Abstract

Title of Dissertation: Scheduling and Allocation in Multiprocessor Systems

Sheng-Tzong Cheng, Doctor of Philosophy, 1995

Dissertation directed by: Professor Ashok K. Agrawala
Department of Computer Science

The problem of allocation has always been one of the fundamental issues of building applications in multiprocessor systems. For real-time applications, the allocation problem should directly address the issues of task and communication scheduling. In this context, the allocation of tasks has to fully utilize the available processors and the scheduling of tasks has to meet the specified timing constraints. Clearly, the execution of tasks under the allocation and schedule has to satisfy the precedence, resources, and synchronization constraints.

Traditionally time constraints for real-time tasks have been specified in terms of ready time and deadlines. Many application tasks have relative timing constraints in which the constraints for the execution of a task are defined in terms of the actual execution instances of prior tasks. In this dissertation we consider the allocation and scheduling problem of the periodic tasks with relative timing requirements. We take a time-based scheduling approach to generate a multiprocessor schedule for a set of periodic tasks. A simulated annealing algorithm is developed as the overall search algorithm for a feasible solution. Our results show that the algorithm performs well and finds feasible allocation and scheduling.

We also investigate how to exploit the replication technique to increase the schedulability and performance of the systems. In this dissertation, we adopt the computation model in which each

task may have more than one copy and a task may start its execution after receiving necessary data from a copy of each of its predecessors. Based on this model, replication techniques are developed to increase the schedulability of the applications in real-time systems and to reduce the execution cost of the applications in non-real-time systems.

Scheduling and Allocation in Multiprocessor Systems

by

Sheng-Tzong Cheng

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1995

Advisory Committee:

Professor Ashok K. Agrawala, Chairman/Advisor
Professor Satish Tripathi
Associate Professor Joel Saltz
Associate Professor Der-Chen Chang
Assistant Professor Jeff Hollingsworth

Dedication

To my parents and my wife

Contents

List of Tables

List of Figures

Scheduling and Allocation in Multiprocessor Systems

Sheng-Tzong Cheng

April 24, 1995

This comment page is not part of the dissertation.

Typeset by $\LaTeX$ using the `dissertation` style by Pablo A. Straub, University of Maryland.

Chapter 1

Introduction

Multiprocessor systems offer the computing power that can be brought to bear on the development and applications of the embedded and/or controller systems. In these systems, the execution of tasks has to be carried out in a temporally determinate manner in that the timing constraints as well as the semantics of the specifications have to be satisfied. In order to build applications on these systems, a crucial problem is the allocation and scheduling of tasks. Primarily, the purpose of task allocation is to distribute the total workload over the available resources and processors in the system, and the purpose of scheduling is to provide the execution start times for tasks to satisfy the given timing constraints. The problem addressed in the dissertation is the problem of nonpreemptive task allocation and scheduling in multiprocessor systems.

Various architectures of multiprocessor systems have been proposed and implemented over the years. A primary distinguishing feature of these architectures is the approach used for memory access and communications. For example, systems with uniform memory access, non-uniform memory access, shared memory, non-shared memory, or loosely-coupled connection using the communications network, have been built. We consider the processors and communications network as the most important resources for embedded systems. The framework that we are addressing in this dissertation does not depend on any specific architecture. However, as the architecture impacts the inter-task communications, the scheduling of communications has to be taken into account as presented in Chapters 4 and 5. Note in this dissertation that we are not considering the allocation problem of multiprocessors to a single task. The allocation problem usually addressed for high preformance computing applications such as LU decomposition is not considered in this dissertation.

The objectives of the allocation problem depend on the applications which are to be built on the systems. Two different kinds of applications are considered in this dissertation; real-time and non-real-time. The tasks of real-time applications are characterized by the activity frequency (e.g. periodic, aperiodic, or sporadic), computation requirement (e.g. worst-case execution time or bounds on the execution time), and the timing constraints (e.g. relative or absolute). On the other hand, the execution of tasks of non-real-time applications is characterized by the cost function. We assume the cost function for a task or a communication depends on the execution time of a task, the transit time of the messages transferred between two tasks, the financial or billing cost, some measure of the resource usage, degree of reliability, failure rate, etc. Informally, the allocation and scheduling problem can be stated as the problem of allocating and scheduling tasks of applications among processors to achieve some objective under defined constraints.

1.1 Motivation

Our research is motivated by the need to develop allocation and scheduling techniques and algorithms to improve the schedulability and performance of multiprocessor systems. Furthermore, the emergence of new timing requirements from the real-time applications is generating interest in new scheduling problems that were not addressed in the past. In the following sections, the motivating factors behind our research are presented.

1.1.1 Pre-run-time versus Run-time Scheduling

While run-time scheduling has the virtues of dynamic reconfiguration and high processor utilization [SK91], the goal of satisfying the critical timing constraints can not be guaranteed. Xu and Parnas [XP91] pointed out that

> "the problem of satisfying timing constraints in a hard-real-time system is inherently a deterministic problem."

One feasible approach to solve such a deterministic problem is to schedule every task *a priori*. Further, they observed that the bulk of the computations are performed by periodic tasks. In this dissertation we consider the pre-run-time scheduling discipline. In particular, we concentrate on the periodic task systems. [1]

[1] As for the scheduling of aperiodic and sporadic tasks, it can be incorporated into the framework in two possible ways: pre-run-time or run-time. However, no guarantee can be made for the run-time scheduling.

1.1.2 Relative and Absolute Timing Constraints

Research on real-time scheduling has focused on absolute timing constraints on the tasks. Each task is associated with a ready time and a deadline. These timing constraints impose a fixed time interval in which a task may be executed. Neither inter-task dependencies nor temporal relationships between two instances of a periodic task can be specified.

While the absolute timing constraints specify the temporal requirements in terms of the offsets from the period, the relative timing constraints are usually enforced on the task start time and finish time. For example,

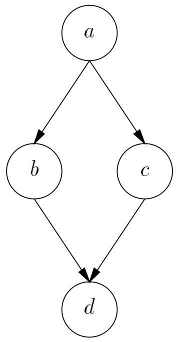
1. The consecutive executions of task 1 are separated at least 5 ms and at most 10 ms.

2. If task 2 sends a message to task 3, then start task 3 not later than 10 ms after the completion of task 2.

We examine the contemporary real-time applications and identify a class of relative timing constraints [CA94, CA95]. In this dissertation we combine the class of relative timing constraints and the absolute timing constraints into a hybrid timing model. We consider the task system with the hybrid timing constraints and present a solution to the assignment and scheduling problem.

1.1.3 Replication or Assignment Problem

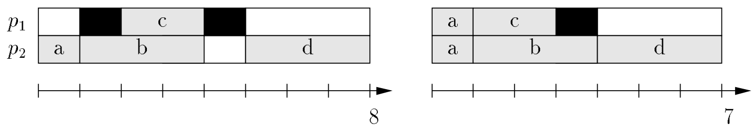
One way of performing the allocation of tasks to different processors is to retain a single copy of each task and run it at one and only one place. The allocation problem, in this case, is referred to as the **assignment problem**. However, with the availability of resources in multiprocessor systems, it is quite feasible to have multiple copies of a task, which execute concurrently on different processors. This approach to the replication has been used in the literature for the purpose of fault tolerance. We explore the replication to improve the schedulability in the real-time systems and performance in the non-real-time systems respectively. An example illustrating this point is shown in Figure 1.1 in which the number of processors in the system is two and the computation times for tasks are listed. We assume the ready time for the task is 0 and the deadline is 7. The inter-processor communication (IPC) between p_1 and p_2 for any two tasks takes one unit and the local communication time on the same processor is zero. If task a is assigned to processor p_1 only, then the minimum finish time is 8 as shown in Gantt chart 1. The dark area in Gantt charts indicates the IPC. If task a is replicated to processors p_1 and p_2, then we see from Gantt chart 2 that the minimum completion time is 7. As depicted in Figure 1.1, the replication of task a on processor 1 eliminates the IPC between tasks



(a) A task graph

$\tilde{v}$ vectors	processor 1	processor 2
task a on	1	1
task b on	3	3
task c on	2	2
task d on	3	5

(b) Computation Vectors



(c) Gantt Chart 1

(d) Gantt Chart 2

Figure 1.1: An illustrative example about how the replication can increase the schedulability

a and c. It enables the earlier start of the execution of task c, increases the degree of schedulability and makes the task meet its deadline.

1.2 Our Approach

1.2.1 Assignment Problem in Real-Time Systems

For the hard real-time applications, such as avionics systems and nuclear power systems, the approach to guarantee the critical timing constraints is to allocate and schedule tasks *a priori*. We take a global view of the system in which the allocation problem directly addresses the scheduling of processors and communications resources. We apply the simulated annealing technique to find a static allocation and a time-based scheduling approach to find a feasible schedule.

Even though we assume that the execution of periodic tasks continues indefinitely, when scheduling a set of such tasks we consider the schedule for a **scheduling frame** which can be repeated indefinitely. Typically the length of the scheduling frame will be the least common multiple (LCM) of the periods of the tasks in the set. The LCM period includes several task instances of each task depending on the task period. We define the **scheduling window** for a task instance to be the time interval in which the instance can start without violating the timing constraints. Unlike the absolute timing model in which the scheduling window for each instance is a fixed interval, the hybrid timing constraints impose a rather complicated and dynamical restriction on the scheduling. In this dissertation we apply the timing constraints analysis first to derive the scheduling window for each task instance.

