to be optimal if every feasible task system is schedulable by the scheduling algorithm.

With respect to the above model, there are several important questions whose answers are essential in validating the temporal correctness of a real-time task system. First, how does one determine if a real-time task system is feasible? Second, how does one determine if a real-time task system is schedulable by a particular scheduling algorithm? Third, what are the optimal scheduling algorithms? By definition, a real-time task system is feasible if and only if it is schedulable by an optimal scheduling algorithm. Thus, these three questions are interrelated.

There are several important assumptions associated with this model. First, e_i is assumed to be the maximum execution time required by T_i. At run time, it is assumed that T_i never requires more than e_i units of execution time at each request, although it could use less time. Second, it is assumed that context switching time is negligible. If this is not a valid assumption, e_i must be adjusted to account for the time loss due to context switching. Third, the minimum time lapse between two consecutive requests of T_i is p_i. At run time, the time lapse between two consecutive requests is at least p_i; it could be more than p_i, but not less. Fourth, the relative deadline of each request is d_i. At run time, the relative deadline of each request can be longer than d_i but not shorter. These assumptions must be strictly adhered to in order for the theory to work.

5. RESULTS AND FUTURE WORK.

5.1 RESULTS.

This project consisted of two major jobs. The first job was to explore and report the industry approaches to scheduling real-time tasks and the tools they use in the verification of temporal correctness. A questionnaire was developed and sent to a number of industry representatives who were involved in developing software for real-time systems.

From the responses of the questionnaire, the following conclusions can be drawn:

- There is a need for scheduling analysis and verification in the avionics industry.
- The current practice is by ad hoc methods. Tools are seldom used either because they are expensive and not available, or the operating systems are for general purpose and not usable for real-time systems.
- The trend is towards multiprocessor systems.
- Software developers do test for fault tolerance, but the main method used is by means of fault injection which is rather ad hoc.

It was concluded that developing defined approaches and algorithms for scheduling, deadline verification, and fault tolerance will significantly help the avionics industry. Furthermore, these theories should be implemented into a software tool suite that can be made available to anyone who desires to use it. As more and more people use these tools (which may need to be qualified), future systems will be less error-prone and easy to maintain and modify.

The second job consisted of developing scheduling algorithms and temporal verification tools for a model of periodic, real-time tasks.

The report began with a discussion of the three approaches that have been used to schedule a real-time task system: (1) Clock-Driven, (2) Processor-Sharing, and (3) Priority-Driven. It was reasoned that the Priority-Driven approach is far superior to the Clock-Driven and Processor-Sharing approaches. The report then reviewed the literature on Priority-Driven scheduling algorithms, which can be divided into two categories: Dynamic-Priority and Fixed-Priority. While Dynamic-Priority scheduling algorithms are more effective than Fixed-Priority scheduling algorithms, they are rarely used in practice because of the overhead involved. Therefore, the report concentrated on Fixed-Priority scheduling algorithms.

Priority-Driven scheduling is probably the most appropriate approach in scheduling periodic, real-time tasks. In this project, Fixed-Priority scheduling algorithms for computing systems with limited priority levels were studied. A procedure to test if a given priority assignment is valid (i.e., all deadlines are met) was developed. Furthermore, an optimal priority assignment algorithm, DM-LPL, for one processor was given. The algorithm was implemented using the C language and is shown in appendix C.

For multiprocessors, the problem of finding the minimum number of processors with *m* priority levels to schedule a set of tasks was shown to be NP-hard. Two heuristics were provided, FF and FFDU, for this problem. The special delivery case where each task's execution time is one unit was also considered. Under this model, an attempt was made to develop a utilization threshold, *U(n)*, below which a set of *n* tasks is always schedulable. For the unlimited priority levels, it was conjectured that $U(n) = \left(\sum_{i=n-1}^{2n-3} \frac{1}{i} \right) + \frac{1}{2n}$, which is better than the bound of $n(2^{1/n} - 1)$ given by Liu and Layland [3]. This conjecture was proved for two special cases.

5.2 FUTURE WORK.

There are four areas where future work should be performed:

- Develop a full proof of the conjecture started in this report.

  – The special case when d_1 and d_2 are arbitrary has been proved. It remains to be shown that the conjecture is valid when d_3, d_4, …, d_{n-1} are also arbitrary.

- Perform an independent verification of the algorithms in the report, as well as a verification of the full proof.

  – The results obtained in this report should be reviewed by an independent expert in scheduling theory.

- Perform a study of fault-tolerant issues, which are concerned with schedulability analysis when there are time losses due to transient hardware/software failures.

  – A major issue in this area is to characterize the worst-case scenario due to time losses.

- Perform a study of central processing unit (CPU) scheduling coupled with input/output (I/O) activities.

  – The main issue in this area is to couple pre-emptive scheduling (CPU scheduling) with non-pre-emptive scheduling (I/O activities).

6. REFERENCES.

1. Liu, J.W.S., *Real-Time Systems*, Prentice Hall, New Jersey, 2000.

2. Labetoulle, J., "Some Theorems on Real-Time Scheduling," in *Computer Architecture and Networks*, E. Gelenbe and R. Mahl, eds., North-Holland, Amsterdam, 1974.

3. Liu, C.L., and Layland, J.W., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. of ACM*, Vol. 20, 1973, pp. 46-61.

4. Leung, J.Y-*T*., "A New Algorithm for Scheduling Periodic, Real-Time Tasks," *Algorithmica*, Vol. 4, 1989, pp. 209-219.

5. Leung, J.Y-*T*. and Whitehead, J., "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation*, Vol. 2, 1982, pp. 237-250.

6. Dhall, S.K. and Liu, C.L., "On a Real-Time Scheduling Problem," *Operations Research*, Vol. 26, 1978, pp. 127-140.

APPENDIX A—INDUSTRY SURVEY AND RESPONSES

The following is the Real-Time Scheduling Analysis (RTSA) Questionnaire Analysis

A.1 BACKGROUND QUESTION.

A.1.1 BACKGROUND QUESTION #1.

What kind of organization do you work for?

- ❑ Avionics or engine control developer
- ❑ Aircraft or engine manufacturer
- ❑ Communications, navigation, or surveillance system developer for air traffic management
- ❑ Software tool developer
- ❑ Third party software developer (e.g., operating system or library)
- ❑ Consultant
- ❑ Federal Aviation Administration
- ❑ Other government agency (please specify): ____________________
- ❑ Other, (please specify): ____

Of the 15 surveys responding, the breakdown was as follows:

8 Avionics or engine control developer
4 Aircraft or engine manufacturers
1 Software tool developer
1 Consultant
1 Other, (please specify): Software Verification Co.

A.1.2 BACKGROUND QUESTION #2.

What is your role relevant to RTSA? (Check all that apply)

- ❑ I perform RTSA on aviation system projects
- ❑ I develop real-time operating systems (RTOS) that support RTSA
- ❑ I use tools to perform RTSA
- ❑ I verify/test real-time scheduling performance
- ❑ I am an FAA engineer who approves compliance to DO-178B
- ❑ I am a Designated Engineering Representative (DER) who approves compliance to DO-178B

Other, (please specify):

Of the 16 surveys responding, with many individuals responsible for more than one role, the breakdown was as follows:.

6	I perform RTSA on aviation system projects
2	I use tools to perform RTSA
9	I verify/test real-time scheduling performance

5 I am a Designated Engineering Representative (DER) who approves compliance To DO-178B

A.2 REAL-TIME SYSTEM DEVELOPMENT.

A.2.1 REAL-TIME SYSTEM DEVELOPMENT QUESTION A.

What type of events are typically used to trigger time-critical functions of your real-time systems (e.g., interrupts, data message queries, data refresh rates, (pilot) user input, change of state, certain conditions, display refresh rates, etc.)?

Of the 15 surveys responding, their answers were as follows:

1. Combination of fixed schedule (i.e., clock time) plus certain conditions (for example, tracked signal weakness switches mode from tracking to acquisition)

2. Data inputs, data refresh rates, data queries, display refresh rates

3. In the OS all of the above are supported

4. Elapsed time, message update due

5. Interrupts

6. Interrupts, RTOS messages, RTOS semaphores or other similar mechanisms.

7. Fixed time interval scheduling interrupts only

8. RTSA is interrupt driven by time-to-go counters. We use an executive routine to establish priorities. Our Run Time Executive was written In-house.

9. Interrupts and change of state

10. Table driven scheduler based upon statically built table by an external, qualified tool. The table contains information regarding I/O timing (when to send data and when to read data), as well as process scheduling information

11. Time-based interrupts

12. Hardware interrupts, sensor/hardware inputs, airframe computer inputs, etc

13. Typically we use interrupts. In some instances it is an external input, such as incoming data, from another device that causes an interrupt to be generated.

14. Incoming data messages (interrupt or polled), timer expiration, user input (Key press, toggle switch, etc), sensor state change (interrupts or polled).

15. Interrupts are used for the initiation of the real-time processing.

Analysis of Question A Real-Time System Development
Interrupts were mentioned by 10 respondents as the main events that are typically used to trigger time-critical functions of their real-time systems.

A.2.2 REAL-TIME SYSTEM DEVELOPMENT QUESTION B.

What are typical performance requirements that your real-time system must meet?

Of the 15 surveys responding, their answers were as follows:

1. There are hardware and software deadlines. For the software, three independent bits need to be assembled and delivered every 0.6 microseconds.

2. Some requirements need to meet millisecond response times/tolerances; others have multiple second responses with millisecond tolerances.

3. The clock ticks at 1 kHz. All scheduling operations are driven by the ‘ticking clock’. User ‘frame’ times are typically 40-80 Hz

4. Servo loop closure without overshoot or significant lag. Service data bus and access information from the data bus according to schedule. Determine fault state based on successive out of range input for time frame.

5. Critical task must meet their real-time dead-line scheduled activities.

6. Sample rates in excess of 50 Hhz. For DSP applications. Multiple 8 KHz. Sample rate audio channels.

7. Level A assurance that all functions are completed before the next interrupt.

8. Max min limits on input/output events, gain phase margin on control loops, iteration rates and transport delay times on selected functions

9. Throughput margins of 50% were required by government contract. There were similar hardware reserve margin requirements on memory and I/O

10. There is a 1 KHz interrupt and several I/O devices interrupt

11. All aircraft I/O includes transmission and jitter requirements. Typical data rates range from 1 Hz to 80 Hz.

12. 80 Hz update rate

13. The hardware/computer shall operate without adverse affect on the engine or aircraft, such as lost of engine thrust, adverse increase or decrease of engine thrust or cause in-flight shut down.

14. As a military application our performance requirements are mainly based off of our 1553 bus rates. Thus we have certain message rates that our system must maintain for example rates of 50Hz or 25Hz etc. In addition – we may interface to an external device that requires data to be transferred to it within a few milliseconds after the data has changed.

15. Digital signal processing incoming data rate (radio IF): 200 ksam
Radio push-to-talk to RF on: < 50 ms; user input to display update: < 100 m
Sensor change to display update: < 100 ms

16. Performance requirements that are most critical include the time that the system detects an engine threshold exceedance (such as overspeed; under speed; excessive engine temperature; etc.) until the fuel flow is terminated to shut the engine down. This is usually on the order of a few milliseconds.

A.2.3 REAL-TIME SYSTEM DEVELOPMENT QUESTION C.

Where are your performance requirements for time-critical functions typically defined? (e.g., system requirements or interface control documents, software requirements document)

Of the 15 surveys responding, their answers were as follows:

1. System requirements and interface requirements

2. Time critical requirements may show up in our system requirements, software requirements or software design (low level requirements)

3. System requirements

4. Software Requirements Document

5. Software requirements Documents

6. System requirements, interface control documents, software requirements document

7. All the above

8. Ours were specified by government contract- typically some version of the TADSTAND

9. Through system design analysis or from similarity to previous systems, we can usually distinguish time-critical functions.

10. System requirements and interface control documents. On occasion, lower level performance requirements may appear in the software requirements document. Hardware documents also contain performance requirements as they pertain to the support of the I/O and O/S.

11. Systems requirements

12. Interface control document, computer hardware operation requirements and software requirement documents

13. Critical time with max latencies would be defined in our System and Sub-System Specification (i.e., our system requirements) document. 1553 message rates would be defined in our Interface Control Documents and some timing would be documented in our Software Requirements.

14. For inter-process communications timing. System requirements for input to output timings and user interface performance, software requirement

15. These are typically defined in the software requirements document- with some being defined in the system requirements document.

A.2.4 REAL-TIME SYSTEM DEVELOPMENT QUESTION D.

How do you distinguish time-critical functions from other functions in your application?

Of the 15 surveys responding, their answers were as follows:

1. All requirements must be met eventually. Some deadlines are obviously easy to meet; others will require some effort and/or have some risk associated with them. Perhaps it would be helpful to imagine all requirements having a deadline to be filled in, and if a recipient of an item (data, for example) really doesn't care by when something is accomplished, they should say so. NASA/JPL sometimes uses a time constraint network, where timing is stated with respect to other events, rather than with respect to time deadlines. For example, some aspects of a flight might not be required to be delivered until the flight arrives at the gate.

2. By the time element of the requirement (i.e., must respond within 200 mS, or must do x after y +/- z seconds.

3. Time critical information is embedded in the requirement

4. Error reporting has a unique identifier for each malfunction

5. By creating software or system requirements specifying the timing

6. Badly

7. I believe your question refers to arbitration done by the Executive. Levels of priority were established and the Executive routine carried them out. Priorities were established during the software design phase following analysis of the requirements.

8. Through system design analysis or from similarity to previous systems, we can usually distinguish time-critical functions.

9. From an O/S perspective, all functions are time critical. That is, they have specific timing constraints placed upon them that must be honored. The determination of who is "more time critical" than others is a system architecture exercise.

10. We don't

11. We don't

12. Time-critical functions would be documented as such

13. Listed in software design description, notation in function comment header

14. All functions in the application are time-driven and are treated as time-critical- with the exception of the low-level transmission and reception of serial data. This serial communication is interrupt-driven and will be serviced when data is available from the host computer to which the electronic control unit is communicating or when new data needs to be sent to the host computer. The processing and responding to complete messages in the system is also time-driven (a determination is made for whether a completed message has been received that needs to be acted on). Consequently, distinguishing from the time-critical functions and those that are not time-critical is done by dedicating separate memory resources to the serial.

A.2.5 REAL-TIME SYSTEM DEVELOPMENT QUESTION E.

Do your time-critical functions have dependencies on hardware or shared hardware devices (central processing unit, memory, data buses, I/O ports, queues, etc.) with other software functions of your application or other applications resident in the system? If yes, please explain.

Of the 15 surveys responding, their answers were as follows:

1. Multiple software entities can be incomplete at one time; knowledge about which ones are incomplete is used in deciding which entities get resource use assigned to them.

2. Sometimes, might have multi-processor system that must meet data bus timing protocol. Might be a hardware event that triggers the start of a time critical function.

3. The application has a dependency on the RTOS. The RTOS has a dependency on the system and applications, e.g. cache usage, memory configurations (coherent or non-coherent etc).

4. No

5. Yes. A system clock (counter/timer) is used for all software related timing functions. The RTS owns the system clock. A hardware watch dog is also used, which must be updated on a periodic basis

6. Yes, A/Ds and D/As for RF and audio sampling

7. No

8. It depends- hardware limitations that were known were part of the design. These dependencies were usually noted in the System and/or Software Requirements Specification. They were occasionally "discovered" during the Requirements Analysis phase- at great embarrassment.

9. Yes. Generally the interrupt servicing functions must share hardware devices and memory with other applications.

10. Typically the answer is yes, when looking at I/O hardware devices. However, our hardware designers use queues, state machines and other methods to abstract as much timing critical "stuff" away from the software, as possible.

11. No

12. Yes. There is memory loader software that is activated only when certain hardware and software conditions are set.

13. Yes, we must receive data from one subsystem and within a specified amount of time pass the information onto another subsystem – in some case using the input data to calculate the data that is to be output.

14. Time-critical functions typically share CPU, memory and other resources with other functions

15. Yes, the time-critical functions do have hardware dependencies on I/O devices.

A.2.6 REAL-TIME SYSTEM DEVELOPMENT QUESTION F.

What are some mechanisms that your application (software and hardware) uses to ensure that "time-critical triggers" get handled at the appropriate priority and gain the relevant resources to ensure that your performance requirements are achieved?

Of the 15 surveys responding, their answers were as follows:

1. A look-up table is provided (all at once) with input describing the tasks currently incomplete. The content of the look-up table tells which software task is to receive the processing resources next.

2. We don't use RTOS, so we write software to achieve desired responses (when possible).

3. Interrupt locking is used, OS/kernel locking, semaphores and priority inheritance, work queues for work deferrals, etc.

4. High Interrupt priority, also time critical code has hard vectors to interrupt service routines that cannot be modified by a run time operation. We are concerned about software maintenance activities adding new priority sets with higher priorities for less important processes. We document heavily the priority rationale.

5. Task priorities and interrupt priorities.

6. They are given the highest priority level within the RTOS. Or else they are handled outside of the RTOS with independent interrupts

7. By restricting to the one interrupt, and insisting on completion of time critical tasks.

8. Careful Integration testing and thorough V&V was our preferred path. Demonstration, analysis, whatever was appropriate.

9. Interrupt and task priorities are assigned appropriate values

10. All resources are defined statically at build time by a qualified tool. Processes are assigned specific resources. This ensures the resources will be ready when needed, and permits the O/S to detect any resource usage violations (i.e., partitioning violations).

11. Check on process completion and hardware watchdog

12. Our software uses watch dog monitor and certain CPU interrupt handlers.

13. As we use a PowerPC with only one external interrupt the "time critical" issues have to be designed into the system. There is NO way to prioritize interrupts in our system so the top level design must take this issue into account. Software must ensure all interrupt handlers are as quick as possible.

14. Hardware timers, prioritized hardware and software interrupts, pre-emptable and prioritized tasks.
    No (or very low iteration count) loops in round-robin systems.

15. There are no priority mechanisms in an interrupt occurs at a pre-defined interval and this initiates real-time processing. The software verifies that the processing from the previous interval has completed. If it has not, the first time the anomaly occurs, it is simply

recorded and processing for the next interval is initiated directly after the completion of the tasks from the previous interval. If two such overruns occur in a row, then the application simply executes an infinite loop, which will cause the engine to shut down. In addition, there is a hardware watchdog discrete output that needs to get strobed each interval. If the watchdog is NOT strobed during each interval, then the hardware will cause the engine to shut down our software.

A.2.7 REAL-TIME SYSTEM DEVELOPMENT QUESTION G.

What type of reviews, analyses and testing does your team use to ensure that time-critical functions will satisfy their performance requirements, especially in worst-case condition scenarios?

Of the 15 surveys responding, their answers were as follows:

1. We know the instructions and the instruction execution times.

2. Worst case timing analysis, overloaded resource tests

3. Need a two day essay to answer this.
   In the past we analyzed the paths of the RTOS and determined what the relationship to time was with inputs. Things have got a lot trickier since then. We are currently doing this as a study for a partitioned RTOS that we are certifying.

4. Reviews are requirements based reviews. Analysis includes calculation of worst case safety margin for timing, for memory usage, and for stack usage. Verification tests use emulators that track the percentage of time spent different areas of the software, so we can get a good estimate of how much timing margin remains in the testing stage.

5. Unit test are run on all related software modules. Integration tests are run to verify functionality in the system. Built-in-tests check functionality each time the unit is powered up.

6. Requirements, Design and code reviews are conducted. A worst-case throughput analysis is conducted.

7. Two aspects – one at the requirement level, one at the implementation level.

8. Requirements are peer reviewed for ‘within our product’ issues and reviewed with data source / sink suppliers for external stuff. Code modules are all path tested and execution time monitored. Each module is assigned its longest run time. The total of these times for all modules that execute between timer interrupts are summed and must be less than the minimum interrupt time.

9. For interrupts, endurance testing with appropriate monitors allows us to determine the effective worst-case timing as well as the average conditions.

10. Automated test cases are developed that exercise the limits of the system. A "rogue partition" is used to stress the partitioning aspects of the O/S.

11. Timing measurements

12. All of them. Software requirements/design review, test readiness review, code/design review, unit tests, integration tests, software/hardware system tests

13. The systems team uses MatrixX to output equations. These equations are then modeled and simulated using MatLab which also does bode plots. The software team utilizes a tool (WindView) to verify that the tasks are being scheduled as required and pre-empted as required.

14. End to end testing of LRU inputs and outputs. Test cases designed to assure full loading on data inputs

15. The worst-case interval time is measured and recorded. The fact that the system is so time-deterministic combined with the above handling of overrun conditions assures that while we are executing the engine control software, we will be meeting our time-critical functions.

A.3 MESSAGE PASSING.

A.3.1 MESSAGE PASSING QUESTION A.

What approaches to message passing have your projects utilized?

Of the 15 surveys responding, their answers were as follows:

1. Write to hardware buffers that are swapped on a fixed time.

2. Typically we've used dual port ram for multiply processor systems, we don't use mult-threaded executives.

3. Message passing library is provided in the RTOS. Priority and FIFO based.
   Semaphores of various types are also used. (Binary, Counting and MUTEX)

4. We pass messages in one of two ways. The preferred method is to place the data packet in general purpose registers, and call the service routine with the knowledge that the service routine will look for specific data in specific registers. An alternate way is to pass a pointer to the data specific to the message to the service routine.

5. Events flags; rendezvous; data passed through shared memory, and through shared buffers owned by the RTS.

6. RTOS based messages

7. Don’t do it

8. Or was Unfamiliar with the term- don’*t* understand the question. Our communications software was either internally developed code to do data handling within the processor system unit formatted to one of the popular protocols such as ARINC 429, RS 232, RS422 MIL STD 1558, etc.

9. Shared memory accesses using semaphores, double buffering, and interrupt blocking have all been used

10. Periodic inter-process messages and mailboxes

11. None

12. ARINC

13. Information passed internal on the same processor is often passed via message queues. Information passed to other processors is passed in dual-port memory.

14. Shared memory and semaphores, mailboxes, queues

15. And not via message passing mechanisms. The major components in the system communicate largely via memory interfaces

A.3.2 MESSAGE PASSING QUESTION B.

Do your messages communicate with each other? If yes, please explain how:

Of the 15 surveys responding, their answers were as follows:

1. Processing entities communicate with each other according to the following paradigm “Fast but dumb vs. smart but slow.” If the messages are simple, the sender communicates directly with the recipient. If the message is not simple, it goes to a processing unit with more processing capability, which can figure out what’s to be done, and inform all interested other processing

2. No

3. Tasks communicate with interrupt routines and other tasks. Messages are just the carriers.

4. No

5. No

6. No. They only pass information to other tasks.

7. Still not clear how to answer- we have serial digital communications and traditional analog /discrete I/O and follow a popular industry standard. Handshaking, parity etc. are used as required.

8. No

9. No

10. ARINC 615 protocol

11. No, our messages internal to our system tend to go one direction

12. Tasks and processes communicate with each other using messages, messages don'*t* communicate with Each other.

13. N/A

A.4 PROCESSOR TYPES.

A.4.1 PROCESSOR TYPES QUESTION A.

What types of processors have you used for your systems?

Of the 15 surveys responding, their answers were as follows:

1. Custom CPUs for the fast but dumb, and commercial CPUs for smart but slow.

2. 8051, 68332, TMS34020, and we are eyeing power PC style processors for future products

3. PowerPC. / Intel X86

4. 8051 eight bit microcontrollers

5. Intel 486DX4, 486DX2, 486Dx, 486SX, 80186, 6802, 8085
   DSPs, TMS320C6711, Intel x86 class processors 80186, 80386ex, & 80486DX4100

6. Currently, home grown

7. TI9900 (I'm a really OLD guy), Intel 80C186, 80C386EX, 68HC11, Motorola 380020 and 380040(memory is hazy- probably wrong), *TI* 320C30 DSP (as I said, I'm an old guy)

8. Intel 80186, Motorola 68HC11, Motorola 68HC16, Motorola 683XX and PowerPC403

9. Intel x86 family (386, 486, Pentium I, II, III)

10. Intel 80960, Motorola 68332

11. 68000, 68020 Power PC

12. Currently we are using the PowerPC for our controlling processor. Previously we used the Intel i960MC.

13. 8051 (8-bit), XAG49 (16-bit)
80186 (16-bit), 80386 (32-bit), *TI* and Motorola DSP (16, 24, and 32-bit)

14. PowerPC 603e, PowerPC 555, i960, Intel x86

A.4.2 PROCESSOR TYPES QUESTION B.

Have you found any peculiarities with any of the processors that affect the real-time scheduling analysis? If yes, please explain the peculiarities and how they were addressed:

Of the 15 surveys responding, their answers were as follows:

1. Yes, don't details aren't readily available

2. Memory management units and cache memory make a difference

3. We have found that the uncertainty in the number of instruction cycles to vector to an interrupt service routine has led us to utilize external hardware clocks for timing in certain cases

4. Great care must be used when using non-maskable interrupts

5. No

6. Slightly off topic – a long time ago a Texas Instruments processor which claimed an asynchronous reset capability was shown to have a synchronous window when it did not reset. We changed processors.

7. We were doing the work during a time when simulation tools didn'*t* exist, and when they did- it was usually too rich for our pocketbook. The general approach was to "let it fly" for many times the expected mission time and fully exercise all modes of the application code and pray we had timing margins by looking at what was felt to be a "worst case" data moving scenario. I'm certain those tools have developed and are more affordable today.

8. Different processors have different mechanisms for assigning interrupt priorities and for interrupt masking

9. Not that I am aware of

10. Caching makes timing measurements using bus analyzers a nightmare

11. Don't know

12. The limitation of the PowerPC (not a peculiarity), as I see it, is that it only allows for one external interrupt into the processor. As we rely on multiple interrupts, our system hardware has to design external interrupt controls to handle the multiple interrupts – but it is impossible to prioritize the interrupts since the PowerPC accepts only one external interrupt. The Intel i960MC handled multiple interrupts that could easily be prioritized. The interrupt limitation is addressed at design attempting to ensure that the interrupt will be spaced out and then the software works to ensure the handlers are as quick as possible.

13. Instruction pipelines. Avoid software timing loops

14. No

A.4.3 PROCESSOR TYPES QUESTION C.

Do your systems use a single processor or multiple processors? If multiple processors, how is the system functionality distributed and handled across processors?

Of the 15 surveys responding, their answers were as follows:

1. Multiple. Minimize the amount of communication that must flow between processors. Try to have a predominant direction for communication: A with respect to B is mainly A sends to B.

2. Both single and multiple processor products.
   We establish functionality responsibilities up front and don't adjust in real time

3. Our current work uses a single processor. Many years ago a multiprocessor system was used with shared memory.

4. Single

5. Multiple. Functionality is allocated at the System document level. Inter-processor communication is by dual port shared memory or through a RS232 serial link or a synchronous link.

6. Typically single processor. However, one was a multi-processor system that communicated via Dual Port RAM. An ICD was created to describe this communication.

7. Redundant single

8. Our applications were generally single processor. We did do some multiple processor work, but it was usually with some form of shared dual port registers or RAM. We felt interaction needed to be minimized to mitigate control problems where one unit might "jam" another one. Unambiguous synchronization was occasionally a problem.

9. Sometimes multiple processors are used. Usually the functionality is distributed according to high level function and re-uses considerations.

10. Multiple processors, however, each is treated as an independent processor. That is, they work as federated LRUs interconnected by an avionics bus.

11. Single

12. It is a dual channel operation with each channel having the same software. Each channel is executing during operation with one channel being in control and active while the other channel is in the stand-by mode.

13. Our system contains multiple processors but each processor handles a separate function. One processor is the controller, another processor handles displays, another processor communicates to recording devices, etc. Information between the processors is communicated through dual-port shared memory.

14. Multiple. Display/keyboard, communications I/O, sensor I/O, high level radio functions, low level radio functions, DSP Serial data link is typically used between processors.

15. The system is a single processor architecture

A.5 SCHEDULING.

A.5.1 SCHEDULING QUESTION A.

What scheduling algorithms/approaches have you used to schedule your system tasks at run time? Please match the algorithm (e.g., preemptive priority, round robin, etc.) with the system type (e.g., display, communication, navigation, etc.)

Scheduling algorithm/approach	System type

Of the 15 surveys responding, their answers were as follows:

1.

Combination of schedule established in advance with varying situation-dependent priority with no pre-emption, (but also with critical sections).	communication

2. Do not use real time schedulers

3.

Round robin with time-outs	Overheat detection systems
Periodic tasks, with messaging and semaphores	Temperature monitoring system
Interrupt driven	FADEC
Priority	Control systems

4.

Preemptive priority	Display
Preemptive priority	Navigation - DME

5.

Preemptive priority	Map Display
Preemptive priority	Communication
Cooperative multi-tasking	Communication/Display
Simple sequencer	Communications/DSP

6.

Strict list scheduled	Engine control

7.

Preemptive priority	Navigation

8.

Preemptive priority	Autonomous central control of a UAV

9.

Preemptive priority	Navigation

10.

Rate Monotonic Analysis (RMA)	All

11. Our systems utilize very simple interrupt-based schedulers for efficiency and safety reasons. There are no priorities – process run when invoked; completion is checked using Boolean flags.

12.

Foreground/background with minor frame watch dog monitor	Engine control

13.

Preemptive Priority Scheduling	System Controller (Communication and Navigation)

14.

Preemptive	Communications (radio)
Single Main Loop With Interrupts	User-interface, navigation

A.5.2 SCHEDULING QUESTION B.

If you used priority scheduling, how many priorities levels were assigned? How was priority inversion avoided? How did number of priority levels compare to number of processes?

Of the 15 surveys responding, their answers were as follows:

1. Each process had its own priority level, that is, we really used process identification directly, rather than a classification according to priority. Furthermore, depending upon the situation at hand, different processes were chosen for resource allocation. That is, in one situation where tasks A, B and C were all ready to run, process B would be chosen, but in another situation, process C would be chosen. There was a look-up table which was provided with an "address" constructed of the Booleans for each runnable task, true for each of the tasks that were ready to run, and other situation bits of interest. The content of the look-up table was the identification of the task that was to be run in that situation. Each task had at most a very small critical section. On completion of each task, the situation is detected and supplied as input to the look-up table. Each task is very short.