We apply the simulated annealing technique as the overall search algorithm for a feasible solution. For each solution generated by the simulated annealing technique task instances, based on their scheduling windows, are scheduled one by one using the time-based non-preemptive approach. The outcome of the scheduling is a multiprocessor schedule, also called the **calendar**, which assigns each task to a processor and maintains the start time and finish time for each task instance. The calendar is verified to see if all timing constraints are satisfied. If yes, the problem is solved and the calendar can be repeatedly invoked throughout the execution of the applications. Otherwise, another solution point is generated by using a neighbor finding strategy of the simulated annealing technique. The above process repeats again until a feasible calendar is found or a stopping criteria is satisfied.

1.2.2 Replication Problem in Real-Time Systems

Traditionally, the main purpose of replicating a task on multiple processors is to increase the degree of fault tolerance. If some processors in the system fail, the application may still survive using other replicas, or copies. In such a computation model, a task has to communicate with multiple replicas of other tasks. As a consequence, replication of tasks may delay the execution of tasks due to the extra inter-task communications. Such delay may cause a task to miss its deadline, and degrade the schedulability of the system.

In the dissertation, we adopt a computation model in which the replication of a task is not for the sake of fault tolerance but for schedulability-oriented objectives. We call it the **1-out-of-**

n model [CHA94]. In our model, each task may have more than one copy and it may start its execution if it receives necessary data from any copy of a preceding task. As a consequence, the replicas of a task on different processors can be used to reduce the inter-processor communication (IPC), and to fully utilize the available processors in the system. Such replication may lead to an improved schedulability.

To solve the schedulability-oriented replication problem, we develop a replication technique and embed the technique in a simulated annealing algorithm. The function of the technique consists of two phases, namely (1) identifying bottleneck IPCs and (2) eliminating bottleneck IPCs. We define the bottleneck IPCs to be the communications which delay the execution of the tasks and make the schedule infeasible. Once the bottleneck IPCs are identified, we replicate the sender tasks to eliminate such IPCs. Experimental results show that such replication leads to a higher degree of schedulability.

1.2.3 Replication Problem in Non-Real-Time Systems

In the last few years, a significant amount of research effort has been directed towards the assignment problem [Sto77, AR80a, AR80b, Bok81, Bok87, CHLE80, Lo88, MLT82, MM89, PK84, ST85, Tow86]. Based on the different objectives, the complexity of the problem on various architectures has been established and some solutions have been proposed. However, in the replication problem, very little work has been done. We note that, in general, the replication problem is much more difficult than the assignment problem. In this dissertation we take the minimization of the total cost of task execution and IPC as the objective of the allocation problem in the non-real-time systems. We call it the performance-oriented objective.

In the performance-oriented assignment problem, polynomial-time algorithms exist for special cases, such as tree-structure [Bok81] and series-parallel [Tow86] task graphs. Our research [CA93] represents one of the early attempts at finding special cases for which polynomial time solutions exist for performance-oriented replication problems. The class of applications we consider is computation-intensive applications in which the execution cost of a task is greater than its communication cost. Such applications can be found in many fields, such as digital signal processing, weather forecasting, game searching, etc. We take series-parallel task graphs into consideration. We call this special case the **Minimum Cost Replication Problem of Series-Parallel graphs** (MCRP-SP) for computation-intensive applications.

When an application is computation-intensive, and the objective is to minimize the sum of execution costs and inter-task communication costs, we find that the optimal replication can be done by considering only the replication of the **forkers** in the SP task graph. For all the tasks which do not represent forkers in the task graph, the shortest path algorithms or network flow algorithms can still be applied. Overall, we prove that the MCRP-SP for computation-intensive applications is still NP-complete by mapping the VERTEX COVER problem [GJ79] to this problem, We present a branch-and-bound algorithm to solve it. The worst-case complexity of the solution is $O(n^2 2^n M)$, where n is the number of processors in the system and M is the number of tasks in the graph. Note that the complexity of the algorithm is a linear function of M.

We also consider an approximation technique to solve the problem in polynomial time. Given a forker task a with K successors in the SP graph, the method tries to allocate a to processors based on iterative selection. When the near-optimal solution for i successors is obtained, the method finds the near-optimal solution for $i + 1$ successors until all successors are taken into account. The complexity of the approximation algorithm is $O(n^2 K^2)$.

1.3 Summary of Contributions

The major contributions of this dissertation are the following.

- We identify a hybrid timing model which combines the absolute and relative timing constraints. Real example applications which adopt the timing model are described. Algorithms and techniques are developed to solve the allocation and scheduling problem for the task systems with this timing model.

- We develop a resource reservation technique for the allocation and scheduling problem in multi-resource systems. Each resource in the system is represented by a linked list in which one element of the list corresponds to the scheduling of one task instance. The technique consists of sliding the existent time slots to create a big-enough time slot and scheduling the task instance to the new time slot.

- We consider the schedulability-oriented replication problem of a set of periodic real-time tasks. A replication technique is developed and embedded in a simulated annealing algorithm. Experimental results show that such replication may lead to a higher degree of schedulability.

- We consider the performance-oriented replication problem of series-parallel task graphs. The objective of the problem is to minimize the total cost of task execution and inter-task communication. We present a branch-and-bound method to find an optimal solution as well as an approximation approach for suboptimal solutions. The numerical results show that such replication may lead to a lower cost than the optimal assignment problem.

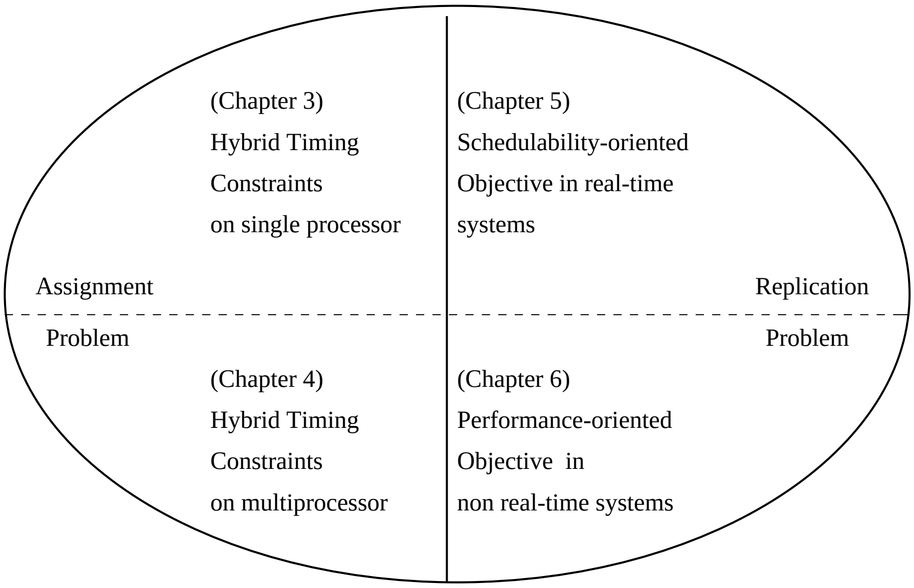


Figure 1.2: The organization of the dissertation

1.4 Organization

The dissertation is organized as follows. Chapter 2 reviews related work. The model of hybrid timing constraints is introduced and analyzed in Chapter3. We first consider the assignment and scheduling problem of task systems with the hybrid timing model on single processor in Chapter 3. Then, in Chapter 4, the results obtained from the single-processor case are extended to the multi-processor case. In Chapter 5, we consider the replication problem with the schedulability-oriented objective on real-time systems. For the non-real-time systems, we consider the replication problem with the performance-oriented objective in Chapter 6. Finally, concluding remarks are presented in Chapter 7. A pictorial outline of Chapters 3 to 6 is given in Figure 1.2.

Chapter 2

Related Work

In order to put our work in context, in this chapter we review the prior work reported in the literature. We discuss the real-time scheduling theory and algorithms, and allocation techniques in non-real-time and real-time systems respectively.

2.1 Real-Time Scheduling

While extensive literature exists in the area of scheduling, we discuss here some of the recent relevant developments in the areas of *time-based scheduling*, *static priority-based scheduling*, *dynamic priority-based scheduling*, and *interval-constrainted scheduling*.

2.1.1 Time-Based Scheduling

In time-based scheduling, a schedule is constructed off-line and is used throughout the execution of the applications. The schedule, also called the *calendar*, maps each task to time lines. The time information is used at run-time to dispatch the tasks.

The problem of finding an optimal or feasible schedule is basically a combinatorial search problem. Time-based techniques have been proposed and developed to solve the problem with various task characteristics. Among these techniques, the branch-and-bound method [XP90] is applied to find an optimal schedule while heuristic search [Ram90, ZRS87] is used to find a feasible schedule. These techniques organize the search space of the problem as a search tree in which the nodes in the tree represent potential solutions. The complexity of searching through such a search tree to find an optimal or feasible solution is usually enormous. Hence, search space reduction becomes a primary concern. While the performance of the branch-and-bound methods depends on estimates of the bounds used to prune the useless subtree and avoid unnecessary searching, heuristic approaches usually use *ad-hoc* search methods.

Temporal analysis is another way to reduce the search space. Yuan and Agrawala [YSA94] consider a decomposition approach in which the set of tasks is decomposed into several independent subsets using *leading* and *strongly leading* relationships. It has been shown that, when these relationships are used for decomposition, the scheduling of one subset precedes that of other subsets. The complexity involved in scheduling a subset of tasks can be significantly reduced compared to that for the whole set. Further, Saksena and Agrawala [SA93] consider the search space pruning by using a pair-wise temporal analysis technique. The technique is designed to extract as much information regarding the scheduling of a pair of tasks as can be extracted by examining the temporal information about that pair. The information can then be used to reduce the complexity of the scheduling problem.

2.1.2 Static Priority-Based Scheduling

In the priority-based scheduling, no explicit schedule is constructed. Instead, at the run time, the ready task with the highest priority is executed. If the priority assignment of tasks is done statically so that it does not change during the execution, the scheduling is called static priority-based scheduling.

Rate-monotonic scheduling [LL73](RM), which is known to be the optimal assignment among the static priority schemes, assigns the priorities to tasks based on their periods; the smaller the period, the higher the priority. RM primarily considers periodic tasks and the schedulability analysis of RM has also been established [LL73].

When tasks have synchronization requirements, it has been shown that some problems such as priority inversion can occur. The priority inversion problem occurs when a high-priority task is blocked by a low-priority task which holds the lock to a critical session. As a consequence, the high-priority task may suffer an unbounded blocking time and miss its deadline. Sha *et al.*[SRL90, RSL89] propose the priority inheritance protocol to solve the problem. The basic idea of priority inheritance is to allow a low-priority task to inherit the priority of the higher priority task and avoid further preemption from other tasks. Based on the priority inheritance protocol, worst-case blocking time has been computed and schedulability of the task systems has been established.

2.1.3 Dynamic Priority-Based Scheduling

In dynamic priority-based scheduling, the priority assignment of tasks changes during the execution of tasks. The Earliest-Deadline-First (EDF) algorithm [LL73], which assigns the priority of tasks based on their deadlines, has been shown to reach a 100% CPU utilization bound. The Minimum-Laxity-First (MLF) algorithm, which dynamically assigns the highest priority to the task with the minimum laxity, also has a CPU utilization bound of 100% for a set of independent tasks where the laxity of a task at time t is defined as (deadline $- \ t \ - \ e_t$) in which e_t is the execution time of task t.

Scheduling Schemes which combine static and dynamic priority scheduling include the Maximum-Urgency-First (MUF) algorithm [SK91]. MUF consists of two parts. The first part is the assignment of the criticality and user priority of a task based on the period of a task and the overall CPU utilization. This part is done once statically. The second part selects a task dynamically based on the criticality, laxity, and user priority. This part is activated whenever a task becomes ready to run. It has been shown that the MUF algorithm can be used to predictably schedule dynamically changing systems.

2.1.4 Interval-Constrainted Scheduling

A model of real-time task systems with temporal interval constraints has been proposed by Han and Lin [HL92b, HL89, HL92a]. In their work, the timing constraints on tasks are expressed by the temporal constraints on consecutive executions of a task and/or executions between two tasks. In the paper [HL92a], the scheduling between two consecutive executions of a task must be separated by at least a given temporal interval. Han and Lin consider the preemptive scheduling by transferring the scheduling problem to the *pin-wheel* problem [HMR$^+$89], and apply the known solution to the pin-wheel problem to solve the interval-constrainted scheduling problem. They further consider the interval constraints on the inter-task case in which tasks have unit execution times [HL92b].

Parametric Scheduling [GPS] is proposed to consider the interval-constrainted scheduling on a single processor in which the exact execution time of a task is unknown *a priori* and has lower and upper bounds. Parametric scheduling consists of two components. Given a total order among tasks, the timing constraints are analyzed to produce a *parametric calendar* by the off-line component. The entries in the parametric calendar consist of arithmetic expressions which depend on parameters such as the bounds and the actual start and finish time of tasks. Note that the values of some of the parameters are not available until the start or finish of a preceding task. When these values become available, the on-line component computes the value for the expression to determine the time when a task may be scheduled.

2.2 Allocation Problem

The allocation problem can be stated formally as the problem of allocating tasks of an application among processors to achieve some objective, under defined constraints. For example, the performance-oriented objective may be to minimize the sum of execution costs and inter-task communication costs, while the reliability-oriented objective could be to maximize the system reliability.

Based on different objectives, the literature on the allocation problem can be classified into three categories; *performance-oriented allocation*, *reliability-oriented allocation* and *schedulability-oriented allocation*. In each category, the allocation strategy can be *static* or *dynamic*. In static allocation all tasks are allocated before the application starts to execute. In dynamic allocation the tasks are allocated/reallocated at the run time based on the system conditions. In this section, we present significant results in these three categories.

2.2.1 Performance-Oriented Allocation

The performance-oriented allocation problem arises when the objective is to minimize (or optimize) a performance-oriented cost function. The strategy chosen to carry out the allocation depends on the performance-oriented cost function. Any values which are measurable in the system can be used as performance indices. For the static allocation problem, the performance indices mostly used are [SSA89].

- **Sum of execution costs and inter-task communication costs**: the total cost incurred by executing the application on a multiprocessor system.
- **Turnaround time**: the sum of execution time and inter- task communication time spent in a bottleneck processor. A bottleneck processor is one which has the maximum execution and communication time over all processors.
- **Inter-processor communication cost**: the sum of communication costs incurred by the tasks resident at different processors.
- **Load Balancing**: the variance of processor utilizations. The less the variance, the better the load balance.

For dynamic allocation problems, the commonly used performance-oriented objectives are:

- **Inter-processor communication cost** and
- **Load Balancing**.

Static Assignment

Some researchers have considered the static assignment problems in which the objective function is to minimize the sum of execution costs and inter-task communication costs. Stone [Sto77] proposes a solution for task assignment of a system with two processors. The 2-processor task assignment problem is transformed into a network flow problem, and then the network flow algorithm is used to solve the problem in polynomial time. In general, given an application with M tasks and n processors, the assignment problem is formally proven to be NP-complete [MM89]. Hence, several researchers focus on the special cases of the general assignment problem. Bokhari [Bok81] considers the particular case where the task graph is a tree structure. A shortest path algorithm [Bok87] is used to solve the problem in $O(n^2M)$ time. Towsley [Tow86] considers the series-parallel task graph and presents an optimal solution whose complexity is $O(n^3M)$. Sagar [SSA89] surveys other sub-optimal techniques which exploit the shortest path approach.

Branch-and-bound methods and heuristic algorithms have been proposed for the general allocation problem. Ma *et al.* [MLT82] exploit a branch-and-bound method in which the allocation constraints are taken into account. When a solution is generated by the method, the solution is checked first to see whether the allocation constraints are satisfied or not. If yes, the cost of the solution is evaluated; otherwise, the solution is discarded and further expansion from the solution is eliminated. The complexity of the method to find an optimal solution is in exponential time. Arora and Rana [AR80b] consider a heuristic approach to solve the general task assignment problem. The approach is based on the iterative transformation. The solution is based on repetitive reassignment of tasks, one at a time. The algorithm terminates when it is not possible to do any further improvement over the current solution.

Shen and Tsai [ST85] propose a graph matching approach to minimize the task turnaround time, where the optimal task assignment problem is transformed into a state space search problem. The state space is defined by the task graph and the processor graph. The state space search problem is solved by using A^* algorithm [HS78]. The graph matching approach needs exponential time in the worst case.

Kirkpatrick *et al.* [KGV83] propose a simulated annealing technique as a global optimization technique. It is derived from the observation that an optimization problem can be identified with a fluid. There exists analogy between minimizing the cost function of a combinatorial optimization problem with many variables and the slow cooling of a molten metal until it reaches its low energy ground state. Hence, the terms about energy function, temperature, and thermal equilibrium are used. During the searching of an optimal solution, higher probability is assigned to the jumps to the points with lower energy function, while there is still a small chance of jumps to the points of higher energy function. This allows the hill climbing from a local optimal configuration. If the temperature remains the same over a number of trials, the thermal equilibrium is reached. The whole process terminates when no jumps have been taken over a number of successive steps. Lin and Hsu [LH91] apply it to the task assignment problem to minimize the turnaround time. They have designed an annealing schedule in which the rules are given to choose an initial temperature, the number of trials at each temperature, the decision test for decreasing the temperature, and the stopping criterion.

Chu *et al.* [CHLE80] address the issue of tradeoff between interprocessor communication and load balancing. At one end, the tasks of an application should be assigned to a single processor to minimize interprocessor communication cost. And, at the other end, taking load balancing into account, we should equally distribute the tasks among all the processors in the system. The optimal assignment should balance these conflicting factors. Sarje and Sagar [SS91] consider the minimization of interprocessor communication cost and load balancing as the objective. A heuristic approach is proposed to find a near optimal solution in polynomial time. Initially, some tasks that can only be executed on certain processors are assigned. The algorithm then calculates the average load, which is used as a reference value. During the assignment, if a processor is under-loaded, it is favored for the target of the next unassigned task. The algorithm terminates when all tasks are assigned.

Dynamic Assignment

For static assignment problems, we assume the execution costs and communication costs for tasks are known *a priori*. However, a large distributed system may consist of hundreds of processors and there may be cases in which the assumptions made for static assignment problems may not be valid. In such cases, dynamic assignment is required.

In general, the aim of dynamic assignment is to reduce the difference in workload among processors at the run time. Barmon *et al.* [BFB91] present algorithms for dynamic load balancing. A task, input to the system through a processor, can be processed locally or transferred for processing to another processor. Dynamic load balancing algorithms distribute tasks using the current state information of the system. If the workload of a processor becomes heavy then the tasks arriving at the processor are transferred to another processor which is not heavily loaded. In their model, a processor in the system maintains two queues for its tasks: a local queue and a ready queue. A local queue holds the tasks that arrive into the system externally. A task waiting in the local queue is either transferred to the ready queue for processing, if the ready queue is not full, or transferred to another processor. Each processor estimates its own load continually, and the estimated values are used to perform the load balancing.