2. N/A

3. Temperature system was event driven. Each task has a unique priority. 5 task system

4. Three priority levels, priority inversion was eliminated with using priority registers initialized and subsequently left alone. There are six processes.

5. On one particular project: 12 levels. Priority inversion was avoided by careful software design. Some tasks share the same priority number; (round robin effect); Total tasks: 15

6. 5-10 levels and we never encountered priority inversion

7. I recall there were at least three levels, possibly as many as five- our applications were tiny- less than 100K lines of HOL code.

8. 10-20 priority levels are typical. Priority inversion is avoided by limiting the amount of resource sharing between widely disparate priority tasks.

9. N/A

10. N/A

11. Ten priority levels are assigned. Each rate group is assigned its own priority level. Message queues are not deleted and non-blocking queues are defined.

12. Four tasks at different priority levels. Design high priority tasks do not wait on low priority tasks

13. N/A

A.5.3 SCHEDULING QUESTION C.

What kind of scheduling problems have you encountered in multitasking systems and how were they addressed?

Of the 15 surveys responding, their answers were as follows:

1. One (more instructions needed in a time interval) was solved by increasing the clock speed on the processor. One (couldn't figure out which task to run next examining successive status bits, because status bits changed so quickly) was solved by latching status into an address register for a look-up table. One (priority inversion) was addressed by aggressively pruning the content of critical sections.

2. N/A

3. Working with duration time for periodic delays caused us to change the implementation to work with Absolute time for delays (delay until) Problem solved in OS by incrementing time in a non-preemptable block.

4. N/A

5. Not enough bandwidth for the processor (this is a common problem). Rates of tasks are adjusted as well as shifting task work loads.

6. None

7. Crashes were painfully examined with support from debuggers and a logic analyzer. Most troubles were traced to improper manipulation of the memory register stack rather than some undocumented hardware feature.

8. I’ve encountered “deadly embrace” problems which were addressed by a re-design of the affected tasks.

9. Don’t know

10. Minor frames overrun, which causes a watch dog interrupt thus cause a software reset

11. We have seen task overruns. When this occurs sometimes a code problem is found and fixed. If indeed more is scheduled then can be completed in the allocated time – all functions in the task are evaluated to determine which functions can be done at a slower rate or divided up slightly different.

12. No comment

13. N/A

A.5.4 SCHEDULING QUESTION D.

Have you used real-time operating systems to support your schedule guarantees? If yes, what kind of operating systems have you used and what kind of scheduling challenges have you encountered?

Real-time operating system type	Scheduling challenges

Of the 15 surveys responding, their answers were as follows:

1. NO

2. Home grown only

3.

Preemptive priority	More robust RTS means slower RTS

4. Not really, we typically use C language asserts to detect their occurrence and then design them out.

5. NA

6.

I left the field before using commercial RTOS

7.

Preemptive multi-tasking	Interrupt latency, excessive overhead

8. In-house proprietary, using RMA.

9.

Operating system using Ada or assembly	Keeping from cause a minor frame overrun

10.

VxWorks 5.4	Main challenge is to interface to our custom hardware

A.5.5 SCHEDULING QUESTION E.

Do you verify what data gets dumped, due to priority settings and functions getting preempted? If yes, how does it affect your system?

Of the 15 surveys responding, their answers were as follows:

1. N/A

2. We have verified context switching during our work on certification of the VxWorks/Cert OS

3. Preempted functions because of priority settings are only delayed. In our designs, pre-emption generally happens when we are trying to simultaneously transmit on an ARINC 429 bus while we are receiving data. This means functions that come up while the transmitter is on will have to wait until it is off to complete. We guarantee that this will happen in less than 200 mSec.

4. Yes. If the problem is severe enough then the system is declared invalid and functionality is removed

5. No

6. N/A

7. Not sure I understand where the question is going.

8. Yes. The impact varies. In some cases the loss of data is acceptable and in other cases it is absolutely critical that no data is lost.

9. Don’t know

10. N/A

11. No

12. Buffer sizes and interrupt/task priorities are selected to assure that no data are lost

13. N/A

A.5.6 SCHEDULING QUESTION F.

Do you use tools to assist in the real-time scheduling analysis? If yes, what kind of tools? How are the outputs of these tools verified?

Scheduling Tool Type	Approach to Verifying Tool Output

Of the 15 surveys responding, their answers were as follows:

1.

	review of source code, test of executable on platform

2. Emulators and simulators

3. None yet

4.

Emulator	We can insert test patterns on port pins that model one or two of the module entry points, then monitor them with an oscilloscope to validate the emulator calculations.

5.

Real-time timing and analysis tools assisted by debugger tools	Timing measurements and analysis of data.

6.

MS Excel	We just use MS Excel for analysis. We then attempt to create a design that accommodates the worst case. This has worked well so far.

7. Not sure I understand where the question is going.

8. No

9.

In-house proprietary.	Qualified tools, via DO-178B

10.

Scope and ARINC bus analyzer	Monitor certain variables in software

11.

WindView	Off the shelf tool - it is not verified in house

12. NO

13. No tools were utilized in RTSA

A.5.7 SCHEDULING QUESTION G.

What trends in commercial aviation systems do you think will challenge the current scheduling approaches (i.e., may lead to the need for new scheduling algorithms)?

Of the 15 surveys responding, their answers were as follows:

1. Desire to reuse, desire to inherit confidence from re-use, desire to use non-developmental items

2. Don't know

3. Partition based OS's, Event based scheduling

4. A virtual explosion in the use of very high speed data buses will be very challenging.

5. Multiple thread real-time dead line scheduling analysis

6. High speed data busses

7. The regulatory environment

8. More display and I/O requirements

9. Don't know.

10. Partitioning (ARINC-653)

11. Don't know

12. The move to 178B will challenge our existing approach. Thus we are looking into off the shelf RTOSs that handle these challenges. It seems that data protection and proof of code coverage will be bigger challenges then the scheduling algorithms.

13. Fly-by-wire, automated landing,

14. Collision avoidance

A.6 TIMING.

A.6.1 TIMING QUESTION A.

After system development, do you verify that deadlines are met and scheduling analysis assumptions are correct? If yes, please explain how.

Of the 15 surveys responding, their answers were as follows:

1. Yes. Test all the paths through the logic, which we did, using test inputs and a logic analyzer. Also, in a communication system, the achievable bit error rate vs. signal-to-noise ratio is calculable. We could control the signal and the noise, and we used end-to-end bit error rate testing over very long periods of time, and could observe performance to be that predicted by theory.

2. Yes, by test and analysis

3. Not yet, we leave this to our customers

4. Yes, we verify at the board level using functional tests, we incorporate flight tests with parameter evaluation, and we measured during software verification and validation.

5. Yes. Real-time checks are continually made in the software. Status codes are stored in NVRAM for after the fact viewing.

6. All deadlines specified as system or software requirements are verified. Less formal analysis is not always verified.

7. As mentioned above, I'm an old guy and we were rather primitive in the 1970s an*d*1980s.

8. Yes. Software monitors are inserted to verify scheduling performance

9. Yes, via a combination of system/function testing (based upon system requirements) and also standard software testing.

10. Yes, timing analyzers

11. Yes. Per our software requirements, we review the timing analysis data.

12. Yes, several methods are used for verification. Running special system scripts an oscilloscope is used to verify timing of certain events. 1553 Bus captures are done to ensure specific data is sent out at the required time. Inputs are stimulated and latency to the associated output is measured to verify it is within tolerance. In addition, inspections are done to analyze that the code is performing certain operations at a specified time – if the previous methods cannot be used to verify this.

13. Yes. Apply data at maximum rate to LRU inputs, check for timely and appropriate response at LRU outputs. Error messages emitted on buffer overflow

14. Yes. The critical timing functions are verified by externally recording the time from the input initiation to the required action. This recording is done external to the electronic control unit by inputting stimuli to the unit and measuring the action from the system.

A.6.2 TIMING QUESTION B.

In what areas of timing verification or validation have you encountered problems and how were they addressed?

Of the 15 surveys responding, their answers were as follows:

1. Timing verification for us was much easier using test equipment than by using software self-monitoring tools. Writing to a logic-analyzer accessible output register at known points in the software can be helpful.

2. Biggest problems have been in area of system response times following application of power. Most often means of addressing problem has been to shed tasks needed to be performed before system responses are supplied (e.g., BIT).

3. Other area is in time base data processing systems where uneven work load has resulted in some main loop overruns, solution has been to reallocate tasks to even processing out.

4. Timing is hard in a tasking preemptive OS in the presence of caching

5. Problems have occurred in measurement uncertainties in the emulator. We discovered that we can not rely on snap shot measurements; we need to allow a long elapsed time measurement to get the most accurate picture.

6. If an abnormality occurs, there is never enough data to help in analysis. If the problem is very intermittent then special modified software may be needed to analyze the problem.

7. People not doing what they should

8. I think you’re asking a question that may be a no-brainer because we tended to be rather conservative in our designs. It just wasn’t part of our culture to press the envelope since we lacked adequate simulation tools that should be part of a designer’s kit in handling timing issues.

9. Throughput problems have lead to a re-balancing of tasks and priorities

10. Timing problems typically appear because of shortcomings in the original requirements.

11. Instrumentation requires access to the address and data buses; this is not possible in production (sealed box) hardware.

12. Cannot test every possible failure mode or condition. It gets even harder to test multiple failure modes/conditions.

13. Verifying the rate of data exchange between processors has been challenging. Special “debug” code is left in a build to be used for verification

14. Unknown

15. None

A.7 FAULT-TOLERANT.

Does your testing allow for faults? If yes, please explain.

Of the 15 surveys responding, their answers were as follows:

1. Situation bits in the “what to do next” look-up table included error conditions having been signaled. Error detection and error handlers appear in the system at several levels. We controlled the communication input, as well as the signal-to-noise ratio, and we could inject faults in many places to test responses to faults.

2. Don’t understand the question

3. N/A

4. Our testing incorporates fault injection at times to evaluate failure mechanism monitors. For example, we turn off the watchdog toggle and evaluate the impact it has on timing. We also write patches that allow delay loops to extend to infinity while doing nothing. This helps us evaluate the effectiveness of our monitors and our protection schemes.

5. Yes. A persistence count of faults is maintained and monitored continuously

6. C language asserts are set up in the code to catch timing faults that occur during run-time. These cause a pre-defined software fault that can be detected and traced external to the system.

7. External to the processor system – yes. See above ‘all paths’ comments

8. Our faults were externally inserted through the I/O, but generally we ignored this area by creating an environment where program execution was assumed by a Watchdog Timer that was updated at some regular interval from the Executive. Being a UAV had its advantages.

9. Yes. In some cases, things such as task slippage are completely acceptable

10. Fault injection testing is included as a part of robustness testing

11. Yes. When we try to accommodate it by using other sensor data if condition is adverse we switch to the stand-by channel

12. No

13. Unknown

14. No

<u>A.8 OPINION ABOUT RTSA AND ITS VERIFICATION</u>.

In your opinion, what are the major issues regarding RTSA and its verification?

Of the 15 surveys responding, their answers were as follows:

1. The greatest difficulty I have experienced is from sources of requirements who believe what they want can be had sufficiently instantaneously that they do not examine their needs, and are therefore not able to establish what their requirements are. The second largest source of difficulty is from software teams that do not want to have to meet real time requirements, and tend to claim that whatever the task, their code and their favorite operating system is bound to be good enough. This gets reflected to the team manager who cannot force the programmers to use the appropriate tools.

2. Confirmation of timing issues under all foreseeable circumstances

3. The OS, the hardware, the device drivers, are trying to abstract the performance of the underlying application so that it appears to run on a virtual processing system. This virtual processing system is trying to optimize performance by ‘globalizing’ information, e.g. cache (instruction and data), pipelining, speculative instruction scheduling, shared resources (e.g. memory). At the same time the OS is trying its best to keep throughput high using buffering, interrupt driven drivers, asynchronous peripherals, bus snooping etc. This makes time determinism very hard.

4. The major issue is not the first implementation of the real time system. It is the modifications and changes that occur later as the system evolves, picks up more features and functions. Regression testing is supposed to discover all those added anomalies, but it doesn’t always.

5. The tools are very expensive and not always available; ore is training for the tools available

6. Poor early analysis and design. Poor institutionalization of worst case throughput analysis

7. Without a doubt- the FAA is the ONLY issue.

8. It tends to be somewhat intuitive and has the potential for being sub-optimal

9. Operating systems are typically driven by derived or implementation requirements (i.e., low level). Any system requirements (i.e., high level) levied against an OS are so general and broad, that they are of little value. They really can't be tested (other than to say, yup,

it does that). Consequently, the resulting documentation for an OS consists of very detailed software requirements -- without any real system requirements. DO-178B and the FAA/JAA seem to have a difficult time with this reality. I have attended SOI audits were much discussion was spent on the requirements, so they were so detailed and didn't trace up to high level requirements. This is a fact of life, when dealing with platform/foundation/utility software.

10. Don’t know

11. I think the issues for the future mainly have to do with the verification not so much the actual scheduling. Better non-intrusive tools with more visibility into the scheduling aspects. Possibly the processors themselves need build-in hooks/functionality that can be tapped into.

12. No comment

A.9 ADDITIONAL INFORMATION.

What other real-time scheduling experience or issues would you like to share?

Of the 15 surveys responding, their answers were as follows:

1. I chaired a session at an FAA Data conference (sponsored by ASD/Carol Uri/Felix Rausch) and tried to discuss requirements. A vocabulary for communication of requirements between system users and system developers seems to be needed.

2. None

3. Need more time to think about this?

4. None

5. None

6. None

7. None

II. Would you be interested in participating in FAA-sponsored efforts to address real-time scheduling analysis (e.g., participate in an interview, participate in development of policy or guidance, etc.)?

Of the 15 surveys responding, their answers were as follows:

6 yes
8 no
1 Maybe

III: If you said “yes” to II, what is your area of interest?

Of the surveys responding, yes their answers were as follows:

1. ATC systems

2. We certified Ada RTS’s, VxWorks/Cert, a BSP and continue certification work with a Partitioned Integrated Modular Architecture. We are particularly interested in solving time analysis problem. Right now we push the problem back to the user of the RTOS, but I expect we will be asked many more questions about the contribution that the OS makes to the timing of the application.

3. We have thought about this a lot, and while we can produce a lot of data we don’t know what format this should be to be useful to the user.

4. Real time control systems and networking issues

5. Software development.
6. Verification and certification

7. Obviously- Drive-By-Wire technology transfer from the automotive industry

8. I think Cots and partitioned operating systems are getting more & more attention – I’d like to learn more about these

IV. If you said “yes” to II, how can we contact you?

No additional information was provided by the responders.

APPENDIX B—THE RESEARCH PROJECT DETAILS WITH ASSUMPTIONS

In this appendix, the known results are described in sections B.1 and B.2. The new results obtained in this research effort are described in sections B.3 through B.6.

B.1 DYNAMIC-PRIORITY SCHEDULING DISCIPLINE.

As mentioned in section 3, the Earliest-Deadline-First (EDF) algorithm is optimal on one processor with respect to the Dynamic-Priority scheduling discipline [B-1 and B-2]. This means that any real-time task system (TS) schedulable on one processor by any Dynamic-Priority scheduling algorithm is also schedulable by EDF. It also means that a real-time TS is feasible on one processor with respect to Dynamic-Priority scheduling discipline if, and only if, it is schedulable by EDF. Since EDF is optimal for one processor, there is no compelling reason to consider other Dynamic-Priority scheduling algorithms. Therefore, this effort is restricted to EDF only.

EDF can be implemented by maintaining a queue of active tasks (i.e., tasks that have made a request but have not yet finished execution) arranged in ascending order of the deadlines of the requests of the tasks. Whenever the processor becomes free for assignment, the task at the head of the queue will be assigned to the processor. When a new request arrives, its deadline will be compared with the deadline of the task that is currently executing, and if the deadline of the newly arrived request is closer to the current time, it will receive the processor. The task that was executing before will be pre-empted and put back in the queue. EDF is usually implemented by software because of the operations involved. This makes it not as appealing as Fixed-Priority scheduling algorithms, since the context switching time is higher than those of Fixed-Priority scheduling algorithms. On the other hand, EDF yields a higher processor utilization than Fixed-Priority scheduling algorithms.

Table B-1 shows a real-time TS whose EDF schedule is shown in figure B-1.

TABLE B-1. A REAL-TIME TASK SYSTEM (EXAMPLE 1)

T_i	s_i	e_i	d_i	p_i
T_1	0	2	6	15
T_2	1	1	3	5
T_3	0	1	2	3

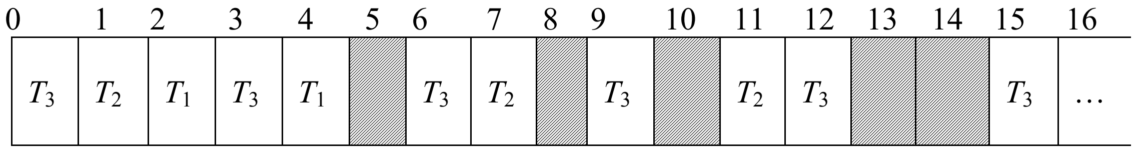

FIGURE B-1. EARLIEST-DEADLINE-FIRST SCHEDULE OF THE TASK SYSTEM IN TABLE B-1

The question of determining if a real-time task system (TS) is schedulable on one processor by EDF is considered. This is tantamount to determining if the schedule produced by EDF is valid. As the next theorem [B-3] shows, some simple cases of this question can be determined efficiently.

Theorem 1: Let $TS = (\{ T_i \}, \{ s_i \} , \{ e_i \} , \{ d_i \} , \{ p_i \})$ be a real-time *TS* consisting of *n* periodic, real-time tasks. Then (1) $\sum_{i=1}^{n} \frac{e_i}{d_i} \leq 1$ is a sufficient condition for *TS* to be schedulable by EDF and (2) $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ is a necessary condition for *TS* to be schedulable by EDF. In the special case where $d_i = p_i$ for each $1 \leq i \leq n$, $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ is both a necessary and sufficient condition for *TS* to be schedulable by EDF.

Theorem 1 gives a simple test for schedulability in the special case where $d_i = p_i$ for each $1 \leq i \leq n$. The *TS* is schedulable if, and only if, $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$.

There is no simple test for schedulability when d_i is not equal to p_i for some $1 \leq i \leq n$. If $\sum_{i=1}^{n} \frac{e_i}{p_i} > 1$, one can safely say that it is not schedulable. On the other hand, if $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$, one cannot conclude that it is schedulable. Similarly, if $\sum_{i=1}^{n} \frac{e_i}{d_i} \leq 1$, one can safely say that it is schedulable. On the other hand, if $\sum_{i=1}^{n} \frac{e_i}{d_i} > 1$, one cannot conclude that it is not schedulable.

While there is no simple test for the general case, one can still test schedulability by checking if the deadline of each request of each task in the EDF schedule is met. For this method to work, one needs to establish an a priori time-bound for which one needs to construct the EDF schedule. If the initial request times of all tasks are identical, an obvious time-bound would be *P* = least common multiple of $\{p_1, ..., p_n\}$. This is because all tasks simultaneously make a request at their initial request time, and then again simultaneously make a request at *P* time units later. The EDF schedule will be cyclic with a cycle length equal to *P*. Thus, if the EDF schedule is valid for a period of *P* time units, it will be valid for any length of time.

On the other hand, if the initial request times of the tasks are not identical, it is not clear that such a time-bound necessarily exists. Leung and Merrill [B-3] showed that such a time-bound indeed exists and is given by $s + 2P$, where $s = \max\{s_1, ..., s_n\}$. In the following discussions, $\min\{s_1, ..., s_n\} = 0$ is assumed.

Let *S* be the EDF schedule of the real-time *TS*. Define the configuration of *S* at time *t*, denoted by $C_S(t)$, to be the *n*-tuple $(e_{1,t}, ..., e_{n,t})$, where $e_{i,t}$ is the amount of time for which T_i has executed since its last request up until time *t*, and $e_{i,t}$ is undefined if $t < s_i$. Leung and Merrill [B-3] proved the following theorem.

Theorem 2: Let *S* be the EDF schedule of a real-time *TS*. *TS* is schedulable by EDF on one processor if, and only if, (1) all deadlines in the interval $[0, t_2]$ are met in *S*, where $t_2 = s + 2P$, and (2) $C_S(t_1) = C_S(t_2)$, where $t_1 = t_2 - P$.

An algorithm to determine if a real-time *TS* is schedulable by EDF on one processor consists of constructing an EDF schedule *S* and checking if all deadlines in the interval $[0, t_2]$ are met in *S* and $C_S(t_1) = C_S(t_2)$. By Theorem 2, *TS* is schedulable by EDF if, and only if, both conditions are satisfied.

Note that the running time of the above algorithm is an exponential function of the input parameters, $p_1, ..., p_n$. One wonders whether there are more efficient algorithms, e.g., algorithms whose running time is a polynomial function of the input parameters. Unfortunately, Leung and Merrill [B-3] showed that it is unlikely that such an algorithm could be found, as the next theorem shows.

Theorem 3: The problem of deciding if a real-time *TS* is schedulable by EDF on one processor is nondeterministic polynomial (NP)-complete.

NP-complete problems are a class of notorious computational problems. This class of problems has the property that if any problem in the class has a polynomial-time algorithm, then every problem in the class would have a polynomial-time algorithm. At the present time, none of the NP-complete problems can be solved in polynomial time. Since this class contains many notorious problems (such as traveling salesman problem, Hamiltonian path problem, etc.), which had been studied for more than a century, it is widely conjectured that none of the NP-complete problems can be solved in polynomial time. The reader is referred to the excellent book by Garey and Johnson [B-4] for a discussion of the concept and implications of NP-completeness and NP-hardness (which will be discussed later in this theorem, as well as in theorems 4, 7, 10, and 11).

Leung [B-5] showed that EDF is not optimal for $m > 1$ processors. At the present time, no simple algorithm is known to be optimal for more than one processor. Lawler and Martel [B-6] used the idea of network flow to construct a valid schedule on *m* processors if the *TS* is indeed feasible on *m* processors. At run time, the scheduler must schedule tasks according to the scheduling table. This is essentially the Clock-Driven approach, which, as discussed in section 3, is not desirable.

There are two general approaches in scheduling on $m > 1$ processors: the global approach and the partition approach. In the global approach, the *m* processors are treated as a pool, and an active task is assigned to an available processor if there is one. Otherwise, it will be put into a waiting queue until a processor becomes available. The waiting queue is ordered by the priorities of the tasks. By contrast, the partition approach partitions the set of tasks into *m* groups, with each group of tasks assigned to a processor. Tasks assigned to a processor can only be executed by that processor. The partition approach is generally preferred over the global approach because of the ease in processor management and the availability of optimal algorithms for one processor. The following discussions are restricted to the partition approach only.

The main goal in multiprocessor scheduling is to partition a set of real-time tasks into the smallest number of groups such that each group is schedulable by EDF on one processor. Unfortunately, this problem is also NP-hard, as shown by Leung and Whitehead [B-7]. (Note: An NP-hard problem is at least as hard as an NP-complete problem, and possibly harder. At the present time, there are no known polynomial-time algorithms to solve either an NP-complete or an NP-hard problem. It is widely conjectured that none of these can be solved in polynomial time.)

Theorem 4: The problem of partitioning a set of real-time tasks into the smallest number of groups such that each group is schedulable by EDF on one processor is NP-hard.

Theorem 4 suggests that it is extremely unlikely to solve this problem in polynomial time. Thus, there is a need to develop fast heuristics that will yield near-optimal solutions. In the literature, there seems to be a pronounced absence of fast heuristics for the general case. This is probably due to the fact that it is time-consuming to check if a set of tasks is schedulable on one processor by EDF. However, for the special case where each task has its relative deadline identical to its period, there is a simple test for schedulability; a set of *n* tasks is schedulable on one processor by EDF if, and only if, $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$. One would expect that fast heuristics exist for this special case.

As it turns out, this special case can be modeled as a bin packing problem. In the bin packing problem, an infinite collection of unit-capacity bins and a list of pieces with sizes between 0 and 1 are given. The goal is to pack the pieces into a minimum number of bins so that no bin contains pieces with sizes totaling more than 1. The bin packing problem is known to be NP-hard, but there are numerous effective heuristics for it; see Coffman, Garey, and Johnson [B-8] for a survey. One can model the task partition problem as a bin packing problem as follows. Each processor is viewed as a unit-capacity bin. The tasks are treated as pieces with sizes given by $\frac{e_i}{p_i}$. The problem of partitioning the tasks into a minimum number of groups so that each group is schedulable by EDF is equivalent to packing the pieces into a minimum number of bins so that no bin contains pieces with sizes totaling more than 1.

B.2 FIXED-PRIORITY SCHEDULING DISCIPLINE.

As mentioned in section 3, the Deadline Monotonic (DM) algorithm is optimal on one processor with respect to the Fixed-Priority scheduling discipline [B-7]. This means that any real-time *TS* schedulable on one processor by any Fixed-Priority scheduling algorithm is also schedulable by DM. It also means that a real-time *TS* is feasible on one processor with respect to Fixed-Priority scheduling discipline if, and only if, it is schedulable by DM. Since DM is optimal on one processor, there is no compelling reason to consider other Fixed-Priority scheduling algorithms. For this reason, the study will be restricted to DM only.

DM assigns the highest priority to the task with the smallest relative deadline and the lowest priority to the task with the largest relative deadline. When the relative deadline of each task is identical to its period, DM converges to the Rate-Monotonic (RM) algorithm due to Liu and Layland [B-2]. DM and RM can be implemented by attaching the priority of the task to the hardware interrupt level; i.e., the task with the highest priority is assigned to the highest level interrupt. Scheduling is implemented by hardware interrupt, and context switching is done in the interrupt handling routine. Thus, the overhead involved in scheduling can be kept to a minimum.

Table B-2 shows a real-time *TS* whose DM schedule is shown in figure B-2.

TABLE B-2. ANOTHER REAL-TIME TASK SYSTEM (EXAMPLE 2)

T_i	s_i	e_i	d_i	p_i
T_1	0	1	2	2
T_2	1	1	4	4
T_3	0	1	8	8

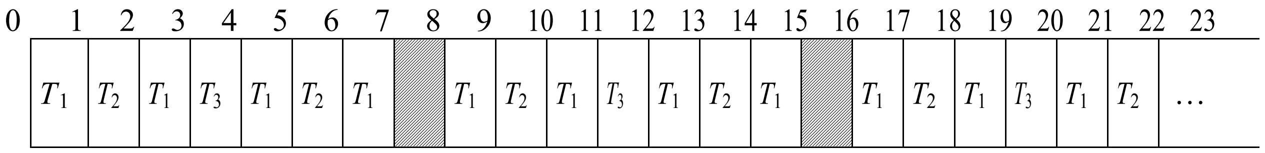
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
T_1 T_2 T_1 T_3 T_1 T_2 T_1 T_1 T_2 T_1 T_3 T_1 T_2 T_1 T_1 T_2 T_1 T_3 T_1 T_2 …

FIGURE B-2. DEADLINE-MONOTONIC SCHEDULE OF THE TASK SYSTEM IN TABLE B-2

The question of determining if a real-time *TS* is schedulable on one processor by DM is considered. This is tantamount to determining if the schedule produced by DM is valid. Liu and Layland [B-2] gave an effective procedure for this, as the next theorem shows.

Theorem 5: The schedule produced by DM is valid if the deadline of the first request of each task is met when all tasks make their first request simultaneously at the same time.

The procedure to determine if a real-time *TS* is schedulable by DM consists of constructing a schedule from time 0 when all tasks make their first request and checking if the deadline of the first request of each task is met. The *TS* is schedulable by DM if the deadline of each task is met. This is a sufficient condition for all situations, even if the initial request times (s_i) of the tasks are not identical.

So far, the study had been assuming that tasks make requests periodically. In some situations, tasks may make requests at random times, but it is guaranteed that two consecutive requests of the same task, say T_i, are separated by a minimum time interval, say p_i. These kind of tasks will be called sporadic tasks, while tasks defined earlier will be called periodic tasks. Note that DM is also optimal and Theorem 5 is also applicable for sporadic tasks.

Liu and Layland [B-2] gave a sufficiency test for a real-time *TS* to be schedulable by DM (or RM). Their sufficiency test assumes that the relative deadline of each task is identical to its period. (Under this assumption, DM and RM are identical.) As the following theorem shows, this gives a test to check schedulability faster than the above method.

Theorem 6: Let *TS* be a real-time *TS* consisting of *n* real-time tasks, where each task's relative deadline is identical to its period. *TS* is schedulable by DM (or RM) if $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq n(2^{1/n} - 1)$.

A simple test for schedulability consists of comparing the sum of the utilization factors $\left(\frac{e_i}{p_i}\right)$ of the *n* tasks against the value $n(2^{1/n} - 1)$. If the sum of the utilization factors is less than or equal to $n(2^{1/n} - 1)$, then the real-time *TS* is schedulable by DM (or RM). Otherwise, there is no conclusive evidence; it may or may not be schedulable. The function $n(2^{1/n} - 1)$ is a decreasing function of *n*. When *n* approaches infinity, the function approaches $ln\ 2 = 0.69....$

Leung and Whitehead [B-7] showed that DM is not optimal for $m > 1$ processors. At the present time, no simple algorithm is known to be optimal for more than one processor. Researchers have considered the partition approach in scheduling on multiprocessor systems. Unfortunately, theorem 7 was shown by Leung and Whitehead [B-7].

Theorem 7: The problem of partitioning a set of real-time tasks into a minimum number of groups so that each group is schedulable by DM on one processor is NP-hard.