Chang and Oldham [CO91] also present a dynamic assignment model in which a set of task clusters is generated according to the current system work load. Each cluster is assigned to a processor. The most independent cluster (i.e. the inter-task communication cost among tasks in this cluster is the smallest) is chosen to be split and reassigned based on a best-fit policy. The algorithm also consists of a reclustering algorithm activated at run-time to reassign and group tasks to accommodate the dynamic change of load in the system.

A number of solution methods are presented in this subsection. For the special case of general task assignment problems, there exists the polynomial time optimal algorithm. Shortest path algorithms and network flow algorithms fall into this category. In general case, branch-and-bound methods and other heuristic approaches are useful to obtain the optimal or near optimal solutions. Graph matching, iterative transformation, and simulated annealing have also been used. Finally, if the objective is to maximize the load balance, we may measure or estimate the processor utilization and use it as an index to perform task assignment.

2.2.2 Reliability-Oriented Allocation

The reliability-oriented allocation problem arises when the objective is to maximize the degree of system reliability. In general, the goal is achieved by replicating the tasks to many processors. In this subsection, we present some previous work done in this area.

Static Replication

One basic issue of static replication for reliability-oriented objectives is how to model and compute system reliability. Jan *et al.* [JLT] introduce a reliability measure to model the reliability of the application. The measure is defined to be the probability that the application can be successfully executed. They consider a pseudo two-terminal series-parallel (TTSP) task graph, which is obtained from the TTSP graph by adding k replications[1] to each task and adding proper links between two tasks. They have presented a linear time algorithm for computing the reliability of pseudo TTSP graph. In the cases where tasks are allowed to have different numbers of replications, Liang *et al.* [LAMS92] consider the fork-join type task graph and present an algorithm for reliability analysis.

Shatz and Wang [SW89] consider the reliability-oriented task replication in redundant multiprocessor systems. Given a non-redundant system, we can construct a redundant system of level r by replicating a processor by r identical processors for each site in a non-redundant system, and each communication link by r identical links. Shatz and Wang present Binary/Triple Modular Redundance (B/TMR) (where $r = 2$ and 3 respectively) models and branch-and-bound algorithms for task allocation. In the models, a task, once allocated to a site, is replicated to r identical processors at that site. The objective functions of the models are functions of the system parameters (such as execution times, inter-site communication costs, and failure rates of hardware units). And the algorithms find optimal solutions using branch-and-bound techniques.

Dynamic Replication

Not much work has been reported in the literature in the area of reliability-oriented dynamic replication. It may be because the purpose of task replication is to address run-time failures, and it can be fulfilled by static replication. However, Thambidurai and Trivedi [TT89] consider real-time systems where the replications of periodic and aperiodic tasks coexist. They have presented a technique that allows a system to guarantee the execution of both periodic and aperiodic tasks within the hard deadlines. The approach is based on dynamically changing the replication factor of periodic tasks in response to aperiodic tasks. It is suitable for handling transient overloads, by decreasing the reliability of periodic tasks, when a burst of aperiodic tasks arrive in the system within a short time interval.

2.2.3 Schedulability-Oriented Allocation

The schedulability-oriented allocation problem arises in real-time systems in which the objective is to find an allocation in which the scheduling of the tasks meets the specified timing constraints.

[1] k is a constant.

Static Assignment

To guarantee the timing constraints, the basic approach is to allocate and schedule all tasks *a priori*. The global view proposed by Ramamritham [Ram90] attempts to solve the allocation problem in conjunction with the scheduling of processors and communication networks. Given a set of tasks with precedence and communications constraints, a comprehensive graph is generated. The comprehensive graph represents the overall task instances within one scheduling frame. A clustering policy is then applied to each pair of subtasks to determine whether the subtasks should be assigned to the same processor or not. Finally, a heuristic search technique is exploited to find a feasible solution.

Tindell *et al.* [TBW92] take the same global view and use a simulated annealing algorithm to solve the allocation and scheduling problem. In the work, a distributed rate-monotonic algorithm is implemented as the scheduling scheme. A token-ring protocol is used to schedule the communications. The token holding time is pre-determined to make sure that the interval is long enough for the transmission of all communications within one period. The overall allocation decision is controlled by simulated annealing.

2.3 Summary

A brief overview of the related work in allocation problems and real-time scheduling has been presented. It is clear that the desire to solve new and practical problems of allocation and scheduling has always been a driving force for the development of feasible and good techniques in multiprocessor systems. The next four chapters detail our work in the scheduling and allocation problems.

Chapter 3

Non-Preemptive Scheduling with Hybrid Timing Constraints on a Single Processor

To solve the allocation and scheduling problem in real-time multiprocessor systems, we begin with the scheduling problem of periodic tasks with hybrid timing constraints on a single processor. The solution to the single-processor case can be extended to the multiple-processor case by taking into account the allocation of the tasks and the scheduling of inter-processor communication.

In this chapter, hybrid timing constraints imposed on a periodic task are formally introduced. In Section 3.2 timing constraints are analyzed to derive the scheduling window for each task instance. In Sections 3.4 and 3.5, the approaches of scheduling a task instance and a task set respectively are presented.

3.1 Problem Statement

Consider a set of periodic tasks, $\Gamma = \{ \tau_i \mid i = 1, \ldots n \}$, where τ_i is a 6-tuple $< p_i, e_i, r_i, d_i, \lambda_i, \eta_i >$ denoting the period, computation time, ready time, deadline, low jitter and high jitter respectively. One instance of a task is executed each period. The execution of a task instance is non-preemptable. Ready time, r_i, and deadline, d_i, are the constant offsets from the beginning of the period. The start times of two consecutive instances of task τ_i are at least $p_i - \lambda_i$ and at most $p_i + \eta_i$ apart.

In order to schedule periodic tasks, we consider the least common multiple (LCM) of all periods of tasks. Let n_i be the number of instances for task τ_i within a schedule of length LCM. Hence, $n_i = \frac{\mathrm{LCM}}{\mathrm{p_i}}$. A schedule for a set of tasks is the mapping of each task τ_i to n_i task instances and the assigning of a start time s_i^j to the j-th instance of task τ_i, τ_i^j, $\forall\ i = 1, \ldots n$ and $j = 1, \ldots, n_i$.

Define r_i^j and d_i^j to be the absolute ready time and deadline for task instance τ_i^j. Namely, $r_i^j = p_i \times (j-1) + r_i$, and $d_i^j = p_i \times (j-1) + d_i$. A *feasible* schedule is a schedule in which the following conditions are satisfied for each task τ_i:

$$
\begin{aligned}
f_i^j &= s_i^j + e_i && \text{(execution time)} && (3.1)\\
s_i^{n_i+1} &= s_i^1 + \mathrm{LCM} && \text{(periodicity)} && (3.2)\\
s_i^j &\geq r_i^j && \text{(ready time)} && (3.3)\\
f_i^j &\leq d_i^j && \text{(deadline)} && (3.4)\\
s_i^j &\geq s_i^{j-1} + p_i - \lambda_i && \text{(low jitter)} && (3.5)\\
s_i^j &\leq s_i^{j-1} + p_i + \eta_i && \text{(high jitter)} && (3.6)\\
& && \forall j = 2, \ldots, n_i + 1. &&
\end{aligned}
$$

The non-preemption scheduling discipline leads to Equation 3.1 where f_i^j is the finish time of τ_i^j. Another condition for non-preemption scheduling is that given any i, j, k and ℓ, if $s_i^j < s_k^\ell$ then $f_i^j \leq s_k^\ell$. It means the schedule for any two instances is non-overlapping. The constructed schedule of length LCM is invoked repeatedly by wrapping-around the end point of the first schedule to the start point of the next. Hence, as shown in Equation 3.2, the start time of the first instance in the next schedule is exactly one LCM away from that of the first schedule. And the absolute timing constraints on each task instance are given as shown in Equations 3.3 and 3.4. Finally, Equations 3.5 and 3.6 specify the relative timing constraints between two consecutive instances of a task.

3.2 Analysis of Relative Timing Constraints

Define the scheduling window for a task instance as the time interval during which the task can start. Traditionally, the lower and upper bounds of the scheduling window for a task instance are called earliest start time (*est*) and latest start time (*lst*) respectively. These values are given and independent of the start times of the preceding instances.

We consider the scheduling of periodic tasks with hybrid timing constraints described in Equations 3.3 through 3.6. The scheduling window for a task instance is derived from the start times of its preceding instances. A *feasible* scheduling window for a task instance τ_i^j is a scheduling window in which any start time makes the timing relation between s_i^{j-1} and s_i^j satisfy Equations 3.5 and 3.6. Formally, given s_i^1, s_i^2, …, and …, s_i^{j-1}, the problem is to derive the feasible scheduling window for τ_i^j such that a feasible schedule can be obtained if τ_i^j is scheduled within the window.

ID	$est = s_i^{j-1} + p_i - \lambda_i$	$lst = s_i^{j-1} + p_i + \eta_i$	start time (s_i^j)
τ_i^1	0	40	4
τ_i^2	39	49	40
τ_i^3	75	85	77
τ_i^4	112	122	113
τ_i^5	148	158	*

Table 3.1: An example to show the wrong setting of scheduling windows

For the sake of simplicity, we assume that $r_i = 0$ and $d_i = p_i$, $\forall\ i$, in this section. Then, simply assigning *est* and *lst* of τ_i^j as $s_i^{j-1} + p_i - \lambda_i$ and $s_i^{j-1} + p_i + \eta_i$ respectively where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n_i$, is not tight enough to guarantee a feasible solution. For example, consider the case shown in Table 3.1 in which a periodic task τ_i is to be scheduled. Let LCM, p_i, λ_i, and η_i be 200, 40, 5, and 5 respectively. Hence, there are 5 instances within one LCM (i.e. $n_i = 5$). The first column in Table 3.1 indicates the instance IDs. The second and third columns give the *est* and *lst* of the scheduling windows for the task instances specified in the first column. The last column shows the actual start times scheduled for the particular task instances. The actual start time is a value between *est* and *lst* of each task instance. For instance, the *est* and *lst* of τ_i^2 are 39 and 49 respectively. It means $39 \leq s_i^2 \leq 49$. The scheduled value for s_i^2, in the example, is 40. Since $s_i^6 = s_i^1 + \text{LCM} = 204$, we find that any value in the interval [148,158] can not satisfy the relative timing constraints between τ_i^5 and τ_i^6. As a consequence, the constructed schedule is infeasible.

We draw a picture to depict the relations among the start times of task instances in Figure 3.1. When τ_i^j is taken into account, the scheduling window for s_i^j is obtained by considering its relation with s_i^{j-1} as well as that with $s_i^{n_i}$ and $s_i^{n_i+1}$. We make sure that once s_i^j is determined, the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible scheduling window for $s_i^{n_i}$. Namely, the interval which is specified by the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j, overlaps the interval $[s_i^{n_i+1} - (p_i + \eta_i),\ s_i^{n_i+1} - (p_i - \lambda_i)]$.

Proposition 3.1 *Let the est and lst of* τ_i^j *be*

$$\begin{aligned} est(\tau_i^j) &= max\{(s_i^{j-1} + p_i - \lambda_i),\ (s_i^1 + (j-1) \times p_i - \\ &\quad (n_i - j + 1) \times \eta_i)\}, \qquad (3.7) \\ \text{and } lst(\tau_i^j) &= min\{(s_i^{j-1} + p_i + \eta_i),\ (s_i^1 + (j-1) \times p_i + \\ &\quad (n_i - j + 1) \times \lambda_i)\}. \qquad (3.8) \end{aligned}$$

If s_i^j *is in between the* $\text{est}(\tau_i^j)$ *and* $\text{lst}(\tau_i^j)$*, then the estimated* est *and* lst *of* $s_i^{n_i}$*, based on* s_i^j *and* $s_i^{n_i+1}$*, specify a feasible window.*

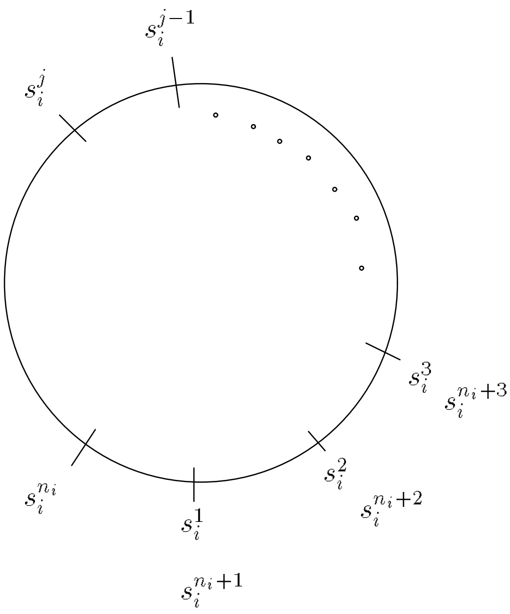


Figure 3.1: The relations between the task instances

Proof: Let ℓ and μ be the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j, respectively.

Accordingly,

$$\ell = s_i^j + (n_i - j) \times (p_i - \lambda_i) \tag{3.9}$$
$$\mu = s_i^j + (n_i - j) \times (p_i + \eta_i) \tag{3.10}$$

To guarantee the existence of feasible start time of $\tau_i^{n_i}$, the interval $[\ell,\mu]$ has to overlap the interval $[s_i^{n_i+1} - (p_i + \eta_i), s_i^{n_i+1} - (p_i - \lambda_i)]$. Thus the following conditions have to be satisfied:

$$s_i^{n_i+1} - \ell \geq p_i - \lambda_i \tag{3.11}$$
$$s_i^{n_i+1} - \mu \leq p_i + \eta_i \tag{3.12}$$

By replacing ℓ in Equation 3.11 with $s_i^j + (n_i - j) \times (p_i - \lambda_i)$, we obtain

$$\begin{aligned} s_i^j &\leq s_i^{n_i+1} - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + \mathrm{LCM} - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + n_j \times p_i - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \end{aligned} \tag{3.13}$$

Instance ID	*est* from Eq. 3.7	*lst* from Eq. 3.8	actual start time (s_i^j)
τ_i^1	0	40	4
τ_i^2	39	49	40
τ_i^3	75	85	77
τ_i^4	114	122	115
τ_i^5	159	160	$159 \sim 160$

Table 3.2: The correct setting of scheduling windows based on Proposition 3.1.

Likewise, by replacing μ in Equation 3.12 with $s_i^j + (n_i - j) \times (p_i + \eta_i)$, we have

$$\begin{aligned} s_i^j &\geq s_i^{n_i+1} - (n_i - j + 1) \times (p_i + \eta_i) \\ &= s_i^1 + \mathrm{LCM} - (n_i - j + 1) \times (p_i + \eta_i) \\ &= s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i \qquad (3.14) \end{aligned}$$

So, According to Equations 3.14 and 3.5, we choose the bigger value between $(s_i^{j-1} + p_i - \lambda_i)$ and $(s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)$ as the *est* of τ_i^j. Similarly, according to Equations 3.13 and 3.6, we assign the smaller value of $(s_i^{j-1} + p_i + \eta_i)$ and $(s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)$ as the *lst*.

□

Example 3.1: To show how Proposition 3.1 gives a tighter bound to find feasible scheduling windows, we consider the case shown in Table 3.1 again. We apply Equations 3.7 and 3.8 to compute the *est* and *lst* of each instance. The results are shown in Table 3.2. Note that the scheduling windows for τ_i^4 and τ_i^5 are tighter than those in Table 3.1. As a consequence, any start time in the interval [159,160] for τ_i^5 satisfies the relative timing constraints between τ_i^5 and τ_i^6.

3.2.1 Property of Scheduling Windows

Let us define $P_i(x, y, z)$ as the predicate in which the estimated *est* and *lst* of τ_i^y, based on s_i^x and s_i^z, specify a feasible scheduling window for τ_i^y. In Proposition 3.1 , we prove that for any s_i^j in between *est*(τ_i^j) and *lst*(τ_i^j) as specified in Equations 3.7 and 3.8, $P_i(j, n_i, n_i + 1)$ is true.

Lemma 3.1 *Given s_i^1, s_i^2, ..., and s_i^j, if, $\forall\ k = 2, \ldots, j$, $\mathrm{est}(\tau_i^k) \leq s_i^k \leq \mathrm{lst}(\tau_i^k)$ as specified in Equations 3.7 and 3.8, then $P_i(j, y, n_i + 1)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.*

Proof: We prove that the estimated *est* and *lst* of τ_i^y, based on s_i^j and $s_i^{n_i+1}$, specify a feasible scheduling window, by showing that (1) the estimated scheduling window of s_i^y, based on s_i^j, is

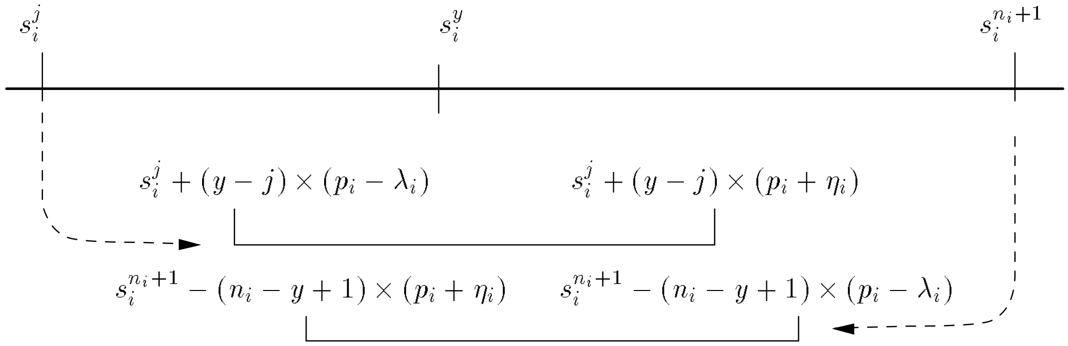


Figure 3.2: The overlapping of two intervals

specified by the interval

$$[s_i^j + (y-j) \times (p_i - \lambda_i), s_i^j + (y-j) \times (p_i + \eta_i)], \tag{3.15}$$

(2) the estimated scheduling window of s_i^y, based on $s_i^{n_i+1}$, is specified by the interval

$$[s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i), s_i^{n_i+1} - (n_i - y + 1) \times (p_i - \lambda_i)], \tag{3.16}$$

and (3) the intervals in Equations 3.15 and 3.16 overlap.