Theorem 7 shows that it is extremely unlikely that a fast algorithm exists to schedule a set of real-time tasks on a minimum number of processors. Motivated by the computational complexity, Dhall and Liu [B-9] considered two heuristics, the Rate-Monotonic-Next-Fit (RMNF) and the Rate-Monotonic-First-Fit (RMFF) algorithms, both of them are adapted from bin packing heuristics. They showed that the worst-case bounds for RMNF and RMFF are 2.67 and $\frac{4 \times 2^{1/3}}{\left(1 + 2^{1/3}\right)}$, respectively. Specifically, they showed the following theorem.

Theorem 8: Let N_{RMNF} and N_{RMFF} be the numbers of processors required to feasibly schedule a set of tasks by the RMNF and RMFF algorithms, respectively, and let N_{OPT} be the minimum number required. Then, as N_{OPT} approaches infinity, $2.4 \leq \frac{N_{RMNF}}{N_{OPT}} \leq 2.67$ and $2 \leq \frac{N_{RMFF}}{N_{OPT}} \leq \frac{4 \times 2^{1/3}}{(1+2^{1/3})}$.

B.3 FIXED-PRIORITY SCHEDULING WITH LIMITED PRIORITY LEVELS.

In section B.2, it was assumed that the system has as many priority levels as the number of real-time tasks. Consequently, each task can be assigned a distinct priority. In practice, the number of priority levels in a computer system is very limited and is far exceeded by the number of real-time tasks. Thus, several tasks need be mapped to the same priority level. In the remainder of this section, it is assumed that there are *m* priority levels into which *n* tasks are mapped, where $m < n$. When tasks with the same priority make a request, it is assumed that they are scheduled in a first-come, first-serve (FCFS) manner. This is usually how the hardware service interrupts.

Two important questions naturally arise. First, how does one determine if a given priority assignment produces a valid schedule? Note that several tasks at the same priority level may simultaneously make a request, and it is not clear how to characterize the worst-case scenario. Second, what is an optimal priority assignment? Here, an optimal priority assignment means that it can always schedule any set of tasks that can be feasibly scheduled on *m* priority levels.

Pertaining to the first question, theorem 5 (stated in section B.2) is still applicable to this model, but one more condition needs to be satisfied as well. The additional condition is that tasks at the same priority level are serviced in the reverse order of their relative deadlines; *i*.e., the task with the largest relative deadline is serviced first and the task with the smallest relative deadline is serviced last. This is because tasks at the same priority level may make requests in the reverse order of their relative deadlines. Since they are scheduled in a FCFS manner, this represents the worst-case scenario. Thus, the following theorem exists.

Theorem 9: The schedule produced by a given priority assignment is valid if the deadline of the first request of each task is met when all tasks make their first request simultaneously at the same time, with the stipulation that tasks at the same priority level are serviced in the reverse order of their relative deadlines (i.e., tasks with the largest relative deadline serviced first).

An alternative method is to carry out time-demand analysis, as described in reference B-10. Let G denote a priority assignment, and let G_i denote the set of tasks having priority i, $1 \le i \le m$. By convention, assume that priority 1 is the highest priority and priority m is the lowest. Suppose there are k tasks in G_i. Use $G_{i,j}$ to denote the j-th task in the set. Without loss of generality, assume that the relative deadline of $G_{i,j}$ is less than or equal to that of $G_{i,j+1}$, for each $1 \le j \le k$.

For a given task $T_a \in G_i$, suppose it makes a request at time 0, along with all tasks at an equal or higher priority than T_a. Then, the total time demand $w_a(t)$ of this request of T_a, along with all tasks at an equal or higher priority than T_a in the time interval [0, t) is:

$$w_a(t) = \sum_{T_l \in G_i} e_l + \sum_{T_l \in G_j}^{j<i} \left\lceil \tfrac{t}{p_l} \right\rceil e_l, \text{ for } 0 < t \le d_a$$

Task T_a can meet its deadline if, and only if, there is a time instant t, $0 < t \le d_a$, such that $w_a(t) \le t$. Thus, all one needs to do is to check those time instants, t, which are integral multiples of the periods of some tasks belonging to G_j, $1 \le j < i$, to see if there is a time instant t such that $w_a(t) \le t$.

It is easy to see that for the task set G_i, if $G_{i,1}$ can meet its deadline, then all other tasks in G_i can also meet its deadline. Therefore, to show that the schedule produced by a given priority assignment is valid, all one needs to show is that $G_{i,1}$ can meet its deadline using the time-demand analysis, for all i = 1, 2, . . . , m.

The running time of the above procedure is a polynomial function of *m, n* and $\max\{d_i\}$. It is a pseudo-polynomial time algorithm. A pseudo-polynomial time algorithm is one that runs in polynomial time with respect to the size of the input, provided that all integer parameters are represented in unary (base 1) notation. (Note: normally, integer parameters are represented in binary (base 2) notation. The effect of representing integers in unary is to inflate the size of the input so that an exponential-time algorithm looks like polynomial.)

Again, pertaining to the second question, recall that DM assigns the highest priority to the task with the smallest relative deadline and that DM is optimal when there are as many system priority levels as the number of tasks. One can adapt DM to systems with limited priority levels as follows. Assign the highest priority to the task with the smallest relative deadline. Keep on assigning the same priority to the task with the next smallest relative deadline until it is infeasible (according to the time-demand analysis given above), at which point assign the next priority level to the task. This assignment is called the Deadline-Monotonic-with-Limited-Priority-Level (DM-LPL) assignment.

Shown below is algorithm DM-LPL. It tries to assign *m* priority levels to *n* tasks, where $m < n$. If the resulting assignment is not valid, it will output Not schedulable. In the algorithm, G_i denotes the set of tasks with priority level *i*, $1 \le i \le m$; it is assumed that priority level 1 is the highest priority and priority level *m* is the lowest.

```
Sort the jobs in ascending order of their relative deadlines; i.e., $d_1 \le d_2 \le ... \le d_n$.
Let $G_i = \varnothing$ for all i, $1 \le i \le m$.
$i = 1$.
For $j = 1$ to $n$
    $G_i = G_i \cup \{T_j\}$.
    Use time-demand analysis to check if $G_{i,1}$ can meet its deadline.
    If $G_{i,1}$ cannot meet its deadline, then
        $G_i = G_i - \{ T_j \}$.
        If $i + 1 > m$, then output “Not Schedulable” and exit,
        else
            $G_{i+1} = G_{i+1} \cup \{T_j\}$
            Use time demand analysis to check if $G_{i+1,1}$ can meet its deadline.
            If $G_{i+1,1}$ cannot meet its deadline,
                then output ”Not Schedulable” and exit.
            Else $i = i + 1$.
Output $G_1, G_2, ... G_m$.
```

The running time of algorithm DM-LPL is a polynomial function of *n, m*, and $\max\{d_i\}$. It is a pseudo-polynomial time algorithm. The algorithm was implemented in C language. The source code has been tested using a Tornado computer. In reference to this algorithm, see the implementation show in appendix C.

DM-LPL will be proved to be optimal in the sense that any *TS* feasible on one processor with *m* system priority levels is also schedulable by DM-LPL. Before proving the theorem, the next lemma will be proved first. Lemma 1 shows that if a *TS* is feasible, there is always a priority assignment such that tasks with small relative deadlines have equal or higher priority than tasks with large relative deadlines.

Lemma 1: If *TS* is feasible on one processor with *m* system priority levels, then there is a valid priority assignment such that tasks with small relative deadlines have equal or higher priority than tasks with large relative deadlines. In other words, there is no task with a large relative deadline having a higher priority than a task with a small relative deadline.

Proof. Since *TS* is feasible, there must be a valid priority assignment, say *G*, for the set of tasks. If *G* satisfies the property of the lemma, then the lemma is proven. Otherwise, there must be two tasks, T_a and T_b with $d_a < d_b$ such that $T_b \in G_k$ and $T_a \in G_{k+1}$. Create a new priority assignment G' such that T_b has the same priority as T_a and G' is still valid. Repeating this argument will finally prove Lemma 1.

$G' = \{G_1^{'}, G_2^{'}, \ldots G_m^{'}\}$ is defined as: $G_k^{'} = G_k - \{T_b\}$, $G_{k+1}^{'} = G_{k+1} \cup \{T_b\}$, and for all others, $i \neq k, k+1$, $G_i^{'} = G_i$. G' will be shown to be a valid priority assignment next.

Note that the only priority change in G' is T_b; the priority of all other tasks remain the same as before. Since a lower priority is assigned to T_b in G', it is clear that all tasks having equal or higher priority as T_b in *G* can still meet their deadlines. Similarly, all tasks having lower priority than T_a in *G* can also meet their deadlines, since the priority change of T_b will not affect their operation. Therefore, it remains to prove that the tasks in $G_{k+1}^{'}$ can meet their deadlines under the new priority assignment. To prove this, it is sufficient to show that the task $G_{k+1,1}^{'}$ can still meet its deadline. Note that $G_{k+1,1}^{'}$ is the same task as $G_{k+1,1}$. Let this task be T_c.

Since T_c can meet its deadline under *G*, there must be a time instant t_1, $0 < t_1 \leq d_c$, such that the following inequality holds

$$w_c(t_1) = \sum_{T_i \in G_{k+1}} e_i + \sum_{T_i \in G_j}^{j<k+1} \left\lceil \tfrac{t_1}{p_i} \right\rceil e_i \leq t_1$$

Under G', the maximum response time for T_c is

$$w_c^{'}(t) = \sum_{T_i \in G_{k+1}^{'}} e_i + \sum_{T_i \in G_j^{'}}^{j<k+1} \left\lceil \tfrac{t}{p_i} \right\rceil e_i, \quad \text{for} \quad 0 < t \leq d_c$$

By assumption, $d_c \le d_a < d_b \le p_a$. Moreover, $\left\lceil \frac{t_1}{p_b} \right\rceil = 1$, since $t_1 \le d_c \le p_b$. Thus, the maximum response time for T_c at time t_1 is

$$
\begin{aligned}
w_c'(t_1) &= \sum_{T_i \in G_{k+1}'} e_i + \sum_{T_i \in G_j'}^{j<k+1} \left\lceil \frac{t_1}{p_i} \right\rceil e_i \\
&= \sum_{T_i \in G_{k+1}} e_i + e_b + \sum_{T_i \in G_j'}^{j<k+1} \left\lceil \frac{t_1}{p_i} \right\rceil e_i \\
&= \sum_{T_i \in G_{k+1}} e_i + \left\lceil \frac{t_1}{p_b} \right\rceil e_b + \sum_{T_i \in G_j'}^{j<k+1} \left\lceil \frac{t_1}{p_i} \right\rceil e_i \\
&= \sum_{T_i \in G_{k+1}} e_i + \sum_{T_i \in G_j}^{j<k+1} \left\lceil \frac{t_1}{p_i} \right\rceil e_i \\
&= w_c(t_1) \\
&\le t_1 \\
&\le d_c .
\end{aligned}
$$

But this means that T_c can meet its deadline under G'.

Theorem 10: DM-LPL is an optimal priority assignment for one processor.

Proof. Let *TS* be a feasible task system on one processor and let *G* be a valid priority assignment for *TS*. By Lemma 1, it can be assumed that in *G* tasks with small relative deadlines have equal or higher priority than tasks with large relative deadlines. If *G* is identical to the priority assignment obtained by DM-LPL, then the lemma is proved. Otherwise, there must be a task T_b in G_{k+1} that can be feasibly assigned to G_k. Without loss of generality, assume that T_b is $G_{k+1,1}$. Let $G_{k,1}$ be T_a. Since T_b can be feasibly assigned to G_k, there must be a time instant t_1, $0 < t_1 \le d_a$, such that $w_a(t_1) + e_b \le t_1$. Construct a new priority assignment $G' = \{G_1', G_2', \ldots G_m'\}$ such that $G_k' = G_k \cup \{T_b\}$, $G_{k+1}' = G_{k+1} - \{T_b\}$, and for all other $i \ne k, k+1$, $G_i' = G_i$.

One can claim that G' is also a valid priority assignment. It is easy to see that all tasks in G_i', $i \ne k, k+1$, can still meet their deadlines, since $G_i' = G_i$ for $i \ne k, k+1$. Suppose $G_{k+1,1}'$ is T_c. If it can be proved that both T_a and T_c can meet their deadlines under G', then G' is also valid.

By assumption, T_b can be feasibly assigned to G_k. Therefore, under the new assignment G', T_a can still meet its deadline. Since T_b can meet its deadline under *G*, there must be a time instant t_1, $0 < t_1 \le d_b$, such that

$$\begin{aligned} w_b(t_1) &= \sum_{T_i \in G_{k+1}} e_i + \sum_{T_i \in G_j}^{j<k+1} \left\lceil \tfrac{t_1}{p_i} \right\rceil e_i \\ &= \sum_{T_i \in G'_{k+1}} e_i + e_b + \sum_{T_i \in G_j}^{j<k+1} \left\lceil \tfrac{t_1}{p_i} \right\rceil e_i \\ &\le t_1 \end{aligned}$$

For T_c, the maximum response time under G' is

$$w'_c(t_1) = \sum_{T_i \in G'_{k+1}} e_i + \sum_{T_i \in G'_j}^{j<k+1} \left\lceil \tfrac{t_1}{p_i} \right\rceil e_i \quad \text{for } 0 < t \le d_c .$$

Since $t_1 \le d_b \le p_b$, $\left\lceil \frac{t_1}{p_b} \right\rceil = 1$. Letting $t = t_1$, $w'_c(t_1) = w_b(t_1) \le t_1$. Since $t_1 \le d_b \le d_c$, T_c can meet its deadline under the new priority assignment *G*'.

B.4 PRIORITY ASSIGNMENT ON MULTIPROCESSORS WITH LIMITED PRIORITY LEVELS.

Theorem 6 shows that the processor utilization is at least 0.69 when the computing system has infinite priority levels. When the computing system has limited priority levels, however, the processor utilization can be quite poor. Consider *n* tasks with the following characteristics. All *n* tasks have their relative deadlines identical to their periods. Task 1 has its execution time much smaller than its period; i.e., $e_1 << p_1$. Task 2 has its execution time equal to the period of the first task (i.e., $e_2 = p_1$), but is much smaller than its own period (i.e., $e_2 << p_2$). Similarly, task 3 has its execution time equal to the period of the second task (i.e., $e_3 = p_2$), but is much smaller than its own period (i.e., $e_3 << p_3$). The remaining tasks have execution times and periods following the same pattern. Since $e_i + e_{i+1} > p_i$ for all $1 \le i < n$, no two tasks can be assigned the same priority level. Therefore, $\left\lceil \frac{n}{m} \right\rceil$ processors are needed, where each processor has exactly *m* priority levels. Notice that the processor utilization in each processor is very low. On the other hand, if the computing system has infinite priority levels, all *n* tasks can be scheduled on one processor by DM.

The above example reveals that the processor utilization can be quite low for some pathological situations. It is unlikely that this kind of situation occurs frequently in practice. The above example also suggests that more processors are necessary when computing systems with limited priority levels are used, compared with systems that have infinite priority levels. In this subsection, the problem of scheduling tasks on multiprocessors with limited priority levels is considered. The goal is to find the minimum number of processors necessary to schedule *n* tasks. Such an algorithm will be called an optimal algorithm.

In a search for an optimal algorithm, a natural candidate is the greedy algorithm, which works as follows.

- Sort the tasks in ascending order of their relative deadlines.
- Starting with the first task, schedule as many tasks as possible by the DM-LPL algorithm onto the current processor, until a task that cannot be feasibly scheduled by DM-LPL is encountered, at which point schedule the task onto a new processor (which now becomes the current processor).
- The above process is repeated until all tasks have been scheduled.

One wonders if the above greedy algorithm is optimal. Unfortunately, the answer is negative. Consider the following six tasks, all of which have their relative deadlines identical to their periods and all have their initial requests made at time 0. In the following, each task T_i is represented by an ordered pair (e_i, d_i): T_1 = (1, 5), T_2 = (2, 6), T_3 = (3, 9), T_4 = (5, 10), T_5 = (6, 16), and T_6 = (1, 20). Suppose each processor has two priority levels. The greedy algorithm yields three processors, with P_1: $G_1 = \{T_1, T_2\}$, $G_2 = \{ T_3 \}$; for P_2: $G_1 = \{T_4\}$, $G_2 = \{T_5\}$; and for P_3: $G_1 = \{T_6\}$. However, these six tasks can be scheduled on two processors such that P_1: $G_1 = \{T_1, T_3\}$, $G_2 = \{T_5\}$; for P_2: $G_1 = \{T_2\}$, $G_2 = \{T_4, T_6\}$.

As it turns out, the problem of finding the minimum number of processors is NP-hard. By a simple modification of the proof in reference B-7, the following theorem can be proved.

Theorem 11: The problem of finding the minimum number of processors with *m* priority levels to schedule a set of tasks is NP-hard.

Theorem 11 suggests that there is no efficient algorithm to schedule a set of tasks on the minimum number of processors. Motivated by the computational complexity of the problem, two heuristics, called First-Fit (FF) and First-Fit-Decreasing-Utilization (FFDU), respectively, are considered. The FF algorithm sorts the tasks in ascending order of their relative deadlines, while the FFDU algorithm sorts the tasks in ascending order of their utilization factors ($\frac{e_i}{p_i}$). Both algorithms try to schedule the next task into the lowest-indexed processor by the DM-LPL algorithm.

Algorithm FF:

Sort the tasks in ascending order of their deadlines.
Let *k* be the largest index of the processor to which tasks have been assigned. Initially, *k*=1.
For each processor P_i , let $L_i \leq m$ be the lowest priority index (highest priority level) to which tasks have been assigned. Initially, $L_i = 1$ for each P_i . Let G_j^i denote all the tasks assigned to priority index *j* on P_i . Initially, $G_j^i = \varnothing$ for each *i* and *j*.

```
For $j$ = 1 to $n$
    Assign task $T_j$ as follows.
    $i$ = 1 ($i$ is processor index).
    While $T_j$ has not been assigned
    Use time-demand analysis to test if $T_j$ can be assigned to $G^i_{L_i}$.
    If $T_j$ can be assigned to $G^i_{L_i}$
            Assign $T_j$ to $G^i_{L_i}$.
    Else if $L_i + 1 \leq m$ and $T_j$ can be assigned to $G^i_{L_i+1}$
                $L_i = L_i + 1$.
                Assign $T_j$ to $G^i_{L_i}$.
        Else if $i < k$
                $i = i + 1$.
            Else
                $k = k + 1$, $i = k$.
Output $k$.
```

The running time of FF is a polynomial function of *m*, *n*, and $\max\{d_i\}$. It is a pseudo-polynomial time algorithm.

Algorithm FFDU:

Sort the tasks in descending order of their utilization factors.
Let *k* be the largest index of the processor to which tasks have been assigned. Initially, k=1.
For each processor P_i, let G^i denote all the tasks assigned to P_i. Initially, $G^i = \varnothing$ for each *i*.

```
For $j$ = 1 to $n$
    Assign task $T_j$ as follows.
    $i$ = 1 ($i$ is processor index).
    While $T_j$ has not been assigned
    Use the DM-LPL algorithm to test if $T_j$ can be assigned to $P_i$.
    If $T_j$ can be assigned $P_i$
            Assign $T_j$ to $G^i_{L_i}$.
    Else if $i < k$
            $i = i + 1$.
        Else
            $k = k + 1$, $i = k$.
Output $k$.
```

The running time of FFDU is a polynomial function of *m, n* and $\max\{d_i\}$. It is a pseudo-polynomial time algorithm.

The worst-case bounds for FF and FFDU are not known yet.

B.5 UNIT-EXECUTION-TIME TASK SYSTEMS: UNLIMITED PRIORITY LEVELS.

In the previous section, it was shown that processor utilization can be quite low when the computing system has limited priority levels. The situation could be better if each task has identical execution time, say 1 unit. Unit-execution-time task systems are interesting in its own right, since it models bus communication where each packet of information takes 1 unit time to send. Here, the communication bus is the processor. In this section, the case where the computing systems have unlimited priority levels is considered.

A set of *n* real-time tasks, where each task has 1 unit of execution time (i.e., $e_i = 1$ for each $1 \le i \le n$) and the relative deadline of each task is identical to its period (i.e., $d_i = p_i$ for each $1 \le i \le n$), is considered. Thus, each task is completely characterized by its relative deadline (or equivalently, its period), which is assumed to be an integer. Assume that the tasks have been sorted in ascending order of their relative deadlines; i.e., $d_1 \le d_2 \le \ldots \le d_n$.

The goal is to find a utilization bound for *n* tasks, *U(n)*, such that if the *n* tasks have total utilization less than or equal to *U(n)*, then they are always schedulable on a single processor by DM. If the task set has total utilization larger than *U(n)*, then it may or may not be schedulable by DM, depending on the relationships among their relative deadlines. It is conjectured that:

$$U(n) = \tfrac{1}{n-1} + \tfrac{1}{n} + \tfrac{1}{n+1} + \ldots + \tfrac{1}{2n-3} + \tfrac{1}{2n}$$

A full proof has not been obtained. Two special cases as described in appendix B.5.1 and B.5.2 were proved.

Definition 1: A task set is said to perfectly utilize the processor with respect to a priority assignment if it is schedulable, but a decrease in the largest period will make the task set unschedulable under the same priority assignment.

B.5.1 PERFECT INSTANCE WHEN d_1 IS ARBITRARY.

Given the number of tasks *n* and $T_1(1, d_1)$ where $d_1 \le n-1$, construct $T_2(1,d_2)$, $T_3(1, d_3)$, …, $T_n(1,d_n)$ as follows.

For $2 \le i \le n-2$, d_i is the maximum integer such that

$$\begin{cases} d_i = \left\lceil \tfrac{d_i}{d_1} \right\rceil + (n+i-4) \\ t \le \left\lceil \tfrac{t}{d_1} \right\rceil + (n+i-4) \text{ for } t < d_i \end{cases} \qquad \text{(B-1)}$$

To define d_{n-1} and d_n, define two variables t_1 and t_2, which will also be referred throughout the report hereforth.

Let t_1 be the maximum integer such that

$$\begin{cases} t_1 = \left\lceil \frac{t_1}{d_1} \right\rceil + 2n - 5 \\ t \le \left\lceil \frac{t}{d_1} \right\rceil + 2n - 5 \text{ for } t < t_1 \end{cases} \tag{B-2}$$

Let t_2 be the minimum integer such that

$$\begin{cases} t_2 = \left\lceil \frac{t_2}{d_1} \right\rceil + 2n - 3 \\ t < \left\lceil \frac{t}{d_1} \right\rceil + 2n - 3 \text{ for } t < t_2 \end{cases} \tag{B-3}$$

If $t_2 = kd_2 + 1$ for some k, then $d_n = t_2 + 1$ and $d_{n-1} = t_1$. Otherwise, if $t_2 = t_1 + 2$, then $d_{n-1} = d_n = t_1 + 1 = t_2 - 1 = \left\lceil \frac{t_2}{d_1} \right\rceil + 2n - 4$; else $t_2 \ge t_1 + 3$, let $d_{n-1} = t_1$ and $d_n = t_2$.

For convenience, the constructed instance will be called the perfect instance for n and d_1. The constructed perfect instance will be shown to have several properties, and then the instance will be shown to perfectly utilize the processor.

1. For $2 \le i \le n-1$, d_i is not a multiple of d_1.

   Proof: Suppose $d_i = kd_1$ and $2 \le i \le n-2$, then by definition $d_i = \left\lceil \frac{d_i}{d_1} \right\rceil + (n+i-4)$. Since $d_i + 1 = \left\lceil \frac{d_i+1}{d_1} \right\rceil + (n+i-4)$, one can always increase d_i by 1 and still satisfy equation B-1. But this contradicts the fact that d_i is the maximum integer satisfying equation B-1.

   For $i = n-1$, by construction, either $d_{n-1} = t_1$ or $d_{n-1} = t_1 + 1$. In the former case, t_1 cannot be a multiple of d_1, because otherwise the same contradiction as above will arise. In the latter case, $d_{n-1} = t_1 + 1$, which means that $t_2 = t_1+2$. By equations B-2 and B-3, $\left\lceil \frac{t_1+2}{d_1} \right\rceil = \left\lceil \frac{t_1}{d_1} \right\rceil$. However, this holds only when t_1+1, is not a multiple of d_1. Therefore, in both cases, d_{n-1} is not a multiple of d_1.

2. d_2 satisfies the following equation:

$$d_2 = \frac{(n-1)d_1 - r}{d_1 - 1} \qquad \text{for some } r \text{ such that } 1 \le r \le d_1 - 1 \tag{B-4}$$

   Proof: It has already been shown that d_2 is not a multiple of d_1. So, assume that $d_2 = kd_1 + r$ for some integers k and r such that $1 \le r \le d_1 - 1$. According to the construction, d_2 satisfies equation B-1. Therefore, $d_2 = k + 1 + n - 2 = kd_1 + r$, which implies that $d_2 = \frac{(n-1)d_1 - r}{d_1 - 1}$. Throughout this section, when r is mentioned, by default, it means r as defined above.

3. For each $2 \le i \le n-3$, either $d_{i+1} = d_i + 1$ or $d_{i+1} = d_i + 2$. Hence, $d_i = d_2 + i - 2 + \left\lfloor \frac{i+r-3}{d_1-1} \right\rfloor$ for $3 \le i \le n-2$.

Proof: For $2 \le i \le n-3$, let $d_i = k_i d_1 + r_i$ for some integers k_i and r_i such that $1 \le r_i \le d_1 - 1$.

If $r_i \le d_1 - 2$, then $\left\lceil \frac{d_i}{d_1} \right\rceil = \left\lceil \frac{d_i+1}{d_1} \right\rceil$. It is easy to show that $d_i + 1$ satisfies equation B-1 for $i+1$, and it is indeed the maximum integer that satisfies equation B-1. Therefore, $d_{i+1} = d_i + 1$ and $r_{i+1} = r_i + 1$.

If $r_i = d_1 - 1$, then $\left\lceil \frac{d_i+2}{d_1} \right\rceil = \left\lceil \frac{d_i}{d_1} \right\rceil + 1$. Similarly, one can show that $d_i + 2$ is the maximum integer satisfying equation B-1 for $i+1$. So $d_{i+1} = d_i + 2$ and $r_{i+1} = 1$.

In summary, for $2 \le i \le n-3$,

$$d_i = d_2 + i - 2 + \left\lfloor \frac{i+r-3}{d_1-1} \right\rfloor \qquad \text{for } 3 \le i \le n-2 \tag{B-5}$$

4. Either $t_1 = 2d_2 - 1$, or $t_1 = 2d_2 - 2$, or $t_1 = 2d_2 - 3$. Either $t_2 = 2d_2$ or $t_2 = 2d_2 + 1$.

Proof: Let $d_2 = kd_1 + r$ for some integers k and $1 \le r < d_1$. It is desirable to express t_1 in terms of d_2.

- $r = 1$. Then,

$$\begin{aligned} 2d_2 - 3 &= 2\left(\left\lceil \frac{d_2}{d_1} \right\rceil + n - 2\right) - 3 \\ &= \left(\left\lceil \frac{2d_2-3}{d_1} \right\rceil + 2\right) + 2n - 7 \\ &= \left\lceil \frac{2d_2-3}{d_1} \right\rceil + 2n - 5 \end{aligned}$$

One can easily show that $t \le \left\lceil \frac{t}{d_1} \right\rceil + 2n - 5$ for $t < 2d_2$-3 and $t > \left\lceil \frac{t}{d_1} \right\rceil + 2n - 5$ for $t > 2d_2$-3. Therefore, by the definition of t_1, $t_1 = 2d_2$ -3.

- $r < \frac{d_1+2}{2}$.

$$\begin{aligned}2d_2 - 2 &= 2\left(\left\lceil \tfrac{d_2}{d_1} \right\rceil + n - 2\right) - 2\\ &= \left(\left\lceil \tfrac{2d_2-2}{d_1} \right\rceil + 1\right) + 2n - 6\\ &= \left\lceil \tfrac{2d_2-2}{d_1} \right\rceil + 2n - 5\end{aligned}$$

As in above, it can be shown that $2d_2$-2 is the maximum integer satisfying equation B-2. Hence, by definition $t_1 = 2d_2$-2 in this case.

- $r > \frac{d_1+1}{2}$.

$$\begin{aligned}2d_2 - 1 &= 2\left(\left\lceil \tfrac{d_2}{d_1} \right\rceil + n - 2\right) - 1\\ &= \left(\left\lceil \tfrac{2d_2-1}{d_1} \right\rceil + 2\right) + 2n - 5\\ &= \left\lceil \tfrac{2d_2-1}{d_1} \right\rceil + 2n - 5\end{aligned}$$

Similarly, it can be shown that $t_1 = 2d_2$-2 in this case.

Next, express t_2 in terms of d_2.

- $r \le \frac{d_1}{2}$.

$$\begin{aligned}2d_2 &= 2\left(\left\lceil \tfrac{d_2}{d_1} \right\rceil + n - 2\right)\\ &= \left\lceil \tfrac{2d_2}{d_1} \right\rceil + 1 + 2(n-2)\\ &= \left\lceil \tfrac{2d_2}{d_1} \right\rceil + 2n - 3\end{aligned}$$

- $r > \frac{d_1}{2}$.

$$\begin{aligned}2d_2 + 1 &= 2\left(\left\lceil \tfrac{d_2}{d_1} \right\rceil + n - 2\right) + 1\\ &= \left\lceil \tfrac{2d_2}{d_1} \right\rceil + 2n - 3\\ &= \left\lceil \tfrac{2d_2+1}{d_1} \right\rceil + 2n - 3\end{aligned}$$

Using the same argument as for t_1 and by the definition of t_2, it can be shown that $t_2 = 2d_2$ when $r \le \frac{d_1}{2}$ and $t_2 = 2d_2+1$ when $r > \frac{d_1}{2}$.

According to the construction of d_n, if $t_2 = t_1 + 2$, then $d_n = d_{n-1} = t_2 - 1$. In summary, tables B-3 and B-4 show the relationships between d_{n-1}, d_n and d_2.