In Figure 3.2, we see that the necessary and sufficient conditions for the overlapping of the intervals specified in Equations 3.15 and 3.16 are

$$\begin{aligned} s_i^j + (y-j) \times (p_i - \lambda_i) &\leq s_i^{n_i+1} - (n_i - y + 1) \times (p_i - \lambda_i) && (3.17) \\ \text{and} \quad s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i) &\leq s_i^j + (y-j) \times (p_i + \eta_i). && (3.18) \end{aligned}$$

By solving the Equations 3.17 and 3.18, we obtain

$$\begin{aligned} s_i^j &\leq s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \\ \text{and} \quad s_i^j &\geq s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i. \end{aligned}$$

The above two equations describe the same conditions as Equations 3.13 and 3.14 do. Hence, $P_i(j, y, n_i + 1)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.

□

Lemma 3.2 *Given s_i^1, s_i^2, ..., s_i^j, and an integer n_0, where $1 \leq n_0 \leq j$, if, $\forall\ k = 2, \ldots, j$, $est(\tau_i^k) \leq s_i^k \leq lst(\tau_i^k)$ are specified as in Equations 3.7 and 3.8, then $P_i(j, y, n_i + n_0)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.*

Proof: We use the same method in Lemma 3.1 to prove it. We show that (1) the estimated scheduling window of s_i^y, based on s_i^j, is specified by the interval

$$[s_i^j + (y-j) \times (p_i - \lambda_i), s_i^j + (y-j) \times (p_i + \eta_i)], \tag{3.19}$$

(2) the estimated scheduling window of s_i^y, based on $s_i^{n_i+n_0}$, is specified by the interval

$$[s_i^{n_i+n_0} - (n_i + n_0 - y) \times (p_i + \eta_i), s_i^{n_i+n_0} - (n_i + n_0 - y) \times (p_i - \lambda_i)], \tag{3.20}$$

and (3) these two intervals overlap.

The following conditions have to be satisfied to make sure the overlapping of the two intervals.

$$s_i^j \leq s_i^{n_0} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i - (p_i - \lambda) \times n_0 - 1 \tag{3.21}$$

$$\text{and } s_i^j \geq s_i^{n_0} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i - (p_i + \eta_i) \times n_0 - 1. \tag{3.22}$$

Since $s_i^1 \leq s_i^{n_0} - (p_i - \lambda) \times (n_0 - 1)$ and $s_i^1 \geq s_i^{n_0} - (p_i + \eta_i) \times (n_0 - 1)$, we rewrite Equations 3.21 and 3.22

$$\begin{aligned}
s_i^j &\leq \underline{s_i^{n_0}} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \underline{- (p_i - \lambda) \times n_0 - 1} \\
&\leq \underline{s_i^1} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \\
\text{and } s_i^j &\geq \underline{s_i^{n_0}} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i \underline{- (p_i + \eta_i) \times n_0 - 1}. \\
&\geq \underline{s_i^1} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i
\end{aligned}$$

Hence $P_i(j, y, n_i + n_0)$ holds for any $1 \leq n_0 \leq j$.

□

Theorem 3.1 *Given s_i^1, s_i^2, …, and s_i^j, if, $\forall$ $k = 2$, …, j, $\mathrm{est}(\tau_i^k) \leq s_i^k \leq \mathrm{lst}(\tau_i^k)$ as specified in Equations 3.7 and 3.8, then $P_i(j, y, z)$ is true, $\forall$ $y = j+1$, $j+2$, …, n_i, and $z = n_i + 1$, $n_i + 2$, …, $n_i + j$.*

By combining the proofs in Lemmas 3.1 and 3.2, it is easy to see that Theorem 3.1 holds. Based on Theorem 3.1 , we can assign the scheduling window for τ_i^j by using Equations 3.7 and 3.8 once s_i^1, s_i^2, …, s_i^{j-1}.

3.3 Bounds of A Scheduling Window with Hybrid Timing Constraints

In Section 3.2, the earliest and latest start time for a task instance with relative timing constraints are derived in Equations 3.7 and 3.8. When the absolute timing constraints are taken into account, the bounds of the scheduling window for τ_i^j are modified as follows.

$$\begin{aligned} est(\tau_i^j) &= max\{r_i^j,\ (s_i^{j-1} + p_i - \lambda_i),\ (s_i^1 + (j-1) \times p_i - \\ &\quad (n_i - j + 1) \times \eta_i)\}, \qquad (3.23) \\ lst(\tau_i^j) &= min\{(d_i^j - e_i),\ (s_i^{j-1} + p_i + \eta_i),\ (s_i^1 + (j-1) \times p_i + \\ &\quad (n_i - j + 1) \times \lambda_i)\}. \qquad (3.24) \end{aligned}$$

Before we present the scheduling technique for a task instance, let us consider the following objective. Given a set of tasks with the characteristics described in Section 3.1, we schedule the task instances for each task within one LCM to minimize

$$\pi = \sum_{i,j} \alpha(s_i^j - s_i^{j-1} - p_i) \qquad (3.25)$$

Subject to the constraints specified in Equations 3.1 through 3.6,

where $\alpha(x) = x$, if $x \geq 0$; $= -x$, otherwise.

Basically, we try to schedule every instance of a task one period apart from its preceding instance. An optimal schedule is a feasible schedule with the minimum total deviation value from one period apart for instances.

3.4 The Time-Based Scheduling of a Task Instance

We consider the time-based solution to the scheduling problem by using a linked list. Each element in the list represents a time slot assigned to a task instance. A time slot w has the following fields: (1) *task id i* and *instance id j* indicate the identifier of the time slot. (2) *start time st* and *finish time ft* indicate the start time and completion time of τ_i^j respectively. (3) *prev ptr* and *next ptr* are the pointers to the preceding and succeeding time slots respectively. We arrange the time slots in the list in increasing order by using the *start time* as the key. Any two time slots are non-overlapping. Since the execution of an instance is non-preemptable, the time difference between *start time* and *finish time* equals the execution time of the task.

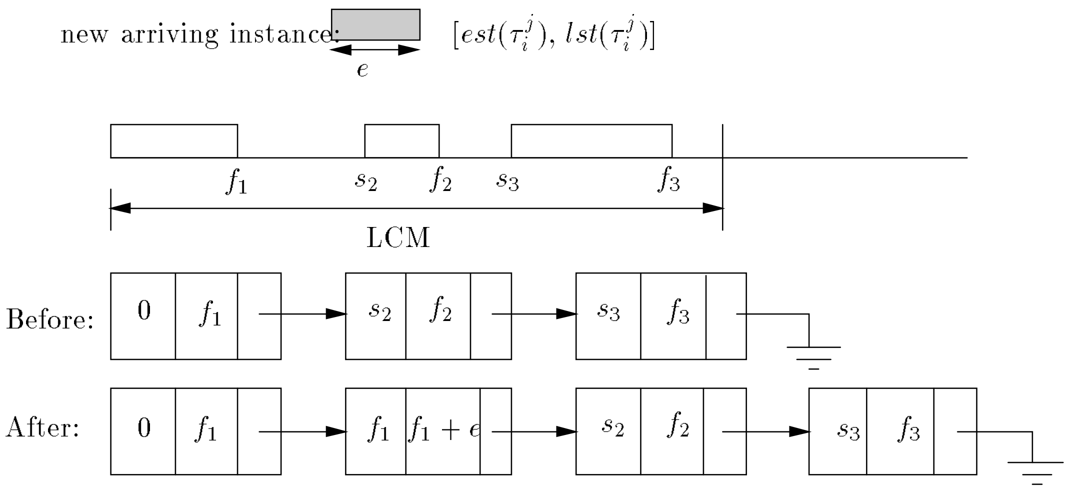


Figure 3.3: Insertion of a new time slot

3.4.1 Creating a Time Slot for the Task Instance

Consider a set of n tasks. Given a linked list and a task instance τ_i^j, we schedule the instance by inserting a time slot in the list. According to equations 3.23 and 3.24, we compute the $est(\tau_i^j)$ and $lst(\tau_i^j)$ first. Let S be the set of unoccupied time intervals that overlap the interval $[est(\tau_i^j),\ lst(\tau_i^j)]$ in the linked list. The unoccupied time intervals in S are collected by going through the list. Each time when a pair of time slots $(w, w+1)$ is examined, we compute $\ell = \max\{est(\tau_i^j),\ ft(w)\}$ and $\mu = \min\{lst(\tau_i^j),\ st(w+1)\}$, where $ft(w)$ is the finish time of the time slot w, and $st(w+1)$ is the start time of the slot next to w. If $\ell \leq \mu$, then we add the interval $[\ell,\ \mu]$ to S.

The free intervals in S are the potential time slots which τ_i^j can be assigned to. Since we try to schedule τ_i^j as close to one period away from the preceding instance τ_i^{j-1} as possible, we sort S, based on the function of the lower bound of each interval, $\alpha(s_i^{j-1} + p_i - \ell)$, in ascending order. Without loss of generality, we assume that S after the sorting is denoted by $\{int_1,\ int_2,\ \ldots,\ int_{|S|}\}$ The idea is that if τ_i^j is scheduled to int_k, then the value in equation 3.25 will be smaller than that of the case in which τ_i^j is scheduled to int_{k+1}.

The scheduling of τ_i^j can be described as follows. Starting from int_1, we check whether the length of the interval is greater or equal to the execution time of τ_i^j. If yes, then we schedule the instance to the interval. One new time slot is created in which the start time is the lower bound of the interval and the finish time equals the start time plus the execution time. The created time slot is added to the linked list and the scheduling is done. If the length is smaller than the execution time, then we check the length of the next interval until all intervals are examined. An example is shown in Figure 3.3 in which the slot with the dark area represents τ_i^j. In this example we assume

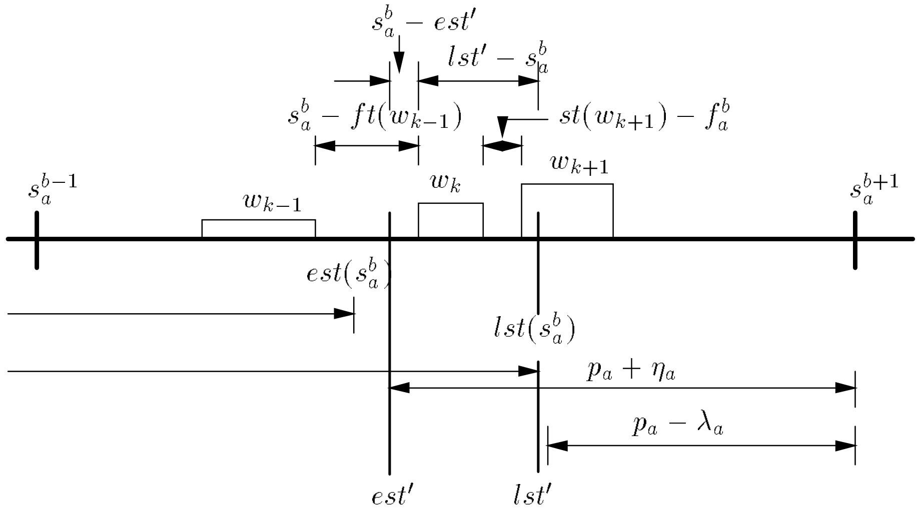


Figure 3.4: An illustration of $left\_laxity(w_k)$ and $right\_laxity(w_k)$

that $est(\tau_i^j) \leq f_1$ and $s_2 - f_1 > e$. It means the free slot between the first and second occupied slots can be assigned to τ_i^j.

3.4.2 Sliding of the Time Slots

If none of the intervals in S can accommodate a task instance, the sliding technique is used to create a big enough interval by sliding the existing time slots in the list. To make the sliding technique work, we maintain two values for each time slot: *left laxity* and *right laxity*. The value of left laxity indicates the amount of time units by which a time slot can be left-shifted to an earlier start time. Similarly, the right laxity indicates the amount of time units by which a time slot can be right-shifted to a later start time.

Given the time slots w_{k-1}, w_k, and w_{k+1}, where a and b are the task and instance identifiers of w_k respectively, the laxity values of the time slot w_k can be computed by:

$$\begin{aligned} left\_laxity(w_k) &= min\{s_a^b - est',\ s_a^b - ft(w_{k-1}) + \\ &\qquad left\_laxity(w_{k-1})\} \qquad (3.26) \\ right\_laxity(w_k) &= min\{lst' - s_a^b,\ st(w_{k+1}) - f_a^b + \\ &\qquad right\_laxity(w_{k+1})\} \qquad (3.27) \\ \text{where} &\quad est' = max\{est(\tau_a^b),\ s_a^{b+1} - (p_a + \eta_a)\} \\ \text{and} &\quad lst' = min\{lst(\tau_a^b),\ s_a^{b+1} - (p_a - \lambda_a)\}. \end{aligned}$$

Note that the interval $[est',\ lst']$ defines the sliding range during which τ_a^b can start without shifting τ_a^{b-1} or τ_a^{b+1}. A schematic illustration of equations 3.26 and 3.27 is given in Figure 3.4.

From equations 3.26 and 3.27, we see that the computing of *left_laxity*(w_k) depends on that of w_{k-1} and the computing of *right_laxity*(w_k) depends on that of w_{k+1}. It implies a two-pass computation is needed to compute the laxity values for all time slots. The complexity is $O(2N)$ where N is the number of time slots in the linked list.

The basic idea of the sliding technique is described as follows. Given a task instance τ_i^j and a set of unoccupied intervals, $S = \{int_1, int_2, \ldots, int_{|S|}\}$, we check one interval at a time to see if the interval can be enlarged by shifting the existent time slots. Two ways of enlargement are (1) by either shifting the time slots, that precede the interval, to the left or (2) shifting the slots, that follow the interval, to the right. The shifting depends on which direction minimizes the objective function in Equation 3.25.

3.4.3 The Algorithm

An algorithmic description about how to schedule a task instance, as described in Sections 3.4.1 and 3.4.2, is given in Figure 3.5.

The procedures Left_Shift(w_k,time_units) and Right_Shift(w_k,time_units) in Figure 3.5 may involve the shifting of more than one time slot recursively. For example, consider the case in Figure 3.4, if Right_Shift(w_k,$lst' - s_a^b$) is invoked (i.e. w_k is to be shifted right by $lst' - s_a^b$ time units), then w_{k+1} has to be shifted too. It is because the gap between w_k and w_{k+1} is $st(w_{k+1}) - f_a^b$ which is smaller than $lst' - s_a^b$. In this case, Right_Shift(w_{k+1},$lst' - s_a^b - st(w_{k+1}) + f_a^b$) is invoked.

We do not enlarge an interval at both ends. Enlarging an interval at both ends needs to shift certain amount of preceding time slots to the left and shift some succeeding slots to the right. It is possible that some task instance τ_x^y is shifted left, while τ_x^{y+1} is shifted right. As a consequence, the timing constraints between s_x^y and s_x^{y+1} could be violated. For example, Let s_x^y and s_x^{y+1} before the shifting be 10 and 20 respectively. The execution time for τ_x is 5 time units. Assume the *left laxity* of τ_x^y is 5 and the *right laxity* of τ_x^{y+1} is 5. It implies $s_x^{y+1} - s_x^y \leq 15$. Consider the scheduling of a task instance τ_i^j with execution time 15. If we enlarge the interval between τ_x^y and τ_x^{y+1} by shifting τ_x^y left 5 time units and τ_x^{y+1} right 5 time units, then we get a new interval with 15 time units for τ_i^j. However, it turns out that $s_x^{y+1} = 25$, $s_x^y = 5$, and the relative timing constraints between τ_x^y and τ_x^{y+1} is violated.

3.5 The Priority-Based Scheduling of a Task Set

We consider the priority-based algorithms for scheduling a set of periodic tasks with hybrid timing constraints. Given a set of periodic tasks $\Gamma = \{ \tau_i \mid i = 1, \ldots, n \}$ with the task characteristics described in Section 3.1, we compute the LCM of all periods. Each task τ_i is extended to n_i task instances: $\tau_i^1, \tau_i^2, \ldots, \tau_i^{n_i}$. A scheduling algorithm σ for Γ is to totally order the instances of all tasks within the LCM. Namely, σ : *task_id* $\times$ *instance_id* $\rightarrow$ *integer*.

Three algorithms are considered. They are *smallest latest-start-time first* (SLsF), *smallest period first* (SPF), and *smallest jitter first* (SJF) algorithms.

3.5.1 SLsF

The scheduling window for a task instance τ_i^j depends on the scheduling of its preceding instance. Once s_i^{j-1} is determined, the scheduling window of the instance can be computed by equations 3.23 and 3.24. The scheduling window for the first instance of a task τ_i is defined as $[r_i, d_i - e_i]$.

The idea of the SLsF algorithm is to pick one candidate instance with the minimum *lst* among all tasks at a time. One counter for each task is maintained to indicate the candidate instance. All counters are initialized to 1. Each time when a task instance with the smallest *lst* is chosen, the algorithm in Figure 3.5 is invoked to schedule the instance. After the scheduling of the instance is done, the counter is increased by one. The counter for τ_i overflows when it reaches $n_i + 1$. It means that all the instances of τ_i are scheduled. The algorithm terminates when all counters overflow.

We can compute the relative deadline for a task instance by adding the execution time to the *lst*. If the execution times for all tasks are identical, the SLsF algorithm is equivalent to the *earliest deadline first* (EDF) algorithm.

3.5.2 SPF

The task periods determine the LCM of Γ and the numbers of instances for tasks within the LCM. In most cases, the task with the smaller period has the tighter timing constraints. Namely, $(\lambda_i + \eta_i) \leq (\lambda_j + \eta_j)$ if $p_i \leq p_j$. To make the tasks with the smaller periods meet their timing constraints, the SPF algorithm favors the tasks with smaller periods.

The SPF algorithm uses the period as the key to arrange all tasks in non-decreasing order. The task with the smallest period is selected for scheduling first. The instances of a particular task are scheduled one by one by invoking the algorithm in Figure 3.5. After all the instances of a task are scheduled, the next task in the sequence is scheduled.

3.5.3 SJF

We define the *jitter* of a task τ_i as $(\lambda_i + \eta_i)$. It is proportional to the range of the scheduling window. Hence, the schedulability of a task also depends on the jitter.

Instead of using the period as the measurement, the SJF algorithm assigns the higher priority to the tasks with the smaller jitters. The task with the smallest jitter is scheduled first.

3.5.4 The Solution

The composition of the time-based scheduling of a task instance and the priority assignment of task instances is shown in Figure 3.6. The priority assignment can be done by using SLsF, SPF, or SJF. The function *Schedule_An_Instance()* is invoked to schedule a single task instance.

3.6 Experimental Evaluation

We conduct two experiments to study and compare the performance of the three algorithms. The purpose of the first experiment is to study the effect of the number of tasks and utilization on the schedulability of each algorithm. The objective of the second experiment is to compare the performance of the three algorithms.

3.6.1 The First Experiment

The task generation scheme for the first experiment is characterized by the following parameters.