TABLE B-3. RELATIONSHIP BETWEEN d_{n-1}, d_n, AND d_2 WHEN d_1 IS EVEN

	t_1	t_2	d_{n-1}	d_n
$r = 1$	$2d_2$-3	$2d_2$	$2d_2$-3	$2d_2$
$2 \le r \le \frac{d_1}{2}$	$2d_2$ - 2	$2d_2$	$2d_2$ - 1	$2d_2$ - 1
$\frac{d_1+2}{2} \le r \le d_1 - 1$	$2d_2$ - 1	$2d_2$ + 1	$2d_2$ - 1	$2d_2$+2

TABLE B-4. RELATIONSHIP BETWEEN d_{n-1}, d_n, AND d_2 WHEN d_1 IS ODD

	t_1	t_2	d_{n-1}	d_n
$r = 1$	$2d_2 - 3$	$2d_2$	$2d_2 - 3$	$2d_2$
$2 \le r \le \frac{d_1-1}{2}$	$2d_2$ - 2	$2d_2$	$2d_2$ - 1	$2d_2$ - 1
$r = \frac{d_1+1}{2}$	$2d_2$ - 2	$2d_2$ + 1	$2d_2$ - 2	$2d_2$+2
$\frac{d_1+3}{2} \le r \le d_1 - 1$	$2d_2$ - 1	$2d_2$ + 1	$2d_2$ - 1	$2d_2$+2

5. Either $d_n = k_n d_1$ or $d_n = k_n d_1 + r_n$ for some integers k_n and r_n, where $r_n \ge 2$.

Proof: There are three cases depending on the relationship between d_n and d_2.

(i) $d_n = d_{n-1} = 2d_2 - 1$. From tables B-3 and B-4, $d_n = d_{n-1}$ happens when $2 \le r \le \frac{d_1-1}{2}$ if d_1 is even or when $2 \le r \le \frac{d_1}{2}$ if d_1 is odd. In either case, $d_n = 2d_2 - 1 = 2kd_1 + 2r - 1$ and $3 \le 2r - 1 \le d_1 - 1$. Therefore $r_n = 2r - 1 \ge 3$.

(ii) $d_n = 2d_2$. From tables B-3 and B-4, $r = 1$, i.e., $d_2 = kd_1 + 1$. Therefore, $d_n = 2kd_1 + 2$. If $d_1 = 2$, then $r_n = 0$; otherwise, $r_n = 2$.

(iii) $d_n = 2d_2 + 2 = 2(kd_1 + r) + 2 = (2k+1)d_1 + (2r - d_1 + 2)$. If $r = d_1 - 1$, then $r_n = 0$; otherwise, if d_1 is even, then $\frac{d_1+2}{2} \le r$ and $r_n \ge 4$, else $\frac{d_1+1}{2} \le r$ and $r_n = 3$.

6. The perfect instance $T_1, T_2, \ldots, T_n$ constructed as above perfectly utilizes the processor.

Proof: Some characterizations about the deadline monotonic schedule of the perfect instance will be given first. The claim follows directly from these characterizations and the definition of perfectly utilization.

a. There is no idle time before t_1.

It is enough to prove that at any time $t < t_1$, the number of requests R from T_1, T_2, … , T_{n-1} is greater than or equal to t. There are two cases depending on t.

- $t = d_i$.
  If $i = 1$, at time $t = d_1$, $R = n - 1$. Since $d_1 \le n-1$, $t \le R$. For $2 \le i \le n-2$, $R = \left\lceil \frac{d_i}{d_1} \right\rceil + 2(i-2) + (n-i) = \left\lceil \frac{d_i}{d_1} \right\rceil + (n+i-4)$. By construction, $t = d_i = R$.

- $d_i < t < d_{i+1}$.

  It has already been shown either $d_{i+1} = d_i + 1$ or $d_{i+1} = d_i + 2$ for $2 \le i \le n-2$. So, $t = d_i + 1 = \left\lceil \frac{d_i}{d_1} \right\rceil + n + i - 3 < d_{i+1}$ in this case. Therefore, at time t, $T_{i+1}, T_{i+2}, ..., T_{n-1}$ each makes one request, while $T_2, ..., T_i$ each makes exactly two requests. Hence, the total number of requests $R = \left\lceil \frac{d_i+1}{d_1} \right\rceil + 2(i-1) + (n-i-1) = \left\lceil \frac{d_i+1}{d_1} \right\rceil + (n+i-3)$. It is obvious that $t \le R$.

b. All requests from T_2, … , T_{n-1} during the period [0, t_1) have been executed at t_1.

From the construction, $d_{n-2} < t_1 \le d_{n-1} < 2d_2 < 2d_i$ for 3 < i < n. So, at time t_1, $T_2, ..., T_{n-2}$ all make exactly two requests, and T_{n-1} makes one request. Therefore, the total number of requests made by T_1, T_2, … , T_{n-1} is

$$R = \left\lceil \frac{t_1}{d_1} \right\rceil + 2(n-3) + 1 = \left\lceil \frac{t_1}{d_1} \right\rceil + 2n - 5$$

which is equal to t_1. Therefore, all requests from $T_2, ..., T_{n-1}$ have been executed.

c. None of $T_3, ..., T_{n-2}$ will make any request during the period [t_1, d_n-1].

First, $d_i < t_1$, 3 ≤ i ≤ n-2. So T_3, … , T_{n-2} have all made their second requests before t_1. On the other hand, $t_1 < d_n - 1 < 2d_i$. Therefore, T_3, … , T_{n-2} will not make their third requests from t_1 until $d_n - 1$.

d. There is no idle time during the period [t_1, d_n-1] and there is no request from $T_1, T_2, ..., T_{n-1}$ at d_n-1. Therefore, the first request from T_n will be executed at d_n-1 and finished just before its deadline d_n.

By (c), $T_3,\ldots,T_{n-2}$ will not make any request during the period $[t_1, d_n-1)$, neither will these tasks make any request at d_n-1. Therefore, it remains to show that the requests from T_1, T_2, and T_{n-1} will occupy the period $[t_1, d_n-1)$ and will not make any requests at d_n-1.

It has been shown that t_1 is not a multiple of d_1 and $t_1 < 2d_2$. So, neither T_1 nor T_2 will make a request at t_1. For T_{n-1}, there are two cases depending on the relationship between t_1 and d_{n-1}:

- $d_{n-1} = d_n = t_1 + 1$. Then, $t_1 = d_n-1 < d_{n-1}$. Therefore, T_{n-1} will not make a request at t_1. Hence, the first request of T_n will be executed at t_1 and the claim is true.

- $d_{n-1} = t_1$. Task T_{n-1} makes the second request at t_1 and the request is executed right away. Again, There are the following cases:

  - $t_1 = 2d_2 - 3$ and $t_2 = 2d_2$.

    Then, $d_{n-1} = 2d_2 - 3$, $d_n = 2d_2$, and $d_n - 1 = 2d_2 - 1$. From tables B-3 and B-4, $r = 1$. Therefore, $t_1 + 1 = 2(kd_1 + 1) - 3 + 1 = 2kd_1$. So, T_1 must make a request at t_1+1. T_2 will not make a request before $2d_2$. Therefore, at times $2d_2 - 3$ and $2d_2 - 2$, the processor will execute the requests from T_{n-1} and T_1, respectively.

  - $t_1 = 2d_2 - 1$ and $t_2 = 2d_2 + 1$.

    Then, $d_{n-1} = 2d_2 - 1$, $d_n = 2d_2 + 2$, and $d_n - 1 = 2d_2 + 1$. T_1 will not make a request at $t_1 + 1 = 2d_2$ since d_2 is not a multiple of d_1, nor will it make a request at $d_n - 1 = 2d_2 + 1$ because it has been proved that d_n is either a multiple of d_1 or a multiple of d_1 plus an integer greater than 1.

    T_2 makes a request at $2d_2$.

    Therefore, at times $2d_2 - 1$ and $2d_2$, the processor will execute the requests from T_{n-1} and T_2, respectively.

  - $t_1 = 2d_2 - 2$ and $t_2 = 2d_2 + 1$. Then, $d_{n-1} = 2d_2 - 2$, $d_n = 2d_2 + 2$, $r = \frac{d_1+1}{2}$ and

    $$t_1 + 1 = 2d_2 - 1 = 2(kd_1 + \tfrac{d_1+1}{2}) - 1 = (2k+1)d_1.$$

So, T_1 will make a request at $2d_2$ -1, and it will not make a request at $2d_2$ and $2d_2 + 1$.

T_2 makes a request at $2d_2$.

Therefore, at times $2d_2$-2, $2d_2$-1, and $2d_2$, the processor will execute the requests from T_{n-1}, T_1, and T_2, respectively.

Claim 1: Given *n,* the perfect instance gets the minimum utilization when $d_1 = n$-1.

Proof: For $n \le 5$, the claim can be proved by just enumerating all task sets. Tables B-5 and B-6 show the perfect instances for $n = 4$ and $n = 5$, respectively. For $n = 3$, the claim is obviously true because there is only one possible value of d_1, which is 2. Therefore, assume that $n \ge 6$ in the following.

TABLE B-5. PERFECT TASK SET INSTANCES WHEN $n = 4$

d_1	d_2	d_3	d_4	U
2	5	7	10	0.942857
3	4	5	8	0.908333

TABLE B-6. PERFECT TASK SET INSTANCES WHEN $n = 5$

d_1	d_2	d_3	d_4	d_5	U
2	7	9	11	14	0.916306
3	5	7	8	12	0.884524
4	5	6	7	10	0.859524

For each n and $d_1 \le n-1$, let the corresponding $d_2 = \frac{(n-1)d_1 - r}{d_1 - 1}$ for some $1 \le r \le d_1 - 1$.

Let

$$\begin{aligned} U(d_1) &= \frac{1}{d_1} + \frac{1}{d_2} + \left(\sum_{i=3}^{n-2} \frac{1}{d_i} \right) + \frac{1}{d_{n-1}} + \frac{1}{d_n} \\ &= \frac{1}{d_1} + \frac{1}{d_2} + \left(\sum_{i=3}^{n-2} \frac{1}{d_2 + i - 2 + \left\lfloor \frac{i+r-3}{d_1-1} \right\rfloor} \right) + \frac{1}{d_{n-1}} + \frac{1}{d_n} \end{aligned} \qquad \text{(B-6)}$$

It will be proved that $U(d_1)$ attains its minimum when $d_1 = n-1$ by showing that (1) the minimum of $U(d_1)$ occurs when $d_1 > \lceil \frac{n}{2} \rceil$ and (2) $U(d_1+1) < U(d_1)$ for $\lceil \frac{n}{2} \rceil \le d_1 \le n-2$.

1. The minimum of $U(d_1)$ occurs when $d_1 > \lceil \frac{n}{2} \rceil$.

From tables B-3 and B-4,

$$\frac{1}{d_{n-1}} + \frac{1}{d_n} \ge \frac{1}{2d_2 - 1} + \frac{1}{2d_2 + 2} \quad .$$

Applying the above inequality to equation B-6 and removing the floor function,

$$\begin{aligned} U(d_1) &\ge \frac{1}{d_1} + \frac{1}{d_2} + \left(\sum_{i=3}^{n-2} \frac{1}{d_2 + i - 2 + \frac{i+r-3}{d_1 - 1}} \right) + \frac{1}{2d_2 - 1} + \frac{1}{2d_2 + 2} \\ &= \frac{1}{d_1} + \frac{1}{d_2} + \left(\sum_{i=3}^{n-2} \frac{d_1 - 1}{(n+i-3)d_1 - 1} \right) + \frac{1}{2d_2 - 1} + \frac{1}{2d_2 + 2} \end{aligned}$$

For convenience, let

$$f(d_1) = \frac{1}{d_1} + \frac{1}{d_2} + \left(\sum_{i=3}^{n-2} \frac{d_1 - 1}{(n+i-3)d_1 - 1} \right) + \frac{1}{2d_2 - 1} + \frac{1}{2d_2 + 2} \qquad \text{(B-7)}$$

Then $U(d_1) \ge f(d_1)$.

Next, it will be proved that $f(d_1)$ is a decreasing function and that $U(n-1) < f\left(\lceil \frac{n}{2} \rceil\right)$.

a. $f(d_1)$ decreases with respect to d_1.

$$\begin{aligned}
f'(d_1) &= -\frac{1}{d_1^2} + \frac{(n-1-r)}{\left[(n-1)d_1 - r\right]^2} + \sum_{i=3}^{n-2} \frac{n+i-4}{\left[(n+i-3)d_1 - 1\right]^2} \\
&\quad + \frac{2n-2r-2}{\left[(2n-3)d_1 - 2r+1\right]^2} + \frac{n-r-1}{2\left(nd_1 - r - 1\right)^2} \\
&< -\frac{1}{d_1^2} + \frac{(n-1-1)}{\left[(n-1)d_1 - d_1\right]^2} + \sum_{i=3}^{n-2} \frac{n+i-4}{\left[(n+i-3)d_1 - d_1\right]^2} \\
&\quad + \frac{2n-2-2}{\left[(2n-3)d_1 - 2d_1\right]^2} + \frac{n-1-1}{2\left(nd_1 - d_1 - d_1\right)^2} \\
&< \frac{1}{d_1^2}\left(-1 + \frac{(n-2)}{\left[(n-1)-1\right]^2} + \sum_{i=3}^{n-2} \frac{n+i-4}{\left[(n+i-3)-1\right]^2} \right. \\
&\quad \left. + \frac{2n-4}{(2n-5)^2} + \frac{n-2}{2(n-2)^2} \right) \\
&= \frac{1}{d_1^2}\left(-1 + \frac{1}{(n-2)} + \sum_{i=3}^{n-2} \frac{1}{(n+i-4)} + \frac{2n-4}{(2n-5)^2} + \frac{1}{2(n-2)} \right) \\
&< 0
\end{aligned}$$

In summary, for any *n*, $f(d_1)$ strictly decreases when d_1 increases. When $d_1 = n-1$, $f(d_1)$ has the minimum value.

b. $U(n-1) < f\left(\left\lceil \frac{n}{2} \right\rceil\right) \le U\left(\left\lceil \frac{n}{2} \right\rceil\right)$.

The proof is by showing that $f\left(\left\lceil \frac{n}{2} \right\rceil\right) - U(n-1) > 0$. It has been shown that:

$$U(n-1) = \frac{1}{n-1} + \frac{1}{n} + \left(\sum_{i=3}^{n-2} \frac{1}{n+i-2} \right) + \frac{1}{2n-3} + \frac{1}{2n} \tag{B-8}$$

Let $d_1 = \left\lceil \frac{n}{2} \right\rceil$. It is desirable to remove the floor function in $f(d_1)$ (see equation B-7). There are two cases, depending on whether *n* is odd or even.

- *n* is even.

  Then, $d_1 = \left\lceil \frac{n}{2} \right\rceil = \frac{n}{2}$. By equation B-1, $d_2 = n+1$ and $r = 1$. Applying to equation B-7,

$$
\begin{aligned}
f\left(\left\lceil \tfrac{n}{2} \right\rceil\right) &= f\left(\tfrac{n}{2}\right) \\
&= \frac{1}{\frac{n}{2}} + \frac{1}{n+1} + \left(\sum_{i=3}^{n-2} \frac{{}^{n}\!/\!{}_{2} - 1}{(n+i-3)\,{}^{n}\!/\!{}_{2} - 1} \right) + \frac{1}{2n+1} + \frac{1}{2n+4} \qquad \text{(B-9)} \\
&= \frac{2}{n} + \frac{1}{n+1} + \left(\sum_{i=3}^{n-2} \frac{n-2}{(n+i-3)n-2} \right) + \frac{1}{2n+1} + \frac{1}{2n+4}
\end{aligned}
$$

Let

$$
\begin{aligned}
S_1 &= \left(\frac{2}{n} + \frac{1}{n+1} + \frac{1}{2n+1} + \frac{1}{2n+4} \right) - \left(\frac{1}{n-1} + \frac{1}{n} + \frac{1}{2n-3} + \frac{1}{2n} \right) \\
&= \frac{5n^3 + 13n^2 + 8n + 1}{n(n+1)(n+2)(2n+1)} - \frac{3n-4}{(n-1)(2n-3)} \\
&= \frac{4n^5 - 12n^4 - 27n^3 + 23n^2 + 27n + 3}{n(n+1)(n+2)(2n+1)(n-1)(2n-3)}
\end{aligned}
$$

For $3 \le i \le n-1$, let

$$
S_i = \left(\frac{1}{n+i-2} - \frac{n-2}{(n+i-3)n-2} \right) .
$$

S_i is a decreasing function with respect to *i*. When *i* = 3,

$$
S_3 = \frac{1}{n+1} - \frac{n-2}{n^2-2} = \frac{n}{(n+1)\left(n^2-2\right)} .
$$

Equation B-9 minus B-8 gives:

$$
\begin{aligned}
& f\left(\tfrac{n}{2}\right) - U(n-1) \\
&= S_1 - \sum_{i=3}^{n-2} S_i \\
&= \frac{4n^5 - 12n^4 - 27n^3 + 23n^2 + 27n + 3}{n(n+1)(n+2)(2n+1)(n-1)(2n-3)} - \sum_{i=3}^{n-2} S_i \\
&> \frac{4n^5 - 12n^4 - 27n^3 + 23n^2 + 27n + 3}{n(n+1)(n+2)(2n+1)(n-1)(2n-3)} - (n-4) \cdot \frac{n}{(n+1)\left(n^2-2\right)} \\
&= \frac{1}{n+1} \square \frac{4n^6 - 20n^5 - 18n^4 + 95n^3 - 19n^2 - 54n - 6}{n(n-1)(n+2)(2n+1)(2n-3)\left(n^2-2\right)} \\
&> 0
\end{aligned}
$$

- n is odd.

  Then $d_1 = \lceil \frac{n}{2} \rceil = \frac{n+1}{2}$. By equation B-1, $d_2 = n$ and $r = \frac{n-1}{2}$. Applying to equation B-7,

$$\begin{aligned} f\left(\lceil \tfrac{n}{2} \rceil\right) &= \frac{1}{\frac{n+1}{2}} + \frac{1}{n} + \left(\sum_{i=3}^{n-2} \frac{\frac{(n+1)}{2} - 1}{(n+i-3)\frac{(n+1)}{2} - 1} \right) + \frac{1}{2n-1} + \frac{1}{2n+2} \\ &= \frac{2}{n+1} + \frac{1}{n} + \frac{n-1}{(n+1)n-2} + \left(\sum_{i=4}^{n-2} \frac{n-1}{(n+i-3)(n+1)-2} \right) + \frac{1}{2n-1} + \frac{1}{2n+2} \qquad \text{(B-10)} \\ &= \frac{5}{2(n+1)} + \frac{1}{n} + \frac{1}{n+2} + \left(\sum_{i=4}^{n-2} \frac{n-1}{(n+i-3)(n+1)-2} \right) + \frac{1}{2n-1} \end{aligned}$$

Let

$$\begin{aligned} S_1 &= \left(\frac{3}{2(n+1)} + \frac{1}{n+2} + \frac{1}{2n-1} \right) - \left(\frac{1}{n-1} + \frac{1}{2n-3} + \frac{1}{2n} \right) \\ &= \frac{12n^2 + 17n - 4}{2(n+1)(n+2)(2n-1)} - \frac{8n^2 - 13n + 3}{2n(n-1)(2n-3)} \\ &= \frac{8n^5 - 40n^4 - 6n^3 + 85n^2 - 41n + 6}{2n(n+1)(n+2)(2n-1)(n-1)(2n-3)} \end{aligned}$$

For $i \geq 4$, let $S_i = \frac{1}{n+i-2} - \frac{n-1}{(n+i-3)(n+1)-2}$. S_i is a decreasing function with respect to *i*. When *i* = 4, $S_4 = \frac{1}{n+2} - \frac{n-1}{(n+1)(n+1)-2} = \frac{n+1}{(n+2)\left[(n+1)^2 - 2\right]}$.

Equation B-10 minus B-8 gives:

$$\begin{aligned} f\left(\lceil \tfrac{n}{2} \rceil\right) - U(n-1) &= S_1 - \sum_{i=4}^{n-2} S_i \\ &> S_1 - (n-5) S_4 \\ &= S_1 - \frac{(n-5)(n+1)}{(n+2)(n^2+2n-1)} \\ &= \frac{1}{n+2} \cdot \frac{24n^6 - 116n^5 + 9n^4 + 195n^3 - 105n^2 + 23n - 6}{2n(n+1)(n-1)(2n-1)(2n-3)(n^2+2n-1)} \\ &> 0 \end{aligned}$$

To summarize, it was first proved that $f(d_1)$ is a decreasing function with respect to d_1. So for any $d_1 \leq \frac{n}{2}$, $f(d_1) > f\left(\lceil \frac{n}{2} \rceil\right)$. On the other hand,

$U(d_1) > f(d_1)$ for any d_1. Therefore, $U(d_1) > f\left(\left\lceil \frac{n}{2} \right\rceil\right)$ for any $d_1 \le \frac{n}{2}$. It was next proved that $U(n-1) < f\left(\left\lceil \frac{n}{2} \right\rceil\right)$. Combining them together, the minimum of *U* must occur when $d_1 \ge \frac{n}{2}$.

2. $U(d_1+1) < U(d_1)$ for $\frac{n}{2} < d_1 \le n-2$.

Let $d_1^{'} = d_1 + 1$, and corresponding to *r*, d_{n-1} and d_n, r', d'_{n-1}, and d'_n are defined. Based on tables B-3 and B-4, the relationship between d_1, *r*, d_{n-1}, d_n, and r', d'_{n-1}, d'_n are shown in tables B-7 and B-8. The tables are built as follows. First, because the case $d_1 > \frac{n}{2}$ is being considered, by equation B-1, $d_2 = n$ and $r = n - d_1$. When d_1 is increased by 1, $d'_1 = d_1+1$, $d'_2 = d_2$, and $r' = r-1$. From tables B-3 and B-4, the expression of d_{n-1} and d_n only changes when *r* changes at some values. So only these special values of *r* and *r'* will be examined. Given *r* and d_1, depending on the parity of d_1, check either table B-3 or B-4 to find the corresponding d_{n-1} and d_n for *r*. If d_{n-1} and d_n are obtained from table B-3, then get d'_{n-1} and d'_n from table B-4, and if d_{n-1} and d_n are obtained from table B-4, then get d'_{n-1} and d'_n from table B-3. Look at entries of the table according to the relationship between r'_1 and d'_1 instead of *r* and d_1. $U(d_1)$ is expressed by equation B-6, and $U(d_1+1)$ can be expressed in the following equation:

$$U(d_1+1) = \frac{1}{d_1+1} + \frac{1}{d_2} + \left(\sum_{i=3}^{n-2} \frac{1}{n+i-2+\left\lfloor \frac{i+(n-d_1-1)-3}{d_1+1-1} \right\rfloor} \right) + \frac{1}{d_{n-1}^{'}} + \frac{1}{d_n^{'}} \qquad \text{(B-11)}$$

Let $H = \left(\frac{1}{d_{n-1}^{'}} + \frac{1}{d_n^{'}} \right) - \left(\frac{1}{d_{n-1}} + \frac{1}{d_n} \right)$.

Let $k_i = \left\lfloor \frac{i+n-d_1-3}{d_1-1} \right\rfloor$ and $k_i^{'} = \left\lfloor \frac{i+n-d_1-4}{d_1} \right\rfloor$.

Let $g(i) = \dfrac{1}{n+i-2+k_i^{'}} - \dfrac{1}{n+i-2+k_i} = \dfrac{1}{n+i-2+\left\lfloor \frac{i+n-d_1-4}{d_1} \right\rfloor} - \dfrac{1}{n+i-2+\left\lfloor \frac{i+n-d_1-3}{d_1-1} \right\rfloor}$.

a. $\frac{n}{2} < d_1 \le \frac{2n-6}{3}$. Then

$$k_i = \begin{cases} 0 & \text{if } 3 \le i < 2d_1 - n + 2 \\ 1 & \text{if } 2d_1 - n + 2 \le i < 3d_1 - n + 1 \\ 2 & \text{if } 3d_1 - n + 1 \le i \le n - 2 \end{cases}$$

$$k_i^{'} = \begin{cases} 0 & \text{if } 3 \le i < 2d_1 - n + 4 \\ 1 & \text{if } 2d_1 - n + 4 \le i < 3d_1 - n + 4 \\ 2 & \text{if } 3d_1 - n + 4 \le i \le n - 2 \end{cases}$$

$$g(i)=\begin{cases}\frac{1}{2d_1}-\frac{1}{2d_1+1} & \text{if } i=2d_1-n+2\\ \frac{1}{2d_1+1}-\frac{1}{2d_1+2} & \text{if } i=2d_1-n+3\\ \frac{1}{3d_1}-\frac{1}{3d_1+1} & \text{if } i=3d_1-n+1\\ \frac{1}{3d_1+1}-\frac{1}{3d_1+2} & \text{if } i=3d_1-n+2\\ \frac{1}{3d_1+2}-\frac{1}{3d_1+3} & \text{if } i=3d_1-n+3\\ 0 & \text{for all other } i\end{cases}$$

For $\frac{n}{2}<d_1\le\frac{2n-6}{3}$, $r=n-d_1>\frac{n+6}{3}>\frac{d_1+3}{2}$. From table B-8, $d_n=d_n^{'}$ and $d_{n-1}=d_{n-1}^{'}=2n-1$. Therefore, $H=0$.

Equation B-6 minus B-11 gives:

$$\begin{aligned}&U(d_1)-U(d_1-1)\\&=\frac{1}{d_1(d_1+1)}-\sum_{i=3}^{n-2}g(i)-H\\&=\frac{1}{d_1(d_1+1)}-\left(\frac{1}{2d_1}-\frac{1}{2d_1+2}+\frac{1}{3d_1}-\frac{1}{3d_1+3}\right)-0\\&>0\end{aligned}$$

TABLE B-7. THE CHANGES OF d_{n-1} AND d_n WHEN d_1 IS EVEN AND IS INCREASED BY 1

r	d_{n-1}	d_n	r'	d'_{n-1}	d'_n
2	$2n-1$	$2n-1$	1	$2n-3$	$2n$
$\frac{d_1+2}{2}$	$2n-1$	$2n+2$	$\frac{(d_1+1)-1}{2}$	$2n-1$	$2n-1$
$\frac{d_1+4}{2}$	$2n-1$	$2n+2$	$\frac{(d_1+1)+1}{2}$	$2n-2$	$2n+2$
others	$d_{n-1}=d_{n-1}^{'}$		$d_n=d_n^{'}$		

TABLE B-8. THE CHANGES OF d_{n-1} AND d_n WHEN d_1 IS ODD AND IS INCREASED BY 1

r	d_{n-1}	d_n	r'	d'_{n-1}	d'_n
2	$2n-2$	$2n-1$	1	$2n-3$	$2n$
$\frac{d_1+1}{2}$	$2n-1$	$2n+2$	$\frac{(d_1+1)-2}{2}$	$2n-1$	$2n-1$
$\frac{d_1+3}{2}$	$2n-1$	$2n+2$	$\frac{(d_1+1)}{2}$	$2n-1$	$2n-1$
others	$d_{n-1}=d'_{n-1}$		$d_n=d'_n$		

b. $\frac{2n-6}{3}<d_1\le\frac{2n-3}{3}$. Then

$$k_i=\begin{cases}0 & \text{if } 3\le i<2d_1-n+2\\ 1 & \text{if } 2d_1-n+2\le i<3d_1-n+1\\ 2 & \text{if } 3d_1-n+1\le i\le n-2\end{cases}$$

$$k'_i=\begin{cases}0 & \text{if } 3\le i<2d_1-n+4\\ 1 & \text{if } 2d_1-n+4\le i\le n-2\end{cases}$$

- $d_1=\frac{2n-3}{3}$.

$$g(i)=\begin{cases}\dfrac{1}{2d_1}-\dfrac{1}{2d_1+1} & \text{if } i=2d_1-n+2\\ \dfrac{1}{2d_1+1}-\dfrac{1}{2d_1+2} & \text{if } i=2d_1-n+3\\ \dfrac{1}{3d_1}-\dfrac{1}{3d_1+1} & \text{if } i=3d_1-n+1=n-2\\ 0 & \text{for all other } i\end{cases}$$

$r=\frac{n+3}{3}=\frac{d_1+3}{2}$, $d_{n-1}=2n-1$, $d_n=2n+2$ and $d'_{n-1}=d'_n=2n-1$. Therefore, $H=\frac{2}{2n-1}-\left(\frac{1}{2n-1}+\frac{1}{2n+2}\right)=\frac{3}{(2n-1)(2n+2)}$.

$$
\begin{aligned}
&U(d_1) - U(d_1 - 1) \\
&= \frac{1}{d_1\left(d_1+1\right)} - \sum_{i=3}^{n-2} g(i) - H \\
&= \frac{1}{d_1\left(d_1+1\right)} - \left(\frac{1}{2d_1} - \frac{1}{2d_1+2} + \frac{1}{3d_1} - \frac{1}{3d_1+1}\right) - H \\
&= \frac{1}{2d_1\left(d_1+1\right)} - \frac{1}{3d_1\left(3d_1+1\right)} - H \\
&= \frac{1}{2\frac{2n-3}{3}\left(\frac{2n-3}{3}+1\right)} - \frac{1}{(2n-3)(2n-2)} - \frac{3}{(2n-1)(2n+2)} \\
&= \frac{1}{2}\cdot\left(\frac{7n-9}{2n(2n-3)(n-1)} - \frac{3}{(2n-1)(n+1)}\right) \\
&= \frac{2n^3+19n^2-34n+9}{4n(2n-3)(n-1)(2n-1)(n+1)} \\
&> 0
\end{aligned}
$$

- $d_1 = \frac{2n-4}{3}$. Then

$$
g(i) = \begin{cases}
\frac{1}{2d_1} - \frac{1}{2d_1+1} & \text{if } i = 2d_1 - n + 2 \\
\frac{1}{2d_1+1} - \frac{1}{2d_1+2} & \text{if } i = 2d_1 - n + 3 \\
\frac{1}{3d_1} - \frac{1}{3d_1+1} & \text{if } i = 3d_1 - n + 1 = n - 3 \\
\frac{1}{3d_1+1} - \frac{1}{3d_1+2} & \text{if } i = 3d_1 - n + 2 = n - 2 \\
0 & \text{for all other } i
\end{cases}
$$

$r = \frac{n+4}{3} = \frac{d_1+4}{2}$. Note that $2n$-4 is an even number and d_1 is an integer. So, $d_1 = \frac{2n-4}{3}$ must also be even. Checking the corresponding entries in table B-7, $d_n = d_n^{'} = 2n+2$, $d_{n-1} = 2n-1$, and $d_{n-1}^{'} = 2n-2$.

$H = \left(\frac{1}{2n-2} + \frac{1}{2n+2}\right) - \left(\frac{1}{2n-1} + \frac{1}{2n+2}\right) = \frac{1}{(2n-1)(2n-2)}$.

$$\begin{aligned}
&U(d_1) - U(d_1 - 1) \\
&= \frac{1}{d_1(d_1+1)} - \sum_{i=3}^{n-2} g(i) - H \\
&= \frac{1}{d_1(d_1+1)} - \left(\frac{1}{2d_1} - \frac{1}{2d_1+2} + \frac{1}{3d_1} - \frac{1}{3d_1+2}\right) - H \\
&= \frac{1}{2d_1(d_1+1)} - \frac{2}{3d_1(3d_1+2)} - H \\
&= \frac{1}{2\frac{2n-4}{3}\left(\frac{2n-4}{3}+1\right)} - \frac{2}{(2n-4)(2n-2)} - \frac{1}{(2n-1)(2n-2)} \\
&= \frac{9}{2(2n-4)(2n-1)} - \frac{1}{(2n-4)(n-1)} - \frac{1}{(2n-1)(n-2)} \\
&= \frac{1}{2}\cdot\left[\frac{5n-7}{2(n-2)(2n-1)(n-1)} - \frac{1}{(2n-1)(n-2)}\right] \\
&= \frac{1}{2}\cdot\frac{3}{2(2n-1)(n-2)} \\
&> 0
\end{aligned}$$

- $d_1 = \frac{2n-5}{3}$. Then

$$g(i) = \begin{cases} \frac{1}{2d_1} - \frac{1}{2d_1+1} & \text{if } i = 2d_1 - n + 2 \\ \frac{1}{2d_1+1} - \frac{1}{2d_1+2} & \text{if } i = 2d_1 - n + 3 \\ \frac{1}{3d_1} - \frac{1}{3d_1+1} & \text{if } i = 3d_1 - n + 1 = n - 4 \\ \frac{1}{3d_1+1} - \frac{1}{3d_1+2} & \text{if } i = 3d_1 - n + 2 = n - 3 \\ \frac{1}{3d_1+2} - \frac{1}{3d_1+3} & \text{if } i = 3d_1 - n + 3 = n - 2 \\ 0 & \text{for all other } i \end{cases}$$

$r = \frac{n+5}{3} > \frac{d_1+3}{2}$ and $r' = \frac{n+2}{3} = \frac{(d_1+1)+2}{2}$. Note that $2n$-5 is an odd number and d_1 is an integer. So, $d_1 = \frac{2n-5}{3}$ must also be odd. Checking the corresponding entries of table B-8, $d_n = d_n' = 2n+2$ and $d_{n-1} = d_{n-1}' = 2n-1$. Therefore, $H = 0$.

$$
\begin{aligned}
&U(d_1)-U(d_1-1)\\
&=\frac{1}{d_1\left(d_1+1\right)}-\sum_{i=3}^{n-2}g(i)-H\\
&=\frac{1}{d_1\left(d_1+1\right)}-\left(\frac{1}{2d_1}-\frac{1}{2d_1+2}+\frac{1}{3d_1}-\frac{1}{3d_1+3}\right)-0\\
&=\frac{1}{6d_1\left(d_1+1\right)}\\
&>0
\end{aligned}
$$

c. $\frac{2n-3}{3}<d_1\le n-2$.

$$
k_i=\begin{cases}0 & \text{if } 3\le i<2d_1-n+2\\ 1 & \text{if } 2d_1-n+2\le i\le n-2\end{cases}
$$

$$
k_i^{'}=\begin{cases}0 & \text{if } 3\le i<2d_1-n+4\\ 1 & \text{if } 2d_1-n+4\le i\le n-2\end{cases}
$$

$$
g(i)=\begin{cases}\dfrac{1}{2d_1}-\dfrac{1}{2d_1+1} & \text{if } i=2d_1-n+2\\[2ex] \dfrac{1}{2d_1+1}-\dfrac{1}{2d_1+2} & \text{if } i=2d_1-n+3\\[2ex] 0 & \text{for all other } i\end{cases}
$$

Equation B-6 minus B-11 gives:

$$
\begin{aligned}
&U(d_1)-U(d_1+1)\\
&=\frac{1}{d_1\left(d_1+1\right)}-\sum_{i=3}^{n-2}g(i)-H\\
&=\frac{1}{d_1\left(d_1+1\right)}-\left(\frac{1}{2d_1}-\frac{1}{2d_1+2}\right)-H\\
&=\frac{1}{2d_1\left(d_1+1\right)}-H
\end{aligned}
\tag{B-12}
$$

Depending on d_1, H takes on different values. There are four cases:

- $d_1=\frac{2n-2}{3}$. Then $r=\frac{d_1+2}{2}$. Because $2n$-2 is an even number and d_1 is an integer, $d_1=\frac{2n-2}{3}$ must be even. From table B-7, $d_n^{'}=d_{n-1}^{'}=2n-1$,

$d_{n-1} = 2n-1$, and $d_n = 2n+2$. Hence, $H = \frac{1}{2n-1} - \frac{1}{2n-2}$. Applying to equation B-12 gives:

$$
\begin{aligned}
& U(d_1) - U(d_1+1) \\
&= \frac{1}{2\,\frac{2n-2}{3}\left(\frac{2n-2}{3}+1\right)} - \left(\frac{1}{2n-1} - \frac{1}{2n-2}\right) \\
&= \frac{6n^2+15n-3}{4(n+1)(n-1)(2n-1)(2n+1)} \\
&> 0
\end{aligned}
$$

- $d_1 = \frac{2n-1}{3}$. Then $r = \frac{d_1+1}{2}$. Because 2*n*-1 is an odd number and d_1 is an integer, $d_1 = \frac{2n-1}{3}$ must be even. From table B-8, $d_n^{'} = d_{n-1}^{'} = 2n-1$, $d_{n-1} = 2n-2$, and $d_n = 2n+2$. Hence, $H = \frac{2}{2n-1} - \left(\frac{1}{2n-2} + \frac{1}{2n+2}\right)$. Applying to equation B-12 gives:

$$
\begin{aligned}
& U(d_1) - U(d_1+1) \\
&= \frac{1}{2\,\frac{2n-1}{3}\left(\frac{2n-1}{3}+1\right)} - \left[\frac{2}{2n-1} - \left(\frac{1}{2n-2} + \frac{1}{2n+2}\right)\right] \\
&= \frac{5n-1}{4(2n-1)(n+1)(n-1)} \\
&> 0
\end{aligned}
$$

- $\frac{2n}{3} < d_1 < n-2$. Then $d_n = d_n^{'}$ and $d_{n-1} = d_{n-1}^{'}$. So $H = 0$.

$$
U(d_1) - U(d_1+1) = \frac{1}{2d_1(d_1+1)} - 0 > 0
$$

- $d_1 = n-2$. Then $r = 2$. From tables B-7 and B-8, $d_n = d_{n-1} = 2n-1$, $d_{n-1}^{'} = 2n-3$, and $d_n^{'} = 2n$. Therefore, $H = \left(\frac{1}{2n-3} + \frac{1}{2n}\right) - \frac{2}{2n-1}$. Applying to equation B-12 gives:

$$
\begin{aligned}
& U(d_1) - U(d_1+1) \\
&= \frac{1}{2(n-2)(n-2+1)} - \left[\left(\frac{1}{2n-3} + \frac{1}{2n}\right) - \frac{2}{2n-1}\right] \\
&= \frac{2n^3-5n^2+8n-6}{2n(n-2)(n-1)(2n-1)(2n-3)} \\
&> 0
\end{aligned}
$$

Claim 2: Given the number of tasks n and $2 \le d_1 \le n-2$, the Perfect Instance for n and d_1 constructed above yields the minimum utilization of n tasks that perfectly utilize the processor among all those task sets that have the same d_1 and d_2 as the Perfect Instance.

Proof: For $n \le 9$, the claim can be proved by enumeration. So, assume that $n \ge 10$. Then $d_2 \ge n \ge 10$. Let the perfect instance for d_1 and n be $TS = \{T_1(1,d_1), T_2(1,d_2), ..., T_n(1,d_n)\}$. It will be proved by contradiction that *TS* has the minimum utilization among all task set instances that have the same d_1 and d_2 as *TS*. Suppose there is another task set $TS' = \{T_1'(1,d_1'), T_2'(1,d_2'), ..., T_n'(1,d_n')\}$ that perfectly utilizes the processor and $d_1' = d_1$ and $d_2' = d_2$, but has a smaller utilization. Then there must be some task T_j, $3 \le j \le n$, such that $d_j < d_j'$. Let T_i be such a task that has the minimum deadline. It will be shown that $i \notin \{3,...,n-2\}$, $i \ne n-1$, and $i \ne n$ case by case.

1. $i \notin \{3,...,n-2\}$.

   Proof. Suppose on the contrary, there is an i, $3 \le i \le n-2$, such that $d'_i > d_i$. Using time demand analysis, it can be proved that: $d'_i = d'_{i+1} = ... = d'_n = d_i + 1 < 2d_2 = 2d'_2$.

$$U' - U \ge \sum_{j=i}^{n} \frac{1}{d'_j} - \sum_{j=i}^{n} \frac{1}{d_j} = \frac{n-i+1}{d_i+1} - \sum_{j=i}^{n} \frac{1}{d_j}$$
$$> \frac{n-i+1}{d_i+1} - (\sum_{j=i}^{n-2} \frac{1}{d_i+j-i} + \frac{1}{d_{n-1}} + \frac{1}{d_n})$$

   There are two cases:

   - $i = n-2$.

     Then, $U' - U = \frac{3}{d_{n-2}+1} - (\frac{1}{d_{n-2}} + \frac{1}{d_{n-1}} + \frac{1}{d_n})$. By construction of d_i,

     – Either $d_{n-1} = d_n \ge d_{n-2} + 2$.

$$U' - U > \frac{3}{d_{n-2}+1} - (\frac{1}{d_{n-2}} + \frac{1}{d_{n-2}+2} + \frac{1}{d_{n-2}+2}) > 0$$

     – Or $d_{n-1} \ge d_{n-2} + 1$ and $d_n \ge d_{n-1} + 3$.

$$U' - U > \frac{3}{d_{n-2}+1} - (\frac{1}{d_{n-2}} + \frac{1}{d_{n-2}+1} + \frac{1}{d_{n-2}+4}) > 0$$

- For all other i,

$$U' - U = \frac{n-i+1}{d_i+1} - \sum_{j=i}^{n} \frac{1}{d_j} > \frac{n-i+1}{d_i+1} - \sum_{j=i}^{n} \frac{1}{d_i+j-i} > 0 .$$

2. $i \neq n-1$.

Proof. Suppose on the contrary $i = n-1$. By construction, either $d_{n-1} = t_1$ or $d_{n-1} = t_1 + 1$. It will be proved that under both cases, $U' - U > 0$, where U' and U are the total utilization of *TS'* and *TS*, respectively.

- $d_{n-1} = t_1$.

  For $2 \leq j \leq n - 2$, $d_j < t_1 < 2d_2 \leq 2d'_j$. So T'_j makes exactly two requests before time t_1. The total number of requests from T'_1, T'_2..., T'_{n-1} before t_1 is $R = \left\lceil \frac{t_1}{d_1} \right\rceil + 2(n-3) + 1$. By equation B-2, $R = t_1$. None of T'_1, T'_2,…,T'_{i-1}, T'_{i+1}, …, T'_{n-2} will make their third requests at time t_1. Because $d'_{n-1} > d_{n-1} = t_1$, T'_{n-1} will not make a request at time t_1. Thus, the first request of T'_n will be executed. This means that $d'_n = d'_{n-1} = d_{n-1} + 1 \leq 2d_2$.

  For $3 \leq j \leq n-2$, by assumption, $d'_j \leq d_j$.

$$U' - U \geq \frac{1}{d'_{n-1}} + \frac{1}{d'_n} - (\frac{1}{d_{n-1}} + \frac{1}{d_n}) = \frac{2}{d_{n-1}+1} - (\frac{1}{d_{n-1}} + \frac{1}{d_n})$$

  By tables B-3 and B-4, one can find that for each pair of d_{n-1} and d_n such that $d_{n-1} = t_1$, the right-hand side of the above inequality is positive, i.e., $U' - U > 0$.

- $d_{n-1} = d_n = t_1 + 1$.

  It will be shown that in this case it is impossible that $d'_{n-1} > d_{n-1}$. By tables B-3 and B-4, $t_1 = 2d_2 - 2 < 2d'_j$, where $2 \leq j \leq n-2$. So at time t_1, task T'_j makes exactly two requests. By equation B-2, all requests from T'_1, …, T'_{n-1} will be executed before t_1, and no request is released at t_1 by task T'_j, where $2 \leq j \leq n-2$, because $t_1 < 2d'_j$. It has been proved that t_1 is not a multiple of $d'_1 = d_1$. Thus, T'_1 will not make a request at t_1. If $d'_{n-1} > d_{n-1}$, T'_{n-1} will not make the second request at t_1. In order to be a task set perfectly utilizing the processor, the first request of T'_n must be executed at t_1. Therefore, $d'_n = d'_{n-1} = d_{n-1}$, which is a contradiction to the assumption that $d'_{n-1} > d_{n-1}$.

3. $i \neq n$.

Suppose $i = n$, i.e.,T_n is the only task such that $d_i < d'_i$. There are two cases to consider depending on the relationship between d_n and d_2.

- $d_n \leq 2d_2$.

  By assumption, $d'_2 = d_2$ and $d'_2 \leq d_j$ for $j \geq 3$. So $d_n \leq 2d'_j$ for $j \geq 2$. On the other hand, $d'_j < d_j < d_n$. So, for all $2 \leq j \leq n-1$, task T'_j makes exactly two requests before time $t = d_n - 1$, and it will not make the third request before time d_n. It has been shown that all those requests by $T'_1, T'_2, ..., T'_{n-1}$ can be executed before time $d_n - 1$, and T'_1 will not make a request at $d_n - 1$. So the first request of T'_n will be executed at time $d_n - 1$. However, this means that $d'_n = d_n$, which contradicts the assumption that $d_n < d'_n$.

- $d_n = 2d_2 + 2$.

  It is known that $d'_3 \geq d'_2 = d_2$. To prove the claim, there are two cases to consider, depending on whether $d'_3 > d_2$ or $d'_3 = d_2$. If $d'_3 > d_2$, then T'_3 will not make its third request before $d_n = 2d_2 + 2$. So, using the same argument for the case $d_n \leq 2d_2$, it can be shown that $d'_n = d_n$, which contradicts the assumption that $d_n < d'_n$.

  Consider now the case $d'_3 = d_2$. The proof is quite involved. The idea is to show that the task set TS' can be obtained from TS by a series of modifications of d_i and d_n for $i \geq 3$. Each step can be described as follows: Starting from i = 3, if $d'_i < d_i$, then decrease d_i to d'_i and if necessary, increase d_n to make the modified task set to continue perfectly utilizing the processor. Let the modified d_n be $d_n(i)$, the new task set be TS_i, and U_i be the total utilization of TS_i. Let S denote the maximum index i such that $d'_i < d_i$. By assumption, $S \leq n-1$. After S steps, the instance TS' is obtained. Let U denote the total utilization of the perfect instance. In the following, it will be proved that $U < U'$, which is a contradiction. For convenience let $U_2 = U$ and $d_n(2) = 2d_2 + 2$.

  The proof is by induction on i - the index of the task besides T_n that is changed at each step.

Induction hypotheses: There are three hypotheses:

1. For $3 \le i \le S-1$, either $d_n(i) = d_n(i-1)+1$, or $d_n(i) = d_n(i-1)+2 ; 2d'_i < d_n(i-1) \le 2d_i$.

2. For $3 \le i \le S-1$, $d_n(i) \le 3d_2$, and

$$d_n(i) = 2d_2 + i + \left\lceil \frac{i+2r-d_1-2}{d_1-1} \right\rceil = 2d_2 + i + \left\lceil \frac{i-(2d_1-2r)}{d_1-1} \right\rceil \quad \text{(B-13)}$$

3. $U < U_i$ for $3 \le i \le S$.

Base case: $i = 3$.

1. $d_n(3) = d_n(2)+1$ or $d_n(3) = d_n(2)+2$; $2d'_3 < d_n(2) \le 2d_3$.

By assumption, $2d'_3 = 2d_2 < d_n(2) = 2d_2 + 2 \le 2d_3$. To prove the remainders, there are two cases depending on *r*.

a. $r = d_1 - 1$.

Then $d_3 = d_2 + 2$. The number of requests of T_3 before time $d_n(2) = d_n = 2d_2 + 2$ is 2. When d_3 is changed to d'_3, which is equal to d_2, the number of requests by T'_3 in the new task instance TS_3 before time d_n is 3, i.e., one more than that of T_3 in the original instance. In order to make the new instance perfectly utilize the processor, d_n needs to be increased by at least 1. However, because $d_n = 2d_2 + 2$ is a multiple of d_1, at time d_n, T_1 will make another request. Therefore, d_n needs to be increased by at least 2. In fact, d_n needs to be increased by exactly 2. Because the period of any other task T_j has not been changed, $4 \le j \le n$ - 1, $2d_j > d_n + 2 = 2d_2 + 4$. So, it makes exactly two requests before time $d_n + 2$, which is the same as in the original task set. The requests of task T_2 before $2d_2 + 4$ is still the same -- 3. Therefore, $d_n(3) = d_n$.

b. $r < d_1 - 1$.

Then $d_3 = d_2 + 1$. The number of requests of T_3 before $d_n(2) = d_n = 2d_2 + 2$ is 2. When d_3 is changed to d'_3, which is equal to d_2, the number of requests by T'_3 in the new task instance TS_3 before d_n

is 1 more than that of T_3 in the original instance. In order to make the new instance perfectly utilizes the processor, d_n needs to be increased by at least 1. It is enough that d_n is increased by exactly 1. Because $d_n = 2d_2 + 2$ is not a multiple of d_1, T_1 will not make any request at time d_n. The period of any other tasks T_j, $4 \le j \le n-1$, is not changed. So T'_j makes exactly two requests before time $d_n + 2$, which is the same as in the original task set. Therefore, $d_n(3) = d_n + 1 = 2d_2 + 3$.

2. $d_n(3) \le 3d_2$ and $d_n(3) = 2d_2 + 3 + \left\lceil \frac{3+2r-d_1-2}{d_1-1} \right\rceil = 2d_2 + 3 + \left\lceil \frac{3-(2d_1-2r)}{d_1-1} \right\rceil$.

   It has already been shown that if $r = d_1 - 1$, then $d_n(3) = 2d_2 + 4$; otherwise, $d_n(3) = 2d_2 + 3$. It is obvious that $d_n(3) \le 3d_2$. In both cases, $d_n(3)$ can be written in the following way:

$$d_n(3) = 2d_2 + 3 + \left\lceil \frac{3+2r-d_1-2}{d_1-1} \right\rceil = 2d_2 + 3 + \left\lceil \frac{3-(2d_1-2r)}{d_1-1} \right\rceil$$

3. $U < U_3$.

   Since only the deadlines of T_3 and T_n are modified, the difference between the total utilization of the new instance U_3 and that of the original instance U is

$$U_3 - U = \frac{1}{d_3} + \frac{1}{d_n(3)} - (\frac{1}{d_3} + \frac{1}{d_n}) = \frac{1}{d_2} + \frac{1}{d_n(3)} - (\frac{1}{d_3} + \frac{1}{2d_2+2}).$$

   It has been shown that either $d_3 = d_2 + 2$ and $d_n(3) = 2d_2 + 4$, or $d_3 = d_2 + 1$ and $d_n(3) = 2d_2 + 3$. It is easy to show that in both cases, $U_3 - U > 0$. Therefore, $U < U_3$.

Induction step: Suppose the hypotheses are true for 3, 4, …, *i*-2, where $i \le$ S. Now there is the task set TS_{i-1}. One needs to change d_i to d'*ᵢ*, and change $d_n(i-1)$ to $d_n(i)$ if necessary. The hypotheses need to be proved to be true for i.

First, it will be proved that $2d_i < d_n(i-1)$. Suppose $2d_i \ge d_n(i-1)$. It will be shown that there is a contradiction to the assumption that *TS′* has a smaller utilization than the perfect instance. When d_i is decreased to d'_i, the number of requests of T'_i before time $d_n(i-1)$ has not changed, i.e., it makes two requests.

So there is no need to increase $d_n(i-1)$, which means that $d_n(i) = d_n(i-1)$. Therefore, $U_{i-1} < U_i$. By induction, $U < U_{i-1}$. Thus, $U < U_i$. Furthermore, for all $i < j \le n-1$, $d'_i \le d'_j$. Therefore, $2d'_j \ge d_n(i)$. So the decrease of d_j to d'_j will not make any change of $d_n(j)$, i.e., $d_n(j) = d_n(i)$. However, this can only make the total utilization increased. Therefore, it must be true that $d_j = d'_j$. This means that $TS' = TS_i$ and $U < U_i = U'$, which is a contradiction.

Now the remainder part of the hypotheses will be proved. There are two cases to consider, $i \le n-2$ and $i = n-1$.

1. $i \le n-2$.

   a. $d_n(i) = d_n(i-1)+1$ or $d_n(i) = d_n(i-1)+2$; and $2d'_i < d_n(i-1) \le 2d_i$.

   It has already been proved that $2d'_i < d_n(i-1)$. By induction, $d_n(i-1) \le 3d_2 \le 3d'_i$. So task T'_i makes exactly three requests before time $d_n(i-1)$, which is one more request than that of T_i in the task set TS_{i-1}. So $d_n(i-1)$ must be increased by at least 1. If $d_n(i-1)$ is a multiple of d_1, $d_n(i-1)$ must be increased by at least 2. On the other hand, it can be shown that $d_n(i-1)+1 \le 3d_2 = 3d'_2 \le 3d'_j$ for $2 \le j \le i-1$. Task T_j will not make its fourth request at $d_n(i-1)$ or $d_n(i-1)+1$. Therefore, $d_n(i-1)$ only need to be increased by exactly 1, or by 2 if $d_n(i-1)$ is a multiple of d_1. Thus, $d_n(i) = d_n(i-1)+1$ or $d_n(i) = d_n(i-1)+2$.

   It has been proved that $2d'_i < d_n(i-1)$. By induction, $d_n(i-1)$ satisfies equation B-13 for $i-1$. Together with equation B-1, it can be shown that $d_n(i-1) \le 2d_i$.

   b. $d_n(i) \le 3d_2$ and $d_n(i) = 2d_2 + i + \left\lceil \frac{i+2r-d_1-2}{d_1-1} \right\rceil = 2d_2 + i + \left\lceil \frac{i-(2d_1-2r)}{d_1-1} \right\rceil$.

   It is known that either $d_n(i) = d_n(i-1)+1$ or $d_n(i) = d_n(i-1)+2$, depending on whether $d_n(i-1)$ is a multiple of d_1. By induction, $d_n(i-1)$ satisfies equation B-13 for $i-1$. Therefore,

$$d_n(i) = 2d_2 + i + \left\lceil \frac{i+2r-d_1-2}{d_1-1} \right\rceil = 2d_2 + i + \left\lceil \frac{i-(2d_1-2r)}{d_1-1} \right\rceil$$

   From the equation, one can show that $d_n(i) \le 3d_2$.

c. $U < U_i$.

It has already been proved that $2d'_i < d_n(i-1) < 2d_i$. So $d'_i \le (d_n(i-1)-1)/2$ and $d_i \ge d_n(i-1)/2$. It has already been proved that $d_n(i) \le d_n(i-1)+2$. Therefore,

$$\begin{aligned}
U_{i-1} - U_i &= (\frac{1}{d_i} + \frac{1}{d_n(i)}) - (\frac{1}{d_i} + \frac{1}{d_n(i-1)}) \\
&> (\frac{2}{d_n(i-1)-1} + \frac{1}{d_n(i-1)+2}) - (\frac{2}{d_n(i-1)} + \frac{1}{d_n(i-1)}) \\
&= \frac{3d_n(i-1)+3}{(d_n(i-1)-1)(d_n(i-1)+2)} - \frac{3}{d_n(i-1)} \\
&= \frac{6}{d_n(i-1)\cdot(d_n(i-1)-1)(d_n(i-1)+2)} \\
&\ge 0
\end{aligned}$$

2. $i = n-1$.

It is known that $S \le n-1$ and $i \le S$. So $i = S$. What remains to be proved for this case is $U < U_{n-1}$.

Like the previous case, it can be proved that $2d'_{n-1} < d_n(n-2) \le 2d_{n-1}$. To prove the remaining hypotheses, there are two cases to consider:

a. $d_n(n-2) < 3d_2$.

If d_{n-1} is changed to d'_{n-1}, the number of requests by T'_{n-1} increases by 1 before time $d_n(n-2)$. $d_n(n-2)$ must be increased by at least 1. If $d_n(n-2)+1$ is a multiple of d_1, then $d_n(n-2)$ needs to be increased by at least 2. It will be shown that one needs to increase by exactly 1 or 2.

It is necessary to ensure that other tasks T'_j, $2 \le j \le n-2$, does not make an extra request after $d_n(n-2)$ is increased by 1 or 2. If $d_n(n-2) \le 3d_2 - 2$, then $d_n(n-2)+2 \le 3d_2$ and T'_j will not make its fourth request before $3d_2$. Thus, $d_n(n-1) \le 3d_2$. As before, it can be shown that $U_{n-2} < U_{n-1}$ by simple calculation. Therefore, $U < U_{n-1}$. Otherwise, $d_n(n-2) = 3d_2 - 1$. In this case, T_1 will not make a request at time $3d_2 - 1$, since $\frac{d_1+1}{2} \le r$ (which is implied by the assumption that $d_n = 2d_2 + 2$ and table B-3). Similarly, T'_j will not make its fourth request before $3d_2$. Thus, $d_n(n-1) = 3d_2$ and one can show that $U < U_{n-1}$.

b. $d_n(n-2) = 3d_2$.

In this case, if d_{n-1} is changed to d'_{n-1}, it is not true that one only needs to increase $d_n(n-2)$ by 1 or 2 as before, because after $d_n(n-2)$ is increased by 1 or 2, T'_2, and T'_3 will make their fourth requests. It may happen that while these requests are being executed, T'_4, …also make requests during this period. So there is no chance to execute the first request of T'_n before finishing other requests with higher priority.

In order to prove the hypotheses, one cannot do the simple calculation this time. One needs to compare U and U_{n-1} term by term. d'$_i$ can be estimated based on the induction.

$$\begin{aligned}
d'_i &\le \left\lceil \frac{d_n(i-1)-1}{2} \right\rceil \\
&= \left\lceil \frac{2d_2+(i-1)+\left\lceil \frac{i-1-(2d_1-2r)}{d_1-1} \right\rceil -1}{2} \right\rceil \\
&\le d_2 + \left\lceil \frac{i-2+\left\lceil \frac{i-1-(2d_1-2r)}{d_1-1} \right\rceil}{2} \right\rceil \\
&\le d_2 + \frac{i-2+\left\lceil \frac{i-(2d_1-2r)}{d_1-1} \right\rceil}{2}
\end{aligned}$$

$$\begin{aligned}
d_i &= d_2+i-2+\left\lceil \frac{i+r-3}{d_1-1} \right\rceil = d_2+i-2+\left\lceil \frac{i+r-2-(d_1-1)}{d_1-1} \right\rceil \\
&= d_2+i-2+\left\lceil \frac{i-(d_1-r+1)}{d_1-1} \right\rceil \\
&\le \frac{(n+i-3)d_1-1}{d_1-1}
\end{aligned}$$

$$\left\lceil \frac{i-(d_1-r+1)}{d_1-1} \right\rceil -1 \le \left\lceil \frac{i-(2d_1-2r)}{d_1-1} \right\rceil \le \left\lceil \frac{i-(d_1-r+1)}{d_1-1} \right\rceil$$

$$\begin{aligned}
d'_i &\le d_i - \frac{i-2+\left\lceil \frac{i-(d_1-r+1)}{d_1-1} \right\rceil}{2} \\
&\le d_i - \frac{i-2+\frac{i-(d_1-r+1)}{d_1-1}}{2} \\
&\le \frac{(2n+i-3)d_1-r-3}{2(d_1-1)}
\end{aligned}$$

$$
\begin{aligned}
\frac{1}{d'_i}-\frac{1}{d_i}&=\frac{d_i-d'_i}{d_i\,d'_i}\\
&\geq\frac{\frac{i-2+\frac{i-(d_1-r+1)}{d_1-1}}{2}}{d_i\,d'_i}\\
&\geq\frac{\frac{(i-2)(d_1-1)+(i-d_1+r-1)}{2(d_1-1)}}{\frac{(n+i-3)d_1-1}{d_1-1}\,\frac{(2n+i-3)d_1-r-3}{2(d_1-1)}}\\
&\geq(d_1-1)\frac{(i-3)d_1+r+3}{[(n+i-3)d_1-1][(2n+i-3)d_1-r-3]}\\
&\geq(d_1-1)\frac{(i-3)d_1}{[(n+i-3)d_1][(2n+i-3)d_1]}\\
&\geq\frac{d_1-1}{d_1}\cdot\frac{i-3}{(n+i-3)(2n+i-3)}
\end{aligned}
$$

$$
\begin{aligned}
U_{n-1}-U&=\sum_{i=1}^{i=n}(\frac{1}{d'_i}-\frac{1}{d_i})=\sum_{i=3}^{n}(\frac{1}{d'_i}-\frac{1}{d_i})\\
&>\sum_{i=3}^{n-2}(\frac{1}{d'_i}-\frac{1}{d_i})+(\frac{1}{d'_{n-1}}-\frac{1}{d_{n-1}})+(0-\frac{1}{d_n})\\
&>\sum_{i=3}^{n-2}(\frac{1}{d'_i}-\frac{1}{d_i})+(\frac{2}{3d_2}-\frac{1}{2d_2-2})+(0-\frac{1}{2d_2+2})\\
&=\sum_{i=3}^{n-2}(\frac{1}{d'_i}-\frac{1}{d_i})-\frac{d_2^2+2}{3d_2(d_2^2-1)}\\
&>(\sum_{i=3}^{n-2}\frac{1}{d'_i}-\frac{1}{d_i})-\frac{1}{3d_2}+\frac{1}{d_2(d_2^2-1)}\\
&>\sum_{i=3}^{n-2}(\frac{1}{d'_i}-\frac{1}{d_i})-\frac{1}{3d_2}+\frac{1}{d_2(n^2-1)}\\
&>(\sum_{i=3}^{n-2}\frac{1}{d'_i}-\frac{1}{d_i})-(\frac{d_1-1}{3[(n-1)d_1-r]}+\frac{d_1-1}{[(n-1)d_1-r](n^2-1)})\\
&>(\sum_{i=3}^{n-2}\frac{1}{d'_i}-\frac{1}{d_i})-[\frac{d_1-1}{(3n-6)d_1}+\frac{d_1-1}{(n-2)(n^2-1)d_1}]\\
&>\frac{d_1-1}{d_1}[(\sum_{i=3}^{n-2}\frac{i-3}{(n+i-3)(2n+i-3)})-(\frac{1}{(3n-6)}+\frac{1}{(n-2)(n^2-1)})]\\
&>0\quad\text{when } n\geq 10
\end{aligned}
$$

It has just been proved that $U<U_S$. Note that $TS_S=TS'$, so $U<U'$.

B.5.2 PERFECT INSTANCE WHEN d_1 AND d_2 ARE ARBITRARY.

In the following, an instance of n tasks for given d_1 and d_2 will be constructed. For convenience, the constructed instance will be called the perfect instance for n, d_1, and d_2. It will first be shown that the instance perfectly utilizes the processor. It is then proved that the instance constructed for $d_1 = n - 1$ has the minimum total utilization among all task instances that have n tasks and perfectly utilize the processor.

Construction of Perfect Instances.

Given the number of tasks n, $T_1(1, d_1)$ and $T_2(1, d_2)$, where $d_1 \le n-1$, $d_2 \le n$ and $d_1 \le d_2$, construct $T_3(1, d_3), \ldots, T_n(1, d_n)$ as follows:

1. For $3 \le i \le n - 2$, let d_i be the maximum integer such that

$$\begin{cases} d_i = \left\lceil \frac{d_i}{d_1} \right\rceil + \left\lceil \frac{d_i}{d_2} \right\rceil + (n+i-6) \\ t \le \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + (n+i-6) \text{ for } t \le d_i \end{cases} \tag{B-14}$$

2. In order to define d_{n-1} and d_n, define the first two variables t_1 and t_2

- Let t_1 be the maximum integer such that

$$\begin{cases} t_1 = \left\lceil \frac{t_1}{d_1} \right\rceil + \left\lceil \frac{t_1}{d_2} \right\rceil + (2n-7) \\ t \le \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + (2n-7) \text{ for } t \le t_1 \end{cases} \tag{B-15}$$

- Let t_2 be the minimum integer such that

$$\begin{cases} t_2 = \left\lceil \frac{t_2}{d_1} \right\rceil + \left\lceil \frac{t_2}{d_2} \right\rceil + (2n-5) \\ t < \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + (2n-5) \text{ for } t < t_2 \end{cases} \tag{B-16}$$

If $t_2 = kd_3 + 1$ for some k, then $d_n = t_2 + 1$ and $d_{n-1} = t_1$. Otherwise, $t_2 \neq kd_3 + 1$ for any k. If $t_2 = t_1 + 2$, let $d_{n-1} = d_n = t_1 + 1 = t_2 - 1$; else $t_2 \ge t_1 + 3$, let $d_{n-1} = t_1$ and $d_n = t_2$. It will first be proved that the constructed perfect instance has several properties. It will then be proved that this instance perfectly utilizes the processor.

Claim 3: For $3 \le i \le n - 1$, d_i is neither a multiple of d_1 nor a multiple of d_2.

Proof. Consider first the case $3 \le i \le n-2$. Let $d_i = k_{i_1}d_1 + r_{i_1} = k_{i_2}d_2 + r_{i_2}$, where k_{i_1} and k_{i_2} are integers, $0 \le r_{i_1} \le d_1 - 1$ and $0 \le r_{i_2} \le d_2 - 1$. It will be proved that both r_{i_1} and r_{i_2} are greater than 0; if not, then there are two cases to consider.

a. Either $r_{i_1} = 0$ or $r_{i_2} = 0$. Suppose that $r_{i_1} = 0$, then $\left\lceil \frac{d_i+1}{d_1} \right\rceil = \left\lceil \frac{d_i}{d_1} \right\rceil + 1$ and $\left\lceil \frac{d_i+1}{d_2} \right\rceil = \left\lceil \frac{d_i}{d_2} \right\rceil$. Therefore, equation B-14 also holds for i when $t = d_i + 1$. But by definition, d_i is the largest integer that satisfies equation B-14. This is a contradiction. The same conclusion can be reached for the case $r_{i_2} = 0$.

b. $r_{i_1} = r_{i_2} = 0$, because $2 \le d_1 \le d_2$, $\left\lceil \frac{d_i+2}{d_1} \right\rceil = \left\lceil \frac{d_i}{d_1} \right\rceil + 1$ and $\left\lceil \frac{d_i+2}{d_2} \right\rceil = \left\lceil \frac{d_i}{d_2} \right\rceil + 1$. Therefore, equation B-14, also holds when $t = d_i + 2$, again, a contradiction.

If $i = n - 1$, by the construction, either $d_{n-1} = t_1$ or $d_{n-1} = t_1 + 1$. In the former case, t_1 cannot be a multiple of d_1, because otherwise the same contradiction can be obtained as before. In the latter case, $d_{n-1} = t_1 + 1$, which means that $t_2 = t_1 + 2$. By equations B-15 and B-16, $\left\lceil \frac{t_1+2}{d_1} \right\rceil + \left\lceil \frac{t_1+2}{d_2} \right\rceil = \left\lceil \frac{t_1}{d_1} \right\rceil + \left\lceil \frac{t_1}{d_2} \right\rceil$. However, this equation holds only when $t_1 + 1$ is not a multiple of d_1 or d_2. Therefore, in both cases, d_{n-1} is not a multiple of d_1 or d_2.

Claim 4: $d_{i+1} = d_i + x$ for $3 \le i \le n - 3$, where $x = 1$ or 2 or 3.

Proof. Let $d_i = k_{i_1} d_1 + r_{i_1} = k_{i_2} d_2 + r_{i_2}$, where $1 \le r_{i_1} \le d_1 - 1$ and $1 \le r_{i_2} \le d_2 - 1$. Similarly, let $d_{i+1} = k_{(i+1)_1} d_1 + r_{(i+1)_1} = k_{(i+1)_2} d_2 + r_{(i+1)_2}$,where $1 \le r_{(i+1)_1} \le d_1 - 1$ and $1 \le r_{(i+1)_2} \le d_2 - 1$. There are three cases to consider:

- $r_{i_1} \le d_1 - 2$ and $r_{i_2} \le d_2 - 2$.

  Then $\left\lceil \frac{d_i+1}{d_1} \right\rceil = \left\lceil \frac{d_i}{d_1} \right\rceil + 1$ and $\left\lceil \frac{d_i+1}{d_2} \right\rceil = \left\lceil \frac{d_i}{d_2} \right\rceil$. Let $t = d_i + 1$. It is easy to show that equation B-14 holds for $i + 1$, but for any $t' > t$, equation B-14 does not hold. Therefore, $d_{i+1} = d_i + 1$.

- $r_{i_1} = d_1 - 1$ and $r_{i_2} \le d_2 - 3$, or $r_{i_1} \le d_1 - 3$ and $r_{i_2} = d_2 - 1$.

  Suppose that $r_{i_1} = d_1 - 1$ and $r_{i_2} \le d_2 - 3$, then $\left\lceil \frac{d_i+2}{d_1} \right\rceil = \left\lceil \frac{d_i}{d_1} \right\rceil + 1$ and $\left\lceil \frac{d_i+2}{d_2} \right\rceil = \left\lceil \frac{d_i}{d_2} \right\rceil$. Let $t = d_i + 2$. One can show that equation B-14 holds for $i + 1$, but for any $t' > t$, equation B-14 does not hold. Therefore, $d_{i+1} = d_i + 2$ in this case. For the case $r_{i_1} \le d_1 - 3$ and $r_{i_2} = d_2 - 1$, the proof is the same.

- $r_{i_1} = d_1 - 1$ and $r_{i_2} \le d_2 - 2$, or $r_{i_1} \le d_1 - 2$ and $r_{i_2} = d_2 - 1$.

  Using the same argument as above, one can prove that $d_{i+1} = d_i + 3$.

Claim 5: Let $d_3 = k_1 d_1 + r_1 = k_2 d_2 + r_2$, where $1 \le r_1 \le d_1 - 1$ and $1 \le r_2 \le d_2 - 1$. Express t_1 in terms of d_3. The expression depends on the value of r_1 and r_2, and it is shown in tables B-9 to B-16.

- $r_1 = r_2 = 1$. See table B-9.

TABLE B-9. VALUE OF t_1 WHEN $r_1 = r_2 = 1$

	$d_2 = 3$	$d_2 = 4$	$d_2 = 5$ or 6 or 7	$d_2 \geq 8$
$d_1 = 2$	$2d_3 - 15$	$2d_3 - 11$	$2d_3 - 9$	$2d_3 - 7$
$d_1 = 3$	$2d_3 - 9$	$2d_3 - 7$	$2d_3 - 6$	$2d_3 - 6$
$d_1 \geq 4$	-	$2d_3 - 7$	$2d_3 - 5$	$2d_3 - 5$

- $r_1 = 1$ and $r_2 = 2$. See table B-10.

TABLE B-10. VALUE OF t_1 WHEN $r_1 = 1$ AND $r_2 = 2$

	$d_2 = 3$	$d_2 = 4$	$d_2 \geq 5$
$d_1 = 2$	$2d_3 - 11$	$2d_3 - 9$	$2d_3 - 7$
$d_1 = 3$	$2d_3 - 6$	$2d_3 - 6$	$2d_3 - 6$
$d_1 \geq 4$		$2d_3 - 5$	$2d_3 - 5$

- $r_1 = 1$ and $3 \leq r_2 \leq \frac{d_2}{2}$. See table B-11.

TABLE B-11. VALUE OF t_1 WHEN $r_1 = 1$ AND $3 \leq r_2 \leq \frac{d_2}{2}$

$d_1 = 2$ and $r_2 = 1$	$d_1 = 2$ and $r_2 \geq 4$	$d_1 \geq 3$
$2d_3 - 7$	$2d_3 - 5$	$2d_3 - 4$

- $r_1 = 1$ and $r_2 > \frac{d_2}{2}$. If $d_2 = 5$ and $r_2 = 3$, then $t_1 = 2d_3 - 7$; otherwise, see table B-12.

TABLE B-12. VALUE OF t_1 WHEN $r_1 = 1$ AND $r_2 > \frac{d_2}{2}$

	$r_2 = \frac{d_2+1}{2}$ or $\frac{d_2+2}{2}$ or $\frac{d_2+3}{2}$	$r_2 = \frac{d_2+4}{2}$	$r_2 \geq \frac{d_2+5}{2}$
$d_1 = 2$	$2d_3 - 5$	$2d_3 - 5$	$2d_3 - 3$
$d_1 \geq 3$	$2d_3 - 4$	$2d_3 - 3$	$2d_3 - 3$

- $2 \leq r_1 \leq \frac{d_1}{2}$ and $r_2 = 1$.

If $r_1 = 2$, then $t_1 = 2d_3 - 5$; otherwise, $t_1 = 2d_3 - 4$.

- $2 \leq r_1 \leq \frac{d_1}{2}$ and $2 \leq r_2 \leq \frac{d_2}{2}$.

If $r_1 = r_2 = 2$, then $t_1 = 2d_3 - 5$; otherwise $t_1 = 2d_3 - 3$.

- $2 \le r_1 \le \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$, or $r_1 > \frac{d_1}{2}$ and $2 \le r_2 \le \frac{d_2}{2}$. See tables B-13 and B-14, respectively.

TABLE B-13. VALUE OF t_1 WHEN $2 \le r_1 \le \frac{d_1}{2}$ AND $r_2 > \frac{d_2}{2}$

$r_2 = \frac{d_2+1}{2}$	$r_2 = \frac{d_2+2}{2}$	$r_2 \ge \frac{d_2+3}{2}$
$2d_3 - 3$	$2d_3 - 3$	$2d_3 - 2$

TABLE B-14. VALUE OF t_1 WHEN $r_1 > \frac{d_1}{2}$ AND $2 \le r_2 \le \frac{d_2}{2}$

$r_1 = \frac{d_1+1}{2}$	$r_1 = \frac{d_1+2}{2}$	$r_1 \ge \frac{d_1+3}{2}$
$2d_3 - 3$	$2d_3 - 3$	$2d_3 - 2$

- $r_1 > \frac{d_1}{2}$ and $r_2 = 1$. See table B-15.

TABLE B-15. VALUE OF t_1 WHEN $r_1 > \frac{d_1}{2}$ AND $r_2 = 1$

$r_1 = \frac{d_1+1}{2}$ and $d_1 = 3$	$r_1 = \frac{d_1+1}{2}$ and $d_3 > 3$	$r_1 = \frac{d_1+2}{2}$ or $\frac{d_1+3}{2}$	$r_1 \ge \frac{d_1+4}{2}$
$2d_3 - 5$	$2d_3 - 4$	$2d_3 - 4$	$2d_3 - 3$

- $r_1 > \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$. See table B-16.

TABLE B-16. VALUE OF t_1 WHEN $r_1 > \frac{d_1}{2}$ AND $r_2 > \frac{d_2}{2}$

	$r_2 = \frac{d_2+1}{2}$	$r_2 = \frac{d_2+2}{2}$	$r_2 \ge \frac{d_2+3}{2}$
$r_1 = \frac{d_1+1}{2}$	$2d_3 - 3$	$2d_3 - 3$	$2d_3 - 2$
$r_1 = \frac{d_1+2}{2}$	$2d_3 - 3$	$2d_3 - 3$	$2d_3 - 1$
$r_1 \ge \frac{d_1+3}{2}$	$2d_3 - 2$	$2d_3 - 1$	$2d_3 - 1$

To check the correctness of the values of t_1 for different cases, it is sufficient to show that in each case

$$t_1 - 1 \le \left\lceil \frac{t_1-1}{d_1} \right\rceil + \left\lceil \frac{t_1-1}{d_2} \right\rceil + (2n-7)$$

$$t_1 = \left\lceil \tfrac{t_1}{d_1} \right\rceil + \left\lceil \tfrac{t_1}{d_2} \right\rceil + (2n-7)$$

$$t_1 + 1 > \left\lceil \tfrac{t_1+1}{d_1} \right\rceil + \left\lceil \tfrac{t_1+1}{d_2} \right\rceil + (2n-7),$$

all of which can be easily verified.

Claim 6: Let $d_3 = k_1 d_1 + r_1 = k_2 d_2 + r_2$, where $1 \le r_1 \le d_1 - 1$ and $1 \le r_2 \le d_2 - 1$. Depending on r_1 and r_2, the value of t_2 is shown in tables B-17 to B-23.

- $r_1 = r_2 = 1$. See table B-17.

TABLE B-17. VALUE OF t_2 WHEN $r_1 = r_2 = 1$

	$d_2 = 3$	$d_2 = 4$	$d_2 \ge 5$
$d_1 = 2$	$2d_3 - 8$	$2d_3 - 4$	$2d_3 - 4$
$d_1 = 3$	$2d_3 - 5$	$2d_3 - 3$	$2d_3 - 3$
$4 \le d_1 \le d_2$	-	$2d_3 - 3$	$2d_3 - 3$

- $r_1 = 1$ and $2 \le r_2 \le \frac{d_2}{2}$. See table B-18.

TABLE B-18. VALUE OF t_2 WHEN $r_1 = 1$ AND $2 \le r_2 \le \frac{d_2}{2}$

	$r_2 = 2$	$r_2 \ge 3$
$d_1 = 2$	$2d_3 - 4$	$2d_3 - 2$
$d_1 \ge 3$	$2d_3 - 2$	$2d_3 - 2$

- $r_1 = 1$ and $r_2 > \frac{d_2}{2}$. See table B-19.

TABLE B-19. VALUE OF t_2 WHEN $r_1 = 1$ AND $r_2 > \frac{d_2}{2}$

$r_2 = \dfrac{d_2+1}{2}$	$r_2 = \dfrac{d_2+2}{2}$	$r_2 = \dfrac{d_2+3}{2}$	$r_2 \ge \dfrac{d_2+4}{2}$
$2d_3 - 2$	$2d_3 - 2$	$2d_3$	$2d_3$

- $2 \le r_1 \le \frac{d_1}{2}$ and $r_2 = 1$.

  $t_2 = 2d_3 - 2$.

- $2 \le r_1 \le \frac{d_1}{2}$ and $2 \le r_2 \le \frac{d_2}{2}$.

  $t_2 = 2d_3 - 1$.

- $2 \le r_1 \le \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$, or $r_1 > \frac{d_1}{2}$ and $2 \le r_2 \le \frac{d_2}{2}$. See tables B-20 and B-21, respectively.

TABLE B-20. VALUE OF t_2 WHEN $2 \le r_1 \le \frac{d_1}{2}$ AND $r_2 > \frac{d_2}{2}$

$r_2 = \frac{d_2+1}{2}$	$r_2 = \frac{d_2+2}{2}$	$r_2 \ge \frac{d_2+3}{2}$
$2d_3 - 1$	$2d_3$	$2d_3$

TABLE B-21. VALUE OF t_2 WHEN $r_1 > \frac{d_1}{2}$ AND $2 \le r_2 \le \frac{d_2}{2}$

$r_1 = \frac{d_1+1}{2}$	$r_1 = \frac{d_1+2}{2}$	$r_1 \ge \frac{d_1+3}{2}$
$2d_3 - 1$	$2d_3$	$2d_3$

- $r_1 > \frac{d_1}{2}$ and $r_2 = 1$. See table B-22.

TABLE B-22. VALUE OF t_2 WHEN $r_1 > \frac{d_1}{2}$ AND $r_2 = 1$

$r_1 = \frac{d_1+1}{2}$ or $\frac{d_1+2}{2}$	$r_1 = \frac{d_1+3}{2}$	$r_1 \ge \frac{d_1+4}{2}$
$2d_3 - 2$	$2d_3$	$2d_3$

- $r_1 > \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$. See table B-23.

TABLE B-23. VALUE OF t_2 WHEN $r_1 > \frac{d_1}{2}$ AND $r_2 > \frac{d_2}{2}$

	$r_2 = \frac{d_2+1}{2}$	$r_2 = \frac{d_2+2}{2}$	$R_2 \geq \frac{d_2+3}{2}$
$r_1 = \frac{d_1+1}{2}$	$2d_3 - 1$	$2d_3 + 1$	$2d_3 + 1$
$r_1 = \frac{d_1+2}{2}$	$2d_3 + 1$	$2d_3 + 1$	$2d_3 + 1$
$r_1 \geq \frac{d_1+3}{2}$	$2d_3 + 1$	$2d_3 + 1$	$2d_3 + 1$

Claim 7: Based on t_1 and t_2, d_{n-1} and d_n are constructed as shown in tables B-24 to B-26.

- $r_1 = r_2 = 1$. See table B-24.

TABLE B-24. VALUE OF d_{n-1} AND d_n WHEN $r_1 = r_2 = 1$

	$d_2 = 3$	$d_2 = 4$	$d_2 = 5$ or 6	$d_2 \geq 7$
$d_1 = 2$	$d_{n-1} = 2d_3 - 15$ $d_n = 2d_3 - 8$	$d_{n-1} = 2d_3 - 11$ $d_n = 2d_3 - 4$	$d_{n-1} = 2d_3 - 9$ $d_n = 2d_3 - 4$	$d_{n-1} = 2d_3 - 7$ $d_n = 2d_3 - 4$
$d_1 = 3$	$d_{n-1} = 2d_3 - 9$ $d_n = 2d_3 - 5$	$d_{n-1} = 2d_3 - 7$ $d_n = 2d_3 - 3$	$d_{n-1} = 2d_3 - 6$ $d_n = 2d_3 - 3$	$d_{n-1} = 2d_3 - 6$ $d_n = 2d_3 - 3$
$4 \leq d_1 \leq d_2$	- -	$d_{n-1} = 2d_3 - 7$ $d_n = 2d_3 - 3$	$d_{n-1} = 2d_3 - 4$ $d_n = 2d_3 - 4$	$d_{n-1} = 2d_3 - 4$ $d_n = 2d_3 - 4$

- $r_1 = 1$ and $r_2 = 2$. See table B-25.

TABLE B-25. VALUE OF d_{n-1} AND d_n WHEN $r_1 = 1$ AND $r_2 = 2$

	$d_2 = 3$	$d_2 = 4$	$d_2 \geq 5$
$d_1 = 2$	$d_{n-1} = 2d_3 - 11$ $d_n = 2d_3 - 4$	$d_{n-1} = 2d_3 - 9$ $d_n = 2d_3 - 4$	$d_{n-1} = 2d_3 - 7$ $d_n = 2d_3 - 4$
$d_1 = 3$	$d_{n-1} = 2d_3 - 6$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 6$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 6$ $d_n = 2d_3 - 2$
$d_1 \geq 4$		$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3 - 2$

- $r_1 = 1$ and $3 \le r_2 \le \frac{d_2}{2}$. See table B-26.

TABLE B-26. VALUE OF d_{n-1} AND d_n WHEN $r_1 = 1$ AND $3 \le r_2 \le \frac{d_2}{2}$

$d_1 = 2$ and $r_2 = 3$	$d_1 = 2$ and $r_2 \ge 4$	$d_2 \ge 3$
$d_{n-1} = 2d_3 - 7$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3 - 3$

- $r_1 = 1$ and $3 \le r_2 \le \frac{d_2}{2}$.

If $d_2 = 5$ and $r_2 = 3$, then $d_{n-1} = 2d_3$-7 and $d_n = 2d_3$-2; otherwise, see table B-27.

TABLE B-27. VALUE OF d_{n-1} AND d_n WHEN $r_1 = 1$ AND $3 \le r_2 \le \frac{d_2}{2}$

	$r_2 = \dfrac{d_2+1}{2}$ or $\dfrac{d_2+2}{2}$	$r_2 = \dfrac{d_2+3}{2}$	$r_2 = \dfrac{d_2+4}{2}$	$r_2 \ge \dfrac{d_2+5}{2}$
$d_1 = 2$	$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$
$d_1 \ge 3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3 - 3$	$d_{n-1} = 2d_3 - 4$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$

- $2 \le r_1 \le \frac{d_1}{2}$ and $r_2 = 1$.

If $r_1 = 2$, then $d_{n-1} = t_1 = 2d_3$-5 and $d_n = t_2 = 2d_3 - 2$; otherwise, $d_{n-1} = d_n = 2d_3$ - 3.

- $2 \le r_1 \le \frac{d_1}{2}$ and $2 \le r_2 \le \frac{d_2}{2}$.

If $r_1 = r_2 = 2$, then $d_{n-1} = 2d_3$ - 5 and $d_n = 2d_3$ –1; otherwise, $t_2 = t_1 + 2$ and hence

$d_{n-1} = d_n = t_1 + 1 = 2d_3$ - 2.

- $2 \le r_1 \le \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$, or $r_1 > \frac{d_1}{2}$ and $2 \le r_2 \le \frac{d_2}{2}$. See tables B-28 and B-29, respectively.

TABLE B-28. VALUE OF d_{n-1} AND d_n WHEN $2 \le r_1 \le \frac{d_1}{2}$ AND $r_2 > \frac{d_2}{2}$

$r_2 = \dfrac{d_2+1}{2}$	$r_2 = \dfrac{d_2+2}{2}$	$r_2 \ge \dfrac{d_2+3}{2}$
$d_{n-1} = 2d_3 - 2$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 1$ $d_n = 2d_3 - 1$

TABLE B-29. VALUE OF d_{n-1} AND d_n WHEN $r_1 > \frac{d_1}{2}$ AND $2 \le r_2 \le \frac{d_2}{2}$

$r_1 = \frac{d_1+1}{2}$	$r_1 = \frac{d_1+2}{2}$	$r_1 \ge \frac{d_1+3}{2}$
$d_{n-1} = 2d_3 - 2$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 1$ $d_n = 2d_3 - 1$

- $r_1 > \frac{d_1}{2}$ and $r_2 = 1$. See table B-30.

TABLE B-30. VALUE OF d_{n-1} AND d_n WHEN $r_1 > \frac{d_1}{2}$ AND $r_2 = 1$

$r_1 = \frac{d_1+1}{2}$ and $d_1=3$	$r_1 = \frac{d_1+1}{2}$ and $d_1 > 3$	$r_1 = \frac{d_1+2}{2}$	$r_1 = \frac{d_1+3}{2}$	$r_1 \ge \frac{d_1+4}{2}$
$d_{n-1} = 2d_3 - 5$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3 - 3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 4$ $d_n = 2d_3$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3$

- $r_1 > \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$. See table B-31.

TABLE B-31. VALUE OF d_{n-1} AND d_n WHEN $r_1 > \frac{d_1}{2}$ AND $r_2 > \frac{d_2}{2}$

	$r_2 = \frac{d_2+1}{2}$	$r_2 = \frac{d_2+2}{2}$	$r_2 \ge \frac{d_2+3}{2}$
$r_1 = \frac{d_1+1}{2}$	$d_{n-1} = 2d_3 - 2$ $d_n = 2d_3 - 2$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3 + 2$	$d_{n-1} = 2d_3 - 2$ $d_n = 2d_3 + 2$
$r_1 = \frac{d_1+2}{2}$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3 + 2$	$d_{n-1} = 2d_3 - 3$ $d_n = 2d_3 + 2$	$d_{n-1} = 2d_3 - 1$ $d_n = 2d_3 + 2$
$r_1 \ge \frac{d_1+3}{2}$	$d_{n-1} = 2d_3 - 2$ $d_n = 2d_3 + 2$	$d_{n-1} = 2d_3 - 1$ $d_n = 2d_3 + 2$	$d_{n-1} = 2d_3 - 1$ $d_n = 2d_3 + 2$

Claim 8: $d_n - 1$ is not a multiple of d_1 nor d_2.

Proof. There are three cases to consider:

- $d_n - 1 = t_2$, i.e., $d_n = t_2 + 1$.

  By construction of d_n, this case happens when $t_2 = kd_3 + 1$ for some k. From the tables in claim 7, one can see that this case happens only when $r_1 > \frac{d_1}{2}$ and $r_2 > \frac{d_2}{2}$, where $t_2 = 2d_3 + 1$. Let $d_3 = k_1d_1 + r_1$. Then $t_2 = 2k_1d_1+2r_1+1$. Since $\frac{d_1}{2} \le r_1 \le d_1 - 1$,

$d_1 + 1 \le 2r_1 + 1 \le 2d_1 - 1 < 2d_1$. Therefore, $(2k_1 + 1)d_1 < t_2 < (2k_1 + 2)d_1$, which implies that t_2 cannot be a multiple of d_1. Similarly, one can prove that t_2 is not a multiple of d_2.

- $d_n - 1 = t_2 - 1$, i.e., $d_n = t_2$.

  Suppose that $t_2 - 1$ is a multiple of d_1 but not d_2, then $\left\lceil \frac{t_2-1}{d_1} \right\rceil = \left\lceil \frac{t_2}{d_1} \right\rceil - 1$ and $\left\lceil \frac{t_2-1}{d_2} \right\rceil = \left\lceil \frac{t_2}{d_2} \right\rceil - 1$. Hence, equation B-16 also holds for $t_2 - 1$. However, by definition, t_2 is the minimum integer that satisfies equation B-16. This is a contradiction. Using a similar argument, one can prove that $d_n - 1$ cannot be a multiple of d_2.

- $d_n - 1 = t_2 - 2$, i.e., $d_n = t_2 - 1$.

  In this case, $d_n = d_{n-1}$, which means that $t_2 = t_1 + 2$. By equation B-15 and B-16, one must have $\left\lceil \frac{t_2-2}{d_1} \right\rceil = \left\lceil \frac{t_2}{d_1} \right\rceil$ and $\left\lceil \frac{t_2-2}{d_2} \right\rceil = \left\lceil \frac{t_2}{d_2} \right\rceil$. Thus, $d_n - 1$ or $t_2 - 2$ cannot be a multiple of d_1 or d_2.

Theorem 12: The perfect instance $T_1, T_2, \ldots, T_n$ constructed above perfectly utilizes the processor.

Below are some lemmas about the DM schedule for the perfect instance. The theorem follows directly from these lemmas and the definition of perfect utilization.

Lemma 2: For the DM schedule of the perfect instance, there is no idle time before t_1.

Proof. It is enough to prove that at any time $t < t_1$, the number of requests R from $T_1, \ldots, T_{n-2}, T_{n-1}$ is greater than or equal to t. There are two cases depending on t.

- $t = d_i$.

  $t \le d_j$ for $j \ge i$. So $T_i, \ldots, T_{n-1}$ each makes exactly one request before t. On the other hand, $d_k \le t \le 2d_k$ for $3 \le k \le i - 1$. So $T_3, \ldots, T_{i-1}$ each makes exactly two requests. Therefore, the total number of requests before time t from $T_1, \ldots, T_{n-1}$ is $R = \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + n + i - 6$. By construction, $t = d_i = R$.

- $d_i < t < d_{i+1}$.

- $t < d_j$ for $j \ge i+1$. So $T_{i+1}, \ldots, T_{n-1}$ each makes exactly one request before t. On the other hand, $d_j < t \le 2d_j$ for $3 \le j \le i$. So $T_3, \ldots, T_i$ each makes exactly two requests. Therefore, at time t, the total number of requests from $T_1, \ldots, T_{n-1}$ is $R = \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + n + (i+1) - 6$. By the construction of d_{i+1}, $t \le R$.

Lemma 3: There is no suspending request from $T_3, \ldots, T_{n-2}, T_{n-1}$ at t_1.

Proof. One can show that the number of requests from $T_1, \ldots, T_{n-1}$ during the period $[0, t_1)$ is equal to t_1.

Lemma 4: None of $T_4, \ldots, T_{n-2}$ will make a request during the period $[t_1, d_n\text{-}1]$.

Proof. Because $d_j < t_1 < d_n - 1 \le 2d_3 + 1 < 2d_j$ for $4 \le j \le n\text{-}2$, T_j makes exactly two requests before t_1 and will not make its third request before d_n-1.

Lemma 5: There is no idle time during the period $[t_1, d_n\text{-}1)$ and there is no request from T_1, T_2, T_3, and T_{n-1} at d_n-1. Therefore, the first request from T_n will be executed at d_n-1.

Proof. Let $t_1 \le t \le d_n\text{-}1$. To prove that there is no idle time from t_1 to d_n-1, it is sufficient to show that at t the number of requests from all tasks except T_n is greater than or equal to t.

Consider first $d_n = t_2 \le 2d_3$. At t the requests from T_1 and T_2 are $\left\lceil \frac{t}{d_1} \right\rceil$ and $\left\lceil \frac{t}{d_2} \right\rceil$, respectively. Task T_j, $2 \le j \le n\text{-}1$, has two requests. Together with $T_3, \ldots, T_{n-1}$, there are $2n$-6 requests. The total number of requests by $T_1, \ldots, T_{n-1}$ is $R = \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + (2n-6)$. By the definition of t_2, $t < \left\lceil \frac{t}{d_1} \right\rceil + \left\lceil \frac{t}{d_2} \right\rceil + (2n-5) \le R$.

If $d_n = t_2$ -1 or $d_n = t_2 + 1$, one can use a similar argument.

It will now be shown that there is no request released at d_n-1. It has already been shown that d_n-1 is not a multiple of d_1 or d_2, so neither T_1 nor T_2 makes a request at d_n-1. Since either $d_3 < d_n\text{-}1 < 2d_3$ or $d_n - 1 = 2d_3 + 1$, T_1 will not make its third or fourth request at d_n-1. By construction, either $d_n\text{-}1 = d_{n-1} - 1$ or $d_{n-1} < d_n\text{-}1 < 2d_{n-1}$. So T_{n-1} will not make its second or third request at time d_n-1.

Combining Lemmas 2, 3, and 4, the processor is busy from time 0 to d_n-1, all requests from $T_1, \ldots, T_{n-1}$ have been executed before time d_n-1, and there is no request released at time d_n-1. So the first request of T_n will be executed at time d_n-1. By definition, the perfect instance perfectly utilizes the processor. Thus, theorem 12 is proved.

B.5.3 PERFECT INSTANCE WHEN $d_2 \ge d_1 > \frac{n+1}{2}$.

First, the following lemma is very easy to verify.

Lemma 6: Given n and $d_2 \ge d_1 > \frac{n+1}{2}$, let the perfect instance for n, d_1, and d_2 be $TS = \{T_1(1,d_1), T_2(1,d_2), T_3(1,d_3), \ldots, T_n(1,d_n)\}$, then $d_3 = n + 1$.

Claim 9: Given n and $d_2 \ge d_1 > \frac{n+1}{2}$, let the perfect instance for n, d_1, and d_2 be $TS = \{T_1(1,d_1), T_2(1,d_2), T_3(1,d_3), \ldots, T_n(1,d_n)\}$, and the total utilization be $U(d_2)$. Let

$TS' = \{T_1(1,d_1), T_2(1,d'_2 = d_2+1), T'_3(1,d'_3),...,T'_n(1,d'_n)\}$ be the perfect instance for n, d_1, and d_2+1, and its total utilization be $U(d_2+1)$, then $U(d_2) \geq U(d_2+1)$.

The proof is by showing that $U(d_2) - U(d_2+1) = \sum_{i=1}^{n} \frac{1}{d_i} - \frac{1}{d'_i} > 0$ for every possible d_1 and d_2. The cases are listed in table B-32.

TABLE B-32. THE CASES OF d_i AND d'_i DEPENDING ON d_1 AND d_2

d_1 \ d_2	$[\frac{n+2}{2}, \frac{2n-4}{3}]$	$\frac{2n-3}{3}$	$\frac{2n-2}{3}$	$\frac{2n-1}{3}$	$\frac{2n}{3}$	$\frac{2n+1}{3}$	$\frac{2n+2}{3}$	$[\frac{2n+3}{3}, n-3]$	n-2	n-1
$[\frac{n+2}{2}, \frac{2n-4}{3}]$	7	8	9	10	11	11	11	12	12	15
$\frac{2n-3}{3}$	*	16	**	**	17	**	**	12	12	15
$\frac{2n-2}{3}$	*	*	18	**	**	17	**	12	12	19
$\frac{2n-1}{3}$	*	*	*	20	**	**	21	13	14	19
$\frac{2n}{3}$	*	*	*	*	20	**	**	13	22	19
$\frac{2n+1}{3}$	*	*	*	*	*	20	**	13	23	19
$\frac{2n+2}{3}$	*	*	*	*	*	*	20	24	23	19
$[\frac{2n+3}{3}, n-3]$	*	*	*	*	*	*	*	25	23	19
n-2	*	*	*	*	*	*	*	*	26	19
n-1	*	*	*	*	*	*	*	*	*	27

Note: * impossible cases because $d_1 \leq d_2$.
** impossible cases because both d_1 and d_2 are integers.
Other entries are the numbers of lemmas that prove the corresponding cases.

From lLemma 6, it follows that $d_3 = d'_3 = n+1$, $r_1 = n+1-d_1$, $r_2 = n+1-d_2$ and $r'_2 = r_2-1$. Let $S_1 = \sum_{i=3}^{n-2} (\frac{1}{d'_i} - \frac{1}{d_i})$ and $S_2 = (\frac{1}{d'_{n+1}} + \frac{1}{d'_n}) - (\frac{1}{d_{n+1}} + \frac{1}{d_n})$. Then $U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - S_1 - S_2$. The values of d_i, $4 \leq i \leq n$, depend on the values of d_1 and d_2. Since S_1 and S_2 depend on the values of d_i, their values depend on d_1 and d_2 as well. The claim will be proved for each case. For all cases, the idea is the same: From d_1 and d_2, one can obtain r_1, r_2 and r'_2. Applying claim 7, it is easy to calculate S_2. To calculate S_1, one needs to find d_i and d'_i for $4 \leq i \leq n$-2. Specifically, one needs to express d_i and d'_i in terms of d_3. As it has been shown in claim 4, d_{i+1} is usually greater than d_i by 1. Thus, $d_i = d_3+i-3$. But it has been proved that d_i cannot be a multiple of d_1 or d_2. Therefore, one needs to do some corrections to this formula to jump over those multiples of d_1 or d_2. Initially, d_i is increased by 1 to get d_{i+1}, so $d_i = d_3 + i-3$. At some point $d_i = 2d_1 - 1$. To jump over $2d_1$, the formula becomes $d_i = d_3+i-2$. After that d_i is continuously increased by 1 to d_{i+1}, until at another point where $d_i = 2d_2-1$ or $d_i = 3d_1-1$, then the formula is changed again. These points (the indices of the tasks i, $3 \leq i \leq n$-2) where the formula is corrected varies with d_1 and d_2.

Lemma 7: $U(d_2) \geq U(d_2{+}1)$ when $\frac{n+2}{2} < d_1 \leq d_2 \leq \frac{2(n-2)}{3}$.

Proof. $\frac{d_1+6}{2} < r_1 \leq d_1 - 1$, $\frac{d_2+6}{2} < r_2 \leq d_2 - 1$, and $\frac{d_2+3}{2} < r_2^{'} \leq d_2 - 3$.

By table B-31 in claim 7, $d'_{n-1} = d_{n-1}$, and $d'_n = d_n$. Therefore, S_2=0.
To calculate S_1, first consider the case $d_1 < d_2$. $\frac{2d_2+2}{3} = \frac{2d_2^{'}}{3} < d_1 < d_2$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \leq i \leq 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \leq i \leq 3d_2 - n - 2 \\ d_3 + i + 1 & \text{if } 3d_2 - n - 1 \leq i \leq n - 2 \end{cases}$$

$$d_i^{'} = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \leq i \leq 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \leq i \leq 3d_2 - n + 1 \\ d_3 + i + 1 & \text{if } 3d_2 - n + 2 \leq i \leq n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-33.

TABLE B-33. d_i AND d'_i WHEN $\frac{n+2}{2} \leq d_1 < d_2 \leq \frac{2(n-2)}{3}$

i	$2d_2 - n + 1$	$2d_2 - n + 2$	$3d_2 - n - 1$	$3d_2 - n$	$3d_2 - n + 1$
d_i	$2d_2 + 1$	$2d_2 + 2$	$3d_2 + 1$	$3d_2 + 2$	$3d_2 + 3$
d'_i	$2d_2$	$2d_2 + 1$	$3d_2$	$3d_2 + 1$	$3d_2 + 2$

$$S_1 = (\frac{1}{2d_2} + \frac{1}{3d_2}) - (\frac{1}{2d_2 + 2} - \frac{1}{3d_2 + 3}) = \frac{5}{6d_2 \cdot (d_2 + 1)}$$

$$U(d_2) - U(d_2 + 1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{5}{6d_2 \cdot (d_2+1)} - 0 > 0$$

Now consider the case where $d_1 = d_2$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 1 & \text{if } 2d_1 - n + 2 \le i \le 3d_1 - n - 1 \\ d_3 + i + 1 & \text{if } 3d_1 - n \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_1 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_1 - n + 3 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } i = 3d_1 - n \text{ or } 3d_1 - n + 1 \\ d_3 + i + 1 & \text{if } 3d_1 - n + 2 \le i \le n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-34.

TABLE B-34. d_i AND d'_i WHEN $\frac{n+2}{2} \le d_1 = d_2 \le \frac{2(n-2)}{3}$

i	$2d_2 - n + 2$	$3d_2 - n$	$3d_2 - n + 1$
d_i	$2d_2 + 2$	$3d_2 + 2$	$3d_2 + 3$
d'_i	$2d_2+1$	$3d_2 + 1$	$3d_2 + 2$

$$S_1 = (\frac{1}{2d_1 + 1} + \frac{1}{3d_1 + 1}) - (\frac{1}{2d_1 + 2} - \frac{1}{3d_1 + 3})$$

$$U(d_2) - U(d_2 + 1) = \frac{1}{d_2(d_2 + 1)} - ((\frac{1}{2d_1 + 1} + \frac{1}{3d_1 + 1}) - (\frac{1}{2d_1 + 2} - \frac{1}{3d_1 + 3})) - 0 > 0 .$$

Lemma 8: $U(d_2) \ge U(d_2+1)$ when $\frac{n+2}{2} \le d_1 \le \frac{2n-4}{3}$ and $d_2 = \frac{2n-3}{3}$.

Proof. $r_1 \ge \frac{d_1+6}{2}$, $r_2 = \frac{n+6}{3} = \frac{d_2+5}{2}$, $d'_2 = d_2 + 1 = \frac{2n}{3}$, and $r'_2 = \frac{n+3}{3} = \frac{d_2+2}{2}$. By table B-31 in claim 7, $d'_n = d_n$ and $d'_{n-1} = d_{n-1} = 2d_3 + 2 = 2(n+2)$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n - 1 \le i \le 3d_2 - n - 2 \\ d_3 + i + 1 & \text{if } i = n - 4 \text{ or } n - 3 \text{ or } n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 + 2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-35.

TABLE B-35. d_i AND d'_i WHEN $\frac{n+2}{2} \le d_1 = d_2 \le \frac{2n-3}{3}$

i	$2d_2 - n + 1$	$2d_2 - n + 2$	n -4	n -3	n -2
d_i	$2d_2 + 1$	$2d_2 + 2$	$2n$-2	$2n$-1	$2n$
d'_i	$2d_2$	$2d_2$+1	$2n$-3	$2n$-2	$2n$-1

$$S_1 = \left(\frac{1}{2d_2} + \frac{1}{2n-3}\right) - \left(\frac{1}{2d_2+2} + \frac{1}{2n}\right)$$

$$U(d_2) - U(d_2+1) = \tfrac{1}{d_2 \cdot (d_2+1)} - \left(\tfrac{1}{2d_2} + \tfrac{1}{2n-3}\right) - \left(\tfrac{1}{2d_2+2} + \tfrac{1}{2n}\right) - 0 > 0 .$$

Lemma 9: $U(d_2) \ge U(d_2+1)$ when $\frac{n+2}{2} \le d_1 \le \frac{2n-4}{3}$ and $d_2 = \frac{2n-2}{3}$.

Proof. $r_1 \ge \frac{d_1+6}{2}$, $r_2 = \frac{n+5}{3} = \frac{d_2+4}{2}$, $d'_2 = d_2 + 1 = \frac{2n+1}{3}$, and $r'_2 = \frac{n+2}{3} = \frac{d'_2+1}{2}$. By table B-31 in claim 7, $d_{n-1} = 2d_3$-1 = $2n$+1, $d_n = 2d_3$+2 = $2n$+4, $d'_{n-1} = 2d_3$ –2 = $2n$, and $d'_n = 2d_3 + 2 = 2n + 4$. Therefore,

$$S_2 = \frac{1}{2n(2n+1)}$$

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le 3d_2 - n - 2 = n - 4 \\ d_3 + i + 1 & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-36.

TABLE B-36. d_i AND d'_i WHEN $\frac{n+2}{2} \le d_1 = d_2 \le \frac{2n-2}{3}$

i	$2d_2 - n + 1$	$2d_2 - n + 2$	n -4	n -3
d_i	$2d_2 + 1$	$2d_2 + 2$	$2n$-1	$2n$
d'_i	$2d_2$	$2d_2$+1	$2n$-2	$2n$-1

$$S_1 = \frac{1}{2d_2(d_2+1)} + \frac{1}{2n(n-1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \left(\frac{1}{2d_2(d_2+1)} + \frac{1}{n(2n-1)}\right) - \frac{1}{2n(2n+1)} > 0.$$

Lemma 10: $U(d_2) \ge U(d_2+1)$ when $\frac{n+2}{2} \le d_1 \le \frac{2n-4}{3}$ and $d_2 = \frac{2n-1}{3}$.

Proof. $r_1 \ge \frac{d_1+6}{2}$, $r_2 = \frac{n+4}{3} = \frac{d_2+3}{2}$, $d'_2 = d_2 + 1 = \frac{2n+2}{3}$, and $r_2 = \frac{n+1}{3} = \frac{d_2}{2}$. By tables B-29 and B-31 in claim 7, $d_{n-1} = 2d_3 - 1 = 2n+1$, $d_n = 2d_3 + 2 = 2n+4$, and $d'_{n-1} = d'_n = 2d_3 - 1 = 2n+1$. Therefore,

$$S_2 = \frac{3}{(2n+1)(2n+4)}$$

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le 3d_2 - n - 2 = n - 3 \\ d_3 + i + 1 & \text{if } i = n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-37.

TABLE B-37. d_i AND d'_i WHEN $\frac{n+2}{2} \le d_1 = d_2 \le \frac{2n-1}{3}$

i	$2d_2 - n + 1$	$2d_2 - n + 2$	n -3
d_i	$2d_2 + 1$	$2d_2 + 2$	$2n$
d'_i	$2d_2$	$2d_2+1$	$2n$-1

$$S_1 = \frac{1}{2d_2(d_2+1)} + \frac{1}{2n(2n-1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - \frac{1}{2n(2n-1)} - \frac{3}{(2n+1)(2n+4)} > 0\,.$$

Lemma 11: $U(d_2) \ge U(d_2+1)$ when $\frac{n+2}{2} \le d_1 \le \frac{2n-4}{3}$ and $d_2 = \frac{2n}{3}$ or $\frac{2n+1}{3}$ or $\frac{2n+2}{3}$.

Proof. First, calculate S_1.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \le i \le n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-38.

TABLE B-38. d_i AND d'_i WHEN $\frac{n+2}{2} \le d_1 \le \frac{2n-4}{3}$ AND $d_2 = \frac{2n}{3}$ OR $\frac{2n+1}{3}$ OR $\frac{2n+2}{3}$

i	$2d_2 - n + 1$	$2d_2 - n + 2$
d_i	$2d_2 + 1$	$2d_2 + 2$
d'_i	$2d_2$	$2d_2+1$

$$S_1 = \frac{1}{2d_2(d_2+1)}$$

To calculate S_2, consider the following three cases:

1. $d_2 = \frac{2n}{3}$.

   $r_1 \geq \frac{d_1+6}{2}$, $r_2 = \frac{n+3}{3} = \frac{d_2+2}{2}$, $d_2^{'} = d_2 + 1 = \frac{2n+3}{3}$, and $r_2^{'} = \frac{n}{3} = \frac{d_2^{'}-1}{2}$. By tables B-29 and B-31 in claim 7, $d_{n-1} = 2d_3$-1 = $2n$+1, $d_n = 2d_3$+2 = $2n$+4, and $d'_{n-1} = d'_n$=$2d_3$ -1 =$2n$+1. Therefore,

$$S_2 = \frac{3}{(2n+1)(2n+4)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - \frac{3}{(2n+1)(2n+4)} > 0 .$$

2. $d_2 = \frac{2n}{3}$.

   $r_1 \geq \frac{d_1+6}{2}$, $r_2 = \frac{n+2}{3} = \frac{d_2+1}{2}$, $d_2^{'} = d_2 + 1 = \frac{2n+4}{3}$, and $r_2^{'} = \frac{n-1}{3} = \frac{d_2^{'}-2}{2}$. By tables B-29 and B-31 in claim 7, d_{n-1}= $2d_3$-2 = $2n$, $d_n = 2d_3$+2 = $2n$+4, and $d'_{n-1} = d'_n$= $2d_3$ -1 = $2n$+1. Therefore,

$$S_2 = \frac{2}{2n+1} - \frac{1}{2n+4} - \frac{1}{2n}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - \left(\frac{2}{2n+1} - \frac{1}{2n+4} - \frac{1}{2n}\right) > 0 .$$

3. $d_2 = \frac{2n}{3}$.

   $r_1 \geq \frac{d_1+6}{2}$, $r_2 = \frac{n+1}{3} = \frac{d_2}{2}$, $d_2^{'} = d_2 + 1 = \frac{2n+5}{3}$, and $r_2^{'} = \frac{n-2}{3} = \frac{d_2-3}{2}$. By table B-29 in claim 7, $d_{n-1} = d_n$ = $2d_3$ -1 = $2n$ + 1 and $d'_{n-1} = d'_n$ = $2d_3$ -1 = $2n$+1. Therefore, S_2 = 0.

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - 0 > 0 .$$

Lemma 12: $U(d_2) \geq U(d_2+1)$ when $\frac{n+2}{2} \leq d_1 \leq \frac{2n-2}{3}$ and $\frac{2n+3}{3} \leq d_2 \leq n-3$.

Proof. $r_1 \geq \frac{n+2}{3} \geq \frac{d_1+4}{2}$, $4 \leq r_2 \leq \frac{n}{3} < \frac{d_2}{2}$, and $3 \leq r_2^{'} = r_2 - 1 < \frac{d_2}{2}$. By table B-29 in claim 7, $d_n = d'_n$ and $d_{n-1} = d'_{n-1}$. Therefore, $S_2 = 0$.

1. $d_1 \le \frac{2d_2-2}{3}$.

$$d_i = \begin{cases} d_3+i-3 & \text{if } 3 \le i \le 2d_1-n+1 \\ d_3+i-2 & \text{if } 2d_1-n+2 \le i \le 3d_1-n \\ d_3+i-1 & \text{if } 3d_1-n+1 \le i \le 2d_2-n-1 \\ d_3+i & \text{if } 2d_2-n \le i \le n-2 \end{cases}$$

$$d'_i = \begin{cases} d_3+i-3 & \text{if } 3 \le i \le 2d_1-n+1 \\ d_3+i-2 & \text{if } 2d_1-n+2 \le i \le 3d_1-n \\ d_3+i-1 & \text{if } 3d_1-n+1 \le i \le 2d_2-n+1 \\ d_3+i & \text{if } 2d_2-n+2 \le i \le n-2 \end{cases}$$

If $i = 2d_2 - n$, then $d_i = n + 1 + 2d_2 - n = 2d_2 - n = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2 + 1$.

$$S_1 = \frac{1}{2d_2} - \frac{1}{2d_2+2} = \frac{1}{2d_2\cdot(d_2+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2\cdot(d_2+1)} - \frac{1}{2d_2\cdot(d_2+1)} - 0 > 0\,.$$

2. $d_1 = \frac{2d_2-1}{3}$.

$$d_i = \begin{cases} d_3+i-3 & \text{if } 3 \le i \le 2d_1-n+1 \\ d_3+i-2 & \text{if } 2d_1-n+2 \le i \le 3d_1-n = 2d_2-n-1 \\ d_3+i & \text{if } 2d_2-n \le i \le n-2 \end{cases}$$

$$d'_i = \begin{cases} d_3+i-3 & \text{if } 3 \le i \le 2d_1-n+1 \\ d_3+i-2 & \text{if } 2d_1-n+2 \le i \le 3d_1-n = 2d_2-n-1 \\ d_3+i-1 & \text{if } i = 2d_2-n \text{ or } 2d_2-n+1 \\ d_3+i & \text{if } 2d_2-n+2 \le i \le n-2 \end{cases}$$

If $i = 2d_2 - n$, then $d_i = n + 1 + 2d_2 - n = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2 + 1$.

$$S_1 = \frac{1}{2d_2(d_2+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2 \cdot (d_2+1)} - 0 > 0\,.$$

3. $d_1 = \frac{2d_2}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i & \text{if } 2d_2 - n + 1 \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 3d_1 - n = 2d_2 - n \\ d_3 + i - 1 & \text{if } i = 2d_2 - n + 1 \\ d_3 + i & \text{if } 2d_2 - n + 2 \le i \le n - 2 \end{cases}$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2 + 1$.

$$S_1 = \frac{1}{2(2d_2+1)(d_2+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2(2d_2+1) \cdot (d_2+1)} - 0 > 0\,.$$

4. $d_1 = \frac{2d_2+1}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i & \text{if } 2d_2 - n + 1 \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 1\,. \\ d_3 + i & \text{if } 2d_2 - n + 2 \le i \le n - 2 \end{cases}$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 = 2d_2 + 2$ and $d'_i = d_i - 2 = 2d_2$.

$$S_1 = \frac{1}{2d_2(d_2+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2 \cdot (d_2+1)} - 0 > 0\,.$$

5. $d_1 = \frac{2d_2+2}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } i = 2d_2 - n + 1 \\ d_3 + i & \text{if } 2d_2 - n + 2 \le i \le n - 2 \end{cases}$$

If $d_2 = n - 2$,

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 = n - 2 \end{cases}$$

Otherwise,

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 \\ d_3 + i & \text{if } 2d_2 - n + 3 \le i \le n - 2 \end{cases}.$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 - 1 = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 2$, then $d_i = n + 1 + 2d_2 - n + 2 = 2d_2 + 3$ and $d'_i = d_i - 2 = 2d_2 + 1$.

$$S_1 = \frac{1}{2d_2} - \frac{1}{2d_2+3} = \frac{3}{2d_2(2d_2+3)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2 \cdot (d_2+1)} - 0 > 0.$$

6. $d_1 = \frac{2d_2+3}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } i = 2d_2 - n + 1 \text{ or } 2d_2 - n + 2 \\ d_3 + i & \text{if } 2d_2 - n + 3 \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 \\ d_3 + i & \text{if } 2d_2 - n + 3 \le i \le n - 2 \end{cases}$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 - 1 = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 2$, then $d_i = n + 1 + 2d_2 - n + 2 - 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2 + 1$.

$$S_1 = \frac{1}{2d_2} - \frac{1}{2d_2 + 2} = \frac{1}{2d_2(d_2 + 1)}$$

$$U(d_2) - U(d_2 + 1) = \frac{1}{d_2 \cdot (d_2 + 1)} - \frac{1}{2d_2 \cdot (d_2 + 1)} - 0 > 0$$

7. $d_1 \geq \frac{2d_2+4}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \leq i \leq 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \leq i \leq n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 + 2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \leq i \leq 3d_1 - n - 1 \\ d_3 + i & \text{if } 3d_1 - n \leq i \leq n - 2 \end{cases}$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 - 1 = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 2$, then $d_i = n + 1 + 2d_2 - n + 2 - 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2+1$.

For all other $i \neq 2d_2 - n + 1$ or $2d_2 - n + 2$, $d_i = d'_i$.

$$S_1 = \frac{1}{2d_2} - \frac{1}{2d_2 + 2}$$

$$U(d_2) - U(d_2 + 1) = \tfrac{1}{d_2 \cdot (d_2+1)} - (\tfrac{1}{2d_2} - \tfrac{1}{2d_2+2}) - 0 > 0.$$

Lemma 13: $U(d_2) \geq U(d_2+1)$ when $d_1 = \frac{2n-1}{3}$ or $\frac{2n}{3}$ or $\frac{2n+1}{3}$ and $\frac{2n+3}{3} \leq d_2 \leq n - 3$.

Proof. $r_1 \geq \frac{n+2}{3} > \frac{d_1}{2}$, $4 \leq r_2 \leq \frac{n}{3} < \frac{d_2}{2}$, and $3 \leq r'_2 = r_2 - 1 < \frac{d'_2}{2}$. By table B-29 in claim 7, $d_n = d'_n$ and $d_{n-1} = d'_{n-1}$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 + 2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le n - 2 \end{cases}$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 - 1 = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 2$, then $d_i = n + 1 + 2d_2 - n + 2 - 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2+1$.

For all other $i \neq 2d_2 - n + 1$ or $i \neq 2d_2 - n + 2$, $d_i = d'_i$.

$$S_1 = \frac{1}{2d_2} - \frac{1}{2d_2 + 2}$$

$$U(d_2) - U(d_2 + 1) = \tfrac{1}{d_2 \cdot (d_2+1)} - (\tfrac{1}{2d_2} - \tfrac{1}{2d_2+2}) - 0 > 0 .$$

Lemma 14: $U(d_2) \ge U(d_2+1)$ when $d_1 = \frac{2n-1}{3}$ and $d_2 = n - 2$.

Proof. $r_1 \ge \frac{d_1+3}{2}$, $r_2 = 3$, $d'_2 = d_2 + 1 = n - 1$, and $r'_2 = 2$. By claim 7, $d_n = d_{n-1} = 2d_3 - 1 = 2n + 1$ and $d'_n = d'_{n-1} = 2d_3 - 1 = 2n + 1$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n = n - 4 \\ d_3 + i - 1 & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n + 2 = n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-39.

TABLE B-39. d_i AND d'_i WHEN $d_1 = \frac{2n-1}{3}$ AND $d_2 = n - 2$

i	n -3	n -2
d_i	2n-3	2n-2
d'_i	2n-4	2n-3

$$S_1 = \frac{1}{2(n-1)(n-2)}$$

$$U(d_2) - U(d_2+1) = \tfrac{1}{d_2 \cdot (d_2+1)} - \tfrac{1}{2(n-1)(n-2)} - 0 > 0 .$$

Lemma 15: $U(d_2) \geq U(d_2+1)$ when $\frac{n+2}{2} \leq d_1 \leq \frac{2n-3}{3}$ and $d_2 = n-1$.

Proof. $r_1 \geq \frac{d_1+4}{2}$, $r_2 = 2$, and $r'_2 = 1$. By table B-29 in claim 7, $d_{n-1} = d_n = 2d_3 - 1 = 2n+1$, $d'_{n-1} = 2d_3 - 3 = 2n-1$, and $d'_n = 2d_3 = 2n+2$. Therefore,

$$S_2 = \frac{4n+1}{(2n-1)(2n+2)} - \frac{2}{2n+1}$$

1. $d_1 \leq \frac{2n-4}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 3d_1 - n \\ d_3 + i - 1 & \text{if } 3d_1 - n + 1 \leq i \leq 2d_2 - n - 1 = n - 3 \\ d_3 + i & \text{if } i = n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 3d_1 - n \\ d_3 + i - 1 & \text{if } 3d_1 - n + 1 \leq i \leq n - 2 \end{cases}$$

When $i = n-2$, $d_i = 2n-1$, and $d'_i = 2n-2$.

$$S_1 = \frac{1}{2(n-1)(2n-1)} .$$

$$U(d_2) - U(d_2+1) = \tfrac{1}{d_2 \cdot (d_2+1)} - \tfrac{1}{2(n-1)(2n-1)} - \left(\tfrac{4n+1}{(2n-1)(2n+2)} - \tfrac{2}{2n+1}\right) > 0 .$$

2. $d_1 = \frac{2n-3}{3}$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 3d_1 - n = n - 3 \\ d_3 + i & \text{if } i = n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 3d_1 - n = n - 3 \\ d_3 + i - 1 & \text{if } i = n - 2 \end{cases}$$

When $i = n-2$, $d_i = 2n-1$, and $d'_i = 2n-2$.

$$S_1 = \frac{1}{2(n-1)(2n-1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2(n-1)(2n-1)} - (\frac{4n+1}{(2n-1)(2n+2)} - \frac{2}{2n+1}) > 0.$$

Lemma 16: $U(d_2) \ge U(d_2+1)$ when $d_1 = d_2 = \frac{2n-3}{3}$.

Proof. $r_1 = r_2 = \frac{n+6}{3} = \frac{d_2+5}{2}$, $d'_2 = d_2 + 1 = \frac{2n}{3}$, and $r'_2 = \frac{n+3}{3} = \frac{d_2+2}{2}$. By claim 7, $d_{n-1} = 2d_3 - 1 = 2n+1$, $d_n = 2d_3 + 2 = 2n + 4$, $d'_{n-1} = 2d_3 - 1 = 2n + 1$, and $d'_n = 2d_3 + 2 = 2n + 4$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } i \le 2d_2 - n + 1 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le 3d_2 - n - 1 \\ d_3 + i + 1 & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } i \le 2d_2 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le 3d_2 - n - 1 = n - 4 \\ d_3 + i & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-40.

TABLE B-40. d_i AND d'_i WHEN $d_1 = d_2 = \frac{2n-3}{3}$

i	$2d_2 - n + 2$	n -3	n -2
d_i	$2d_2 + 2$	$2n$-1	$2n$
d'_i	$2d_2+1$	$2n$-2	$2n$-1

$$S_1 = (\frac{1}{2d_2+1} + \frac{1}{2n-2}) - (\frac{1}{2d_2+2} + \frac{1}{2n}) = \frac{1}{(2d_2+1)(2d_2+2)} + \frac{1}{2n(n-1)}.$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - (\frac{1}{(2d_2+1)(2d_2+2)} + \frac{1}{2n(n-1)}) - 0 > 0.$$

Lemma 17: $U(d_2) \geq U(d_2+1)$ when $d_1 = \frac{2n-3}{3}$ *or* $\frac{2n-2}{3}$ and $d_2 = d_1+1$.

Proof. First calculate S_1.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_1 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_1 - n + 3 = 2d_2 - n + 1 \leq i \leq 3d_1 - n - 1 = n - 4 \\ d_3 + i & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

$$d_i^{'} = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \leq i \leq 3d_1 - n - 1 = n - 4 \\ d_3 + i & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-41.

TABLE B-41. d_i AND d'_i WHEN $d_1 = \frac{2n-3}{3}$ or $\frac{2n-2}{3}$ AND $d_2 = d_1+1$

i	$2d_2 - n + 1$	$2d_2 - n + 2$
d_i	$2d_2 + 1$	$2d_2 + 2$
d'_i	$2d_2$	$2d_2 + 1$

$$S_1 = (\frac{1}{2d_2} + \frac{1}{2d_2+1}) - (\frac{1}{2d_2+1} + \frac{1}{2d_2+2}) = \frac{1}{2d_2(d_2+1)}.$$

Now calculate S_2.

1. $d_1 = \frac{2n-3}{3}$ and $d_2 = \frac{2n}{3}$.

   $r_1 = \frac{n+6}{3} = \frac{d_1+5}{2}$, $r_2 = \frac{n+3}{3} = \frac{d_2+2}{2}$, $d_2^{'} = d_2 + 1 = \frac{2n+3}{3}$, and $r_2^{'} = \frac{n}{3} = \frac{d_2^{'}-1}{2}$. By tables B-31 and B-29 in claim 7, $d_{n-1} = 2d_3 - 1 = 2n+1$, $d_n = 2d_3 + 2 = 2n + 4$, and $d'_{n-1} = d'_n = 2d_3 - 1 = 2n + 1$. Therefore,

$$S_2 = \frac{2}{2n+1} - (\frac{1}{2n+1} + \frac{1}{2n+4}) = \frac{3}{2(2n+1)(n+2)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - \frac{3}{(2n+1)(2n+4)} > 0 \ .$$

2. $d_1 = \frac{2n-2}{3}$ and $d_2 = \frac{2n+1}{3}$.

$r_1 = \frac{n+5}{3} = \frac{d_1+4}{2}$, $r_2 = \frac{n+2}{3} = \frac{d_2+1}{2}$, $d'_2 = d_2 + 1 = \frac{2n+4}{3}$, and $r'_2 = \frac{n-1}{3} = \frac{d'_2-2}{2}$. By table B-31 in claim 7, $d_{n-1} = 2d_3 - 1 = 2n + 1$, $d_n = 2d_3 + 2 = 2n + 4$, $d'_{n-1} = 2d_3 - 2 = 2n$, and $d'_n = 2d_3 + 2 = 2n + 4$. Therefore,

$$S_2 = (\frac{1}{2n} + \frac{1}{2n+4}) - (\frac{1}{2n+1} + \frac{1}{2n+4}) = \frac{1}{2n(2n+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - \frac{1}{2n(2n+1)} > 0.$$

Lemma 18: $U(d_2) \geq U(d_2+1)$ when $d_1 = d_2 = \frac{2n-2}{3}$.

Proof. $r_1 = r_2 = \frac{n+5}{3} = \frac{d_2+4}{2}$, $d'_2 = d_2 + 1 = \frac{2n+1}{3}$, and $r'_2 = \frac{n+2}{3} = \frac{d'_2+1}{2}$. By table B-31 in claim 7, $d_{n-1} = 2d_3 - 1 = 2n+1$, $d_n = 2d_3+2 = 2n+4$, $d'_{n-1} = 2d_3 - 2 = 2n$, and $d'_n = 2d_3+2 = 2n+4$. Therefore,

$$S_2 = (\frac{1}{2n} + \frac{1}{2n+4}) - (\frac{1}{2n+1} + \frac{1}{2n+4}) = \frac{1}{2n(2n+1)}$$

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } i \leq 2d_2 - n + 1 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \leq i \leq 3d_2 - n - 1 \\ d_3 + i + 1 & \text{if } i = n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } i \leq 2d_2 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \leq i \leq 3d_2 - n - 1 = n - 3 \\ d_3 + i & \text{if } i = n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in table B-42.

TABLE B-42. d_i AND d'_i WHEN $d_1 = d_2 = \frac{2n-2}{3}$

i	$2d_2 - n + 2$	n - 2
d_i	$2d_2 + 2$	$2n$
d'_i	$2d_2 + 1$	$2n$ -1

$$S_1 = (\frac{1}{2d_2+1} + \frac{1}{2n-1}) - (\frac{1}{2d_2+2} + \frac{1}{2n}) = \frac{1}{(2d_2+1)(2d_2+2)} + \frac{1}{2n(2n-1)}$$

$$U(d_2)-U(d_2+1)=\tfrac{1}{d_2\cdot(d_2+1)}-(\tfrac{1}{(2d_2+1)(2d_2+2)}+\tfrac{1}{2n(2n-1)})-\tfrac{1}{2n(2n+1)}>0\ .$$

Lemma 19: $U(d_2) \geq U(d_2+1)$ when $\frac{2n-2}{3} \leq d_1 \leq n-2$ and $d_2 = n-1$.

Proof.

$$d_i = \begin{cases} d_3+i-3 & \text{if } 3 \leq i \leq 2d_1-n+1 \\ d_3+i-2 & \text{if } 2d_1-n+2 \leq i \leq n-2 \end{cases}$$

$$d'_i = \begin{cases} d_3+i-3 & \text{if } 3 \leq i \leq 2d_1-n+1 \\ d_3+i-2 & \text{if } 2d_1-n+2 \leq i \leq n-2 \end{cases}$$

$$S_1 = 0$$

1. $d_1 = \frac{2n-2}{3}$

$r_1 = \frac{d_1+4}{2}$, $r_2 = 2$ and $r'_2 = 1$. By tables B-29 and B-30 in claim 7, one can get table B-43.

TABLE B-43. d_{n-1}, d_n, d'_{n-1}, AND d'_n WHEN $d_1 = \frac{2n-2}{3}$ AND $d_2 = n-1$

r_2	d_{n-1}	d_n	r'_2	d'_{n-1}	d'_n
2	$2n+1$	$2n+1$	1	$2n-1$	$2n+2$

$$S_2 = \frac{4n+1}{(2n-1)(2n+2)} - \frac{2}{2n+1}$$

$$U(d_2)-U(d_2+1)=\frac{1}{d_2\cdot(d_2+1)}-(\frac{4n+1}{(2n-1)(2n+2)}-\frac{2}{2n+1})-0>0\ .$$

2. $d_1 = \frac{2n-1}{3}$.

$r_1 = \frac{d_1+3}{2}$, r_2 =2, and r'_2 =1. By tables B-29 and B-30 in claim 7, one can get table B-44.

TABLE B-44. d_{n-1}, d_n, d'_{n-1}, AND d'_n WHEN $d_1 = \frac{2n-1}{3}$ AND $d_2 = n-1$

r_2	d_{n-1}	d_n	r'_2	d'_{n-1}	d'_n
2	$2n+1$	$2n+1$	1	$2n-2$	$2n+2$

$$S_2 = \frac{n}{(n-1)(n+1)} - \frac{2}{2n+1}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - (\frac{n}{(n-1)(n+1)} - \frac{2}{2n+1}) > 0 .$$

3. $d_1 = \frac{2n}{3}$.

$r_1 = \frac{d_1+2}{2}$, r_2 =2, and r'_2 = 1. By tables B-29 and B-30 in claim 7, one can get table B-45.

TABLE B-45. $d_{n\text{-}1}$, d_n, $d'_{n\text{-}1}$, AND d'_n WHEN $d_1 = \frac{2n}{3}$ AND $d_2 = n$-1

r_2	$d_{n\text{-}1}$	d_n	r'_2	$d'_{n\text{-}1}$	d'_n
2	2n-1	2n+2	1	2n-1	2n-1

$$S_2 = \frac{3}{(2n-1)(2n+2)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{3}{(2n-1)(2n+2)} - 0 > 0 .$$

4. $d_1 = \frac{2n+1}{3}$.

$r_1 = \frac{d_1+1}{2}$, r_2 =2, and r'_2 =1. By tables B-29 and B-30 in claim 7, one can get table B-46.

TABLE B-46. $d_{n\text{-}1}$, d_n, $d'_{n\text{-}1}$, AND d'_n WHEN $d_1 = \frac{2n+1}{3}$ AND $d_2 = n$-1

r_2	$d_{n\text{-}1}$	d_n	r'_2	$d'_{n\text{-}1}$	d'_n
2	2n	2n	1	2n-1	2n-1

$$S_2 = \frac{1}{n(2n-1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{n(2n-1)} > 0$$

5. $\frac{2n+2}{3} \le d_1 \le n-2$.

$3 \le r_1 \le \frac{d_1}{2}$, r_2 =2 and r'_2 = 1. By tables B-29 and B-30 in claim 7, one can get table B-47.

TABLE B-47. $d_{n\text{-}1}$, d_n, $d'_{n\text{-}1}$ AND d'_n WHEN $\frac{2n+2}{3} \le d_1 \le n-2$ AND $d_2 = n\text{-}1$

r_2	$d_{n\text{-}1}$	d_n	r'_2	$d'_{n\text{-}1}$	d'_n
2	$2n$	$2n$	1	$2n$-1	$2n$-1

$$S_2 = \frac{1}{n(2n-1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{n(2n-1)} > 0.$$

Lemma 20: $U(d_2) \ge U(d_2+1)$ when $d_1 = \frac{2n-1}{3}$ or $\frac{2n}{3}$ or $\frac{2n+1}{3}$ or $\frac{2n+2}{3}$ and $d_2 = d_1$.

Proof. First consider S_1.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } i \le 2d_2 - n + 1 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le 3d_2 - n - 1 = n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } i \le 2d_2 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le 3d_2 - n - 1 = n - 2 \end{cases}$$

If $i = 2d_2 - n + 2$, then $d_i = 2d_2+2$ and $d'_i = 2d_2+1$.

$$S_1 = \frac{1}{2d_2+1} + \frac{1}{2d_2-2} = \frac{1}{(2d_2+1)(2d_2+2)}$$

Now consider S_2.

1. $d_1 = d_2 = \frac{2n-1}{3}$.

$r_1 = r_2 = \frac{n+4}{3} = \frac{d_2+3}{2}$, $d'_2 = d_2 + 1 = \frac{2n+2}{3}$, and $r'_2 = \frac{n+1}{3} = \frac{d'_2}{2}$. By tables B-29 and B-30 in claim 7, $d_{n\text{-}1} = 2d_3\text{-}1 = 2n+1$, $d_n = 2d_3+2 = 2n+4$, and $d'_{n\text{-}1} = d'_n = 2d_3 \text{-}1 = 2n+1$. Therefore,

$$S_2 = \frac{2}{2n+1} - (\frac{1}{2n+1} + \frac{1}{2n+4}) = \frac{3}{(2n+4)(2n+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - (\frac{1}{(2d_2+1)(2d_2+2)}) - \frac{3}{(2n+4)(2n+1)} > 0.$$

2. $d_1 = d_2 = \frac{2n}{3}$.

$r_1 = r_2 = \frac{n+3}{3} = \frac{d_2+2}{2}$, $d'_2 = d_2 + 1 = \frac{2n+3}{3}$, and $r'_2 = \frac{n}{3} = \frac{d'_2-1}{2}$. By claim 7, $d_{n-1} = 2d_3 - 3 = 2n - 1$, $d_n = 2d_3 + 2 = 2n+4$, $d'_{n-1} = 2d_3 - 3 = 2n-1$, and $d'_n = 2d_3 = 2n+2$. Therefore,

$$S_2 = \left(\frac{1}{2n-1} + \frac{1}{2n+4}\right) - \left(\frac{1}{2n-1} + \frac{1}{2n+2}\right) = \frac{1}{2(n+2)(n+1)}.$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \left(\frac{1}{(2d_2+1)(2d_2+2)}\right) - \frac{1}{2(n+2)(n+1)} > 0.$$

3. $d_1 = d_2 = \frac{2n+1}{3}$.

$r_1 = r_2 = \frac{n+2}{3} = \frac{d_2+1}{2}$, $d'_2 = d_2 + 1 = \frac{2n+4}{3}$, and $r'_2 = \frac{n-1}{3} = \frac{d'_2-2}{2}$. By claim 7, $d_{n-1} = d_n = 2d_3 - 2 = 2n$, and $d'_{n-1} = d'_n = 2d_3 - 2 = 2n$. Therefore, $S_2 = 0$.

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \left(\frac{1}{(2d_2+1)(2d_2+2)}\right) - 0 > 0.$$

4. $d_1 = d_2 = \frac{2n+2}{3}$.

$r_1 = r_2 = \frac{n+1}{3} = \frac{d_2}{2}$, $d'_2 = d_2 + 1 = \frac{2n+5}{3}$, and $r'_2 = \frac{n-2}{3} = \frac{d'_2-3}{2}$. By claim 7, $d_{n-1} = d_n = 2d_3 - 2 = 2n$, $d'_{n-1} = d'_n = 2d_3 - 2 = 2n$. Therefore, $S_2 = 0$.

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \left(\frac{1}{(2d_2+1)(2d_2+2)}\right) - 0 > 0.$$

Lemma 21: $U(d_2) \geq U(d_2+1)$ when $d_1 = \frac{2n-1}{3}$ and $d_2 = \frac{2n+2}{3}$.

Proof. $r_1 = \frac{n+4}{3} = \frac{d_1+3}{2}$, $r_2 = \frac{n+1}{3} = \frac{d_2}{2}$, $d'_2 = d_2 + 1 = \frac{2n+5}{3}$, and $r'_2 = \frac{n-2}{3} = \frac{d'_2-3}{2}$. By claim 7, $d_{n-1} = d_n = 2d_3 - 1 = 2n+1$, and $d'_{n-1} = d'_n = 2d_3 - 1 = 2n+1$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_1 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_1 - n + 3 = 2d_2 - n + 1 \leq i \leq 3d_1 - n - 1 = n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \leq i \leq 3d_1 - n - 1 = n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in the table B-48.

TABLE B-48. d_i AND d'_i WHEN $d_1 = \frac{2n-1}{3}$ AND $d_2 = \frac{2n+2}{3}$

i	$2d_2 - n + 1$	$2d_2 - n + 2$
d_i	$2d_2 + 1$	$2d_2 + 2$
d'_i	$2d_2$	$2d_2+1$

$$S_1 = (\frac{1}{2d_2+1} + \frac{1}{2d_2+2}) - (\frac{1}{2d_2} + \frac{1}{2d_2+1}) = \frac{1}{2d_2(d_2+1)}.$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - 0 > 0.$$

Lemma 22: $U(d_2) \geq U(d_2+1)$ when $d_1 = \frac{2n}{3}$ and $d_2 = n-2$.

Proof. $r_1 = \frac{d_1+2}{2}$, $r_2 = 3$, $d'_2 = d_2 + 1 = n - 1$, and $r'_2 = 2$. By table B-29 in claim 7, $d_{n-1} = 2d_3 - 3 = 2n - 1$, $d_n = 2d_3 = 2n + 2$, $d'_{n-1} = 2d_3 - 3 = 2n-1$, and $d'_n = 2d_3 = 2n+2$. Therefore, S_2=0.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n = n - 4 \\ d_3 + i - 1 & \text{if } i = n - 3 \text{ or } n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n + 2 = n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in the table B-49.

TABLE B-49. d_i AND d'_i WHEN $d_1 = \frac{2n}{3}$ AND $d_2 = n-2$

i	n -3	n -2
d_i	$2n$ -3	$2n$ -2
d'_i	$2n$ -4	$2n$ -3

$$S_1 = \frac{1}{2(n-1)(n-2)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2(n-1)(n-2)} - 0 > 0.$$

Lemma 23: $U(d_2) \geq U(d_2+1)$ when $\frac{2n+1}{3} \leq d_1 \leq n-3$ and $d_2 = n-2$.

Proof. $4 \leq r_1 \leq \frac{d_1+1}{2}$, $r_2 = 3$, $d'_2 = d_2 + 1 = n-1$, and $r'_2 = 2$. By table B-29 in claim 7, $d_{n-1} = d_n = 2d_3 - 2 = 2n$ and $d'_{n-1} = d'_n = 2d_3 - 2 = 2n$. Therefore, S_2=0.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n = n - 4 \\ d_3 + i - 1 & \text{if } i = n-3 \text{ or } n-2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n + 2 = n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in the table B-50.

TABLE B-50. d_i AND d'_i WHEN $\frac{2n+1}{3} \leq d_1 \leq n-3$ AND $d_2 = n-2$

i	n -3	n -2
d_i	2n -3	2n -2
d'_i	2n -4	2n -3

$$S_1 = \frac{1}{2(n-1)(n-2)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2(n-1)(n-2)} - 0 > 0\,.$$

Lemma 24: $U(d_2) \geq U(d_2+1)$ when $d_1 = \frac{2n+2}{3}$ and $\frac{2n+5}{3} \leq d_2 \leq n-3$.

Proof. It can be shown that $r_1 = \frac{n+1}{3} = \frac{d_1}{2}$, $r_2 < \frac{d_2}{2}$, and $r'_2 < \frac{d'_2}{2}$. By claim 7, $d_{n-1} = d_n = 2d_3 - 2 = 2n$ and $d'_{n-1} = d'_n = 2d_3 - 2 = 2n$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \leq i \leq n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \leq i \leq 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \leq i \leq 2d_2 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \leq i \leq n - 2 \end{cases}$$

Based on the above expressions, the cases when d_i and d'_i take different values are listed in the table B-51.

TABLE B-51. d_i AND d'_i WHEN $d_1 = \frac{2n+2}{3}$ AND $\frac{2n+5}{3} \le d_2 \le n-3$

i	$2d_2 - n + 1$	$2d_2 - n + 2$
d_i	$2d_2 + 1$	$2d_2 + 2$
d'_i	$2d_2$	$2d_2+1$

$$S_1 = \frac{1}{2d_2(d_2+1)}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \frac{1}{2d_2(d_2+1)} - 0 > 0\,.$$

Lemma 25: $U(d_2) \ge U(d_2+1)$ when $\frac{2n+2}{3} < d_1 \le d_2 \le n-3$.

Proof. One can show that $3 \le r_1 < \frac{d_1}{2}$, $3 \le r_2 < \frac{d_2}{2}$ and $2 \le r'_2 < \frac{d_2}{2}$. By claim 7, $d_n = d_{n-1} = 2d_3$-2 = 2n and $d'_n = d'_n$-1 = 2d_3 -2 = 2n. Therefore, $S_2 = 0$.

First, suppose that $d_1 < d_2$. Then

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 1 \le i \le n - 2 \end{cases}$$

$$d'_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } 2d_1 - n + 2 \le i \le 2d_2 + 2 - n \\ d_3 + i - 1 & \text{if } 2d_2 - n + 3 \le i \le n - 2 \end{cases}$$

If $i = 2d_2 - n + 1$, then $d_i = n + 1 + 2d_2 - n + 1 - 1 = 2d_2 + 1$ and $d'_i = d_i - 1 = 2d_2$.

If $i = 2d_2 - n + 2$, then $d_i = n + 1 + 2d_2 - n + 2 - 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2+1$.

For all other $i \ne 2d_2 - n + 1$ or $i \ne 2d_2 - n + 2$, $d_i = d'_i$.

$$S_1 = \frac{1}{2d_2} - \frac{1}{2d_2+2}$$

$$U(d_2) - U(d_2+1) = \frac{1}{d_2 \cdot (d_2+1)} - \left(\frac{1}{2d_2} - \frac{1}{2d_2+2}\right) - 0 > 0\,.$$

Now, consider the case where $d_1 = d_2$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 1 & \text{if } 2d_1 - n + 2 \le i \le n - 2 \end{cases}$$

$$d_i^{'} = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 \\ d_3 + i - 2 & \text{if } i = 2d_1 - n + 2 \\ d_3 + i - 1 & \text{if } 2d_1 - n + 3 \le i \le n - 2 \end{cases}$$

If $i = 2d_2 - n + 2$, then $d_i = n + 1 + 2d_2 - n + 2 - 1 = 2d_2 + 2$ and $d'_i = d_i - 1 = 2d_2 + 1$.

For all other $i \neq 2d_2 - n + 2$, $d_i = d'_i$.

$$S_1 = \frac{1}{2d_1 + 1} - \frac{1}{2d_1 + 2} = \frac{1}{(2d_2 + 1)(2d_2 + 2)}$$

$$U(d_2) - U(d_2 + 1) = \frac{1}{d_2 \cdot (d_2 + 1)} - \frac{1}{(2d_2 + 1)(2d_2 + 2)} - 0 > 0\,.$$

Lemma 26: $U(d_2) \ge U(d_2+1)$ when $d_1 = d_2 = n$-2.

Proof. $r_1 = r_2 = 3$ and $r'_2 = 2$. By claim 7, $d_n = d_{n-1} = 2d_3$-2 $= 2n$ and $d'_n = d'_n$-1 $= 2d_3$ -2 $= 2n$. Therefore, $S_2 = 0$.

$$d_i = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 = n - 3 \\ d_3 + i - 1 & \text{if } i = n - 2 \end{cases}$$

$$d_i^{'} = \begin{cases} d_3 + i - 3 & \text{if } 3 \le i \le 2d_1 - n + 1 = n - 3 \\ d_3 + i - 2 & \text{if } i = n - 2 \end{cases}$$

If $i = n - 2$, then $d_i = 2n - 2$ and $d'_I = 2n - 3$.

$$S_1 = \frac{1}{2(n-1)(2n-3)}\,.$$

$$U(d_2) - U(d_2 + 1) = \tfrac{1}{d_2 \cdot (d_2 + 1)} - \tfrac{1}{2(n-1)(2n-3)} - 0 > 0\,.$$

Lemma 27: $U(d_2) \geq U(d_2{+}1)$ when $d_1 = d_2 = n$-1.

Proof. $r_1 = 3$, $r_2 = 2$ and $r'_2 = 1$. The values of d_{n-1}, d_n , d'_{n-1}, and d'_n are listed in table B-52.

TABLE B-52. $d_{n\text{-}1}$, d_n, $d'_{n\text{-}1}$, AND d'_n WHEN $d_1 = d_2 = n$-1

r_2	$d_{n\text{-}1}$	d_n	r'_2	$d'_{n\text{-}1}$	d'_n
2	$2n$-3	$2n$+1	1	$2n$-3	$2n$

$$S_2 = \frac{1}{2n(2n+1)} .$$

For $i \leq n$ - 2, $d_i = d_3 + i$ –3 and $d'_i = d_3 + i$ -3. $S_1 = 0$.

$$U(d_2) - U(d_2+1) = \tfrac{1}{d_2 \cdot (d_2+1)} - 0 - \tfrac{1}{2n(2n+1)} > 0 .$$

B.6 UNIT-EXECUTION-TIME TASK SYSTEMS: LIMITED PRIORITY LEVELS.

It was planned that the same problem be studied as in section B.5, with the assumption that each processor has *m* priority levels. Unfortunately, because of limited time, no significant progress was made. It was, however, conjectured that the same threshold holds for computing systems with *m* priority levels, where $m < n$.

B.7 REFERENCES.

B-1. Labetoulle, J., “Some Theorems on Real-Time Scheduling,” in *Computer Architecture and Networks*, E. Gelenbe and R. Mahl, eds., North-Holland, Amsterdam, 1974.

B-2. Liu, C.L. and Layland, J.W., “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *J. of ACM*, Vol. 20, 1973, pp. 46-61.

B-3. Leung, J.Y-*T*. and Merrill, M.L., “A Note on Preemptive Scheduling of Periodic, Real-Time Tasks,” *Information Processing Letters*, Vol. 11, 1980, pp. 115-118.

B-4. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

B-5. Leung, J.Y-*T*., “A New Algorithm for Scheduling Periodic, Real-Time Tasks,” *Algorithmica*, Vol. 4, 1989, pp. 209-219.

B-6. Lawler, E.L. and Martel, C.U., “Scheduling Periodically Occurring Tasks on Multiple Processors,” *Information Processing Letters*, Vol. 12, 1981, pp. 9-12.

B-7. Leung, J.Y-*T*. and Whitehead, J., “On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks,” *Performance Evaluation*, Vol. 2, 1982, pp. 237-250.

B-8. Coffman, Jr., E.G., Garey, M.R., and Johnson, D.S., “Approximation Algorithms for Bin Packing: A Survey,” in *Approximation Algorithms for NP-hard Problems*, D. Hochbaum ed., PWS Publishing Company, 1996.

B-9. Dhall, S.K. and Liu, C.L., “On a Real-Time Scheduling Problem,” *Operations Research,* Vol. 26, 1978, pp. 127-140.

B-10. Liu, J.W.S., *Real-Time Systems*, Prentice Hall, New Jersey, 2000.

APPENDIX C—THE IMPLEMENTATION OF THE ALGORITHM DM-LPL

```
/******************
    greedy-single.h
*********************/
#ifndef _GREEDY_SINGLE_H_
#define _GREEDY_SINGLE_H_

#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <limits.h>

#define INFINITY  INT_MAX

/*Priority 1 is the highest*/
typedef struct Task{
int exe; int deadline; int period; int priority;
} Task;
int greedy_single(int num_task, int  num_priority, Task *task_sys);
#endif


/******************
greedy-single.c
Function:
int greedy-single(int num_task, int  num_priority, Task *task_sys)
input:
    num_task: number of tasks
    num_priority:  number-of-prioritie levels
    task_sys:   pointer to the tasks array
output: The priority assignment to tasks
Algorithm:
    1.  sort the tasks in non-decreasing order of deadline
    2.  starting with priority level 1 and the first task, repeat for each task:

        2-1: try to assign the current task to the current priority level
        2-2: if 2-1 fails, if there is no task assigned the current priority level,
                         then the task system is not schedulable, return  '-1'
                        else if there are available priorities,
                          increase the current priority level by ‘1’
                      else
                          there is not enough priorities, return  ‘0’

        2-3:  if 2-1 succeeds, let the next task be the current task.

    3.   all tasks are assigned priorities, return ‘1’

    Hairong Zhao   07/06/2003

*****************/


#include “greedy-single.h”

/*****************************************************/

int compare( const void *task1, const void *task2);
/* Compare deadlines: return value = (deadline of task1  - deadline of task2)  **/
/*****************************************************/


/******************Main program **********************/
int greedy_single(int num_task, int  num_priority, Task *task_sys)
{

    int i;
    int return_val;
    int cur_priority;
    int cur_exe_sum;
    int cur_first_task;

/*************************************/
/**   Sort according to deadline    ***/
/*************************************/

    qsort((void *)task_sys, num_task, sizeof(Task), compare);

/*************************************/
/********** assign priority in a greedy way **********/
/*******************************************/

    cur_priority = 1;
```

```
/* the task with the smallest deadline in the current level */
    cur_first_task = 0 ;
    task_sys[cur_first_task].priority = cur_priority;
    /* assign the first task with highest priority */
    cur_exe_sum = task_sys[cur_first_task].exe ;
/* the total execution time of the tasks in the current priority level */
    i=1; /* assign priority to task 2,... n */
    while (i < num_task){
        int time;
        /* try to assign current task to current priority */
        task_sys[i].priority = INFINITY; /* initialize */
        cur_exe_sum = cur_exe_sum + task_sys[i].exe;

/* verify whether the smallest task in this priority is still schedulable */
        for(time= cur_exe_sum; time <= task_sys[cur_first_task].deadline; time ++){
            int requests = cur_exe_sum;
            int j;
            for(j = 0; j < i; j++){
               if(task_sys[j].priority < cur_priority)
               requests = requests + task_sys[j].exe * ceil(time*1.0/task_sys[j].period);
            }
            if(time == requests){
            task_sys[i].priority = cur_priority;
            break;
            }
        }

        if(task_sys[i].priority == INFINITY){
            /* task i can't be assigned to the current level */
            if(i== cur_first_task)
```

```
                break;   /* i is the first task in the current level. So the tasks are not schedulable. */
            if(cur_priority < num_priority) {  /* try to assign to the next level*/
                cur_priority = cur_priority +1;
                cur_first_task = i;
                cur_exe_sum = 0;
            }
            else          /* not enough priority level */
                break;
        }
        else{ /* assign the next task */
            i=i+1;
        }
    }

    if(i < num_task) {
         if(i== cur_first_task)
              return_val = -1; /* not schedulable */
         else
              return_val = 0; /* not enough priority levels */
    }
    else
             return_val = 1;  /* all tasks are assigned priorities. */

    return (return_val);

}

int compare( const void *task1, const void *task2)

{
return (((Task *)task1)->deadline - ((Task * )task2)->deadline);

}
```