- Periods of the tasks: We consider a homogeneous system in which the period of one task could be either the same as or multiple of the period of another. We consider a system with 40, 80, 160, 320, and 640 as the candidate periods. There may be more than one task with the same period.
- The execution time of a task, e_i : It has the uniform distribution over the range $[0, \frac{p_i}{16}]$, where p_i is the period of the task τ_i. The execution time could be a real value.
- The jitters of a task: $\lambda_i = \eta_i = 0.1 \times p_i$.

We define the utilization of a task system as

$$\sum_{i=1}^{N} \frac{e_i}{p_i} \tag{3.28}$$

In the first experiment, the utilization value and the number of tasks in a set are the controlled variables. Given utilization value U and the number of tasks N the scheme first generates a set of raw data by randomly generating a set of N tasks based on the the selected periods, jitter values, and the execution time distribution. The utilization of the raw data, u, is then computed by Equation 3.28. Finally, the utilization value of the raw data is scaled up or down to U by multiplying $\frac{U}{u}$ to the execution time of each generated task. As a consequence, we obtain a set of tasks with the specified (U,N) value.

For each combination of (U,N) in which U = 5%, 10%, 15%, ... 100% and N = 10, 20, and 30, we apply the scheme to generate 5000 cases of input data and use the three algorithms to schedule them. The schedulability degree of each (U,N) combination for an algorithm is obtained by dividing the number of solved cases by 5000. Since the jitter values is 1/10 of periods, it is observed that the SPF and SJF algorithms yield the same results. The results are shown in Figure 3.7.

As can be seen in Figures 3.7(a) and (b) the number of tasks has different effects on the three algorithms. For SLsF, given a fixed utilization value, the schedulability degree increases as the number of tasks in a system becomes bigger. It is because the execution time of a task becomes smaller as the number of tasks increases. For a task system with a smaller execution time distribution, the chance for SLsF to find a feasible solution is bigger. The same phenomenon is also found in Figure 3.7(b) for SPF and SJF in the low-utilization cases (i.e. $U \leq 20\%$). However, for the high-utilization cases in Figure 3.7(b), the complexity of the number of tasks dominates the algorithms and the schedulability decreases.

3.6.2 The Second Experiment

The task generation scheme for the second experiment is characterized by the following parameters.

- LCM = 300
- The number of tasks is 20.
- Periods of the tasks: We consider the factors of the LCM as the periods. They are 20, 30, 50, 60, 100, 150, and 300. There may be more than one task with the same period.
- The execution time of a task, e_i : It has the uniform distribution over the range $[0,\frac{p_i}{15}]$, where p_i is the period of the task τ_i. The execution time could be a real value.
- The jitters of a task: $\lambda_i = \eta_i = 0.1 \times p_i + 2 \times e_i$.

The generation scheme for the second experiment is similar to the first one. Given a utilization value U, a set of 20 tasks is randomly generated according to the parameters listed above and then the execution time of each task is normalized in order to make the utilization value equal to U.

We generate 5000 cases of different task sets for each utilization value ranging from 0.05 to 1.00. The schedulability degree of each algorithm on a particular utilization value is obtained by dividing the number of solved cases by 5000. We compare the schedulability degrees of the algorithms on different utilization values. The results are shown in Figure 3.8(a).

As can be seen in Figure 3.8(a) the SLsF algorithm outperforms the other two algorithms. For example, when the utilization = 50%, the schedulability degree of SLsF is 0.575 while those of SPF and SJF are less than 0.2. It is because the way of assigning the priorities to the task instances in the SLsF algorithm reflects the urgency of task instances by considering the latest start times.

We also compare the objective function value π in Equation 3.25 among the three algorithms. We define the normalized objective function for an algorithm as

$$\hat{z} = \sum_{i=1}^{5000} \frac{z_i}{5000}, \tag{3.29}$$

$$\text{where} \quad z_i = \begin{cases} 1 & \text{if the algorithm can not find a feasible solution} \\ & \text{to case i.} \\ 0 & \text{if } max(i) = min(i). \\ \frac{\pi_i - min(i)}{max(i) - min(i)} & \text{otherwise.} \end{cases}$$

Given case i, the values of $min(i)$ and $max(i)$ are calculated among the objective values obtained from the algorithms which solve the case. For the algorithms which can not find a feasible solution to case i, the objective values are not taken into account when $min(i)$ and $max(i)$ are calculated. The results of the normalized objective functions for each algorithm on different utilization values are shown in Figure 3.8(b).

It is observed that in the low-utilization cases SJF finds feasible solutions with smaller objective values. It is because SJF schedules the tasks with the smallest jitters first. By scheduling the tasks with the smaller jitter value first it is easier to make the instances of a task one period apart. However, in the middle- or high-utilization cases, the schedulability dominates the normalized objective function, and SLsF outperforms the other two algorithms in these regions.

3.7 Summary

In this chapter we have considered the static non-preemptive scheduling algorithm on a single processor for a set of periodic tasks with hybrid timing constraints. The time-based scheduling algorithm is used to schedule a task instance once the scheduling window of the instance is given. We have also presented three priority assignment algorithms for the task instances and conducted experiments to compare the performance. From the simulation results, we see that the SLsF outperforms the other two algorithms.

The techniques presented in this chapter can be applied to multi-processor real-time systems. Communication and synchronization constraints can be also incorporated. In the next chapter, the extension to a distributed computing systems is presented.

```
Schedule_An_Instance (τ_i^j):
Input: A linked list, a task instance τ_i^j and a sequence of sorted free
    intervals, S = { int_1, int_2, ..., int_|S| }, in which each
    interval overlaps [est(τ_i^j),lst(τ_i^j)].
Let the execution time of τ_i^j be e.
For n = 1 to |S| do
    Let int_n be [ℓ,μ].
    If μ − ℓ ≥ e then
        Return a new time slot with start time = ℓ
            and finish time = ℓ + e.
Compute left laxity and right laxity for each time slot in the linked list
    by equations 3.26 and 3.27.
For n = 1 to |S| do
    Let int_n be [ℓ,μ].
    If ℓ ≥ s_i^{j−1} + p_i then /* Try left shift first then right shift */
        Let the time slot that immediately precedes int_n be w_k.
        If left_laxity(w_k) + μ − ℓ ≥ e then /* Left shift */
            Left_Shift(w_k,e − μ + ℓ).
            Return a new time slot with start time = μ − e
                and finish time = μ.
        Else
            Let the time slot that immediately follows int_n be w_k.
            If right_laxity(w_k) + μ − ℓ ≥ e then /* Right shift */
                Right_Shift(w_k,e − μ + ℓ).
                Return a new time slot with start time = ℓ
                    and finish time = ℓ + e.
    Else /* Try right shift first then left shift */
        Let the time slot that immediately follows int_n be w_k.
        If right_laxity(w_k) + μ − ℓ ≥ e then /* Right shift */
            Right_Shift(w_k,e − μ + ℓ).
            Return a new time slot with start time = ℓ
                and finish time = ℓ + e.
        Else
            Let the time slot that immediately precedes int_n be w_k.
            If left_laxity(w_k) + μ − ℓ ≥ e then /* Left shift */
                Left_Shift(w_k,e − μ + ℓ).
                Return a new time slot with start time = μ − e
                    and finish time = μ.
Schedule τ_i^j at the end of linked list.
```

Figure 3.5: The Scheduling of a Task Instance

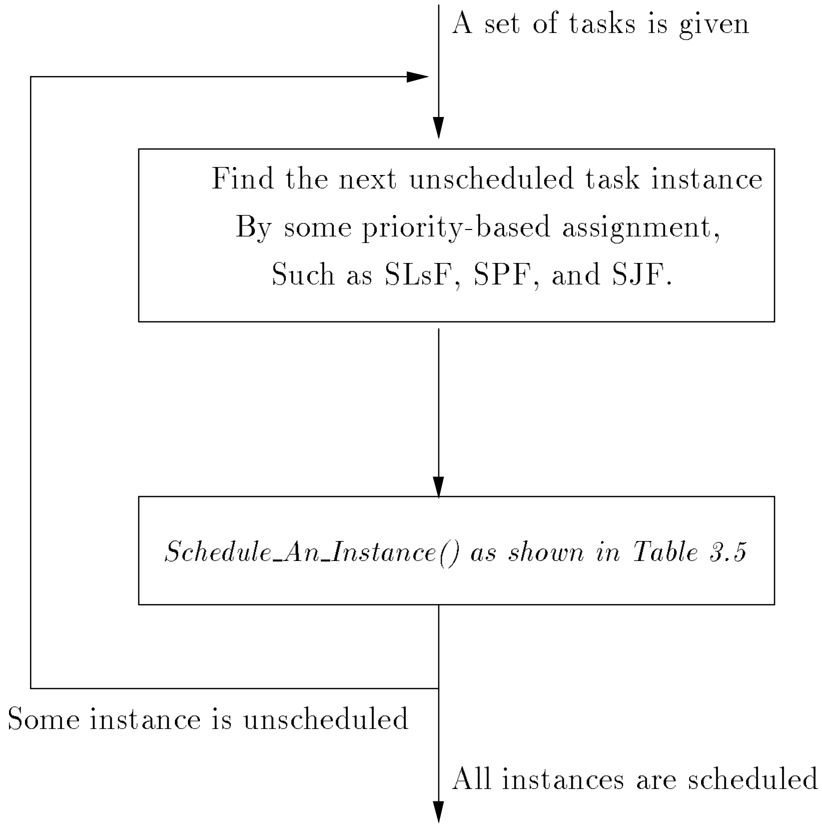


Figure 3.6: A schematic flowchart for the solution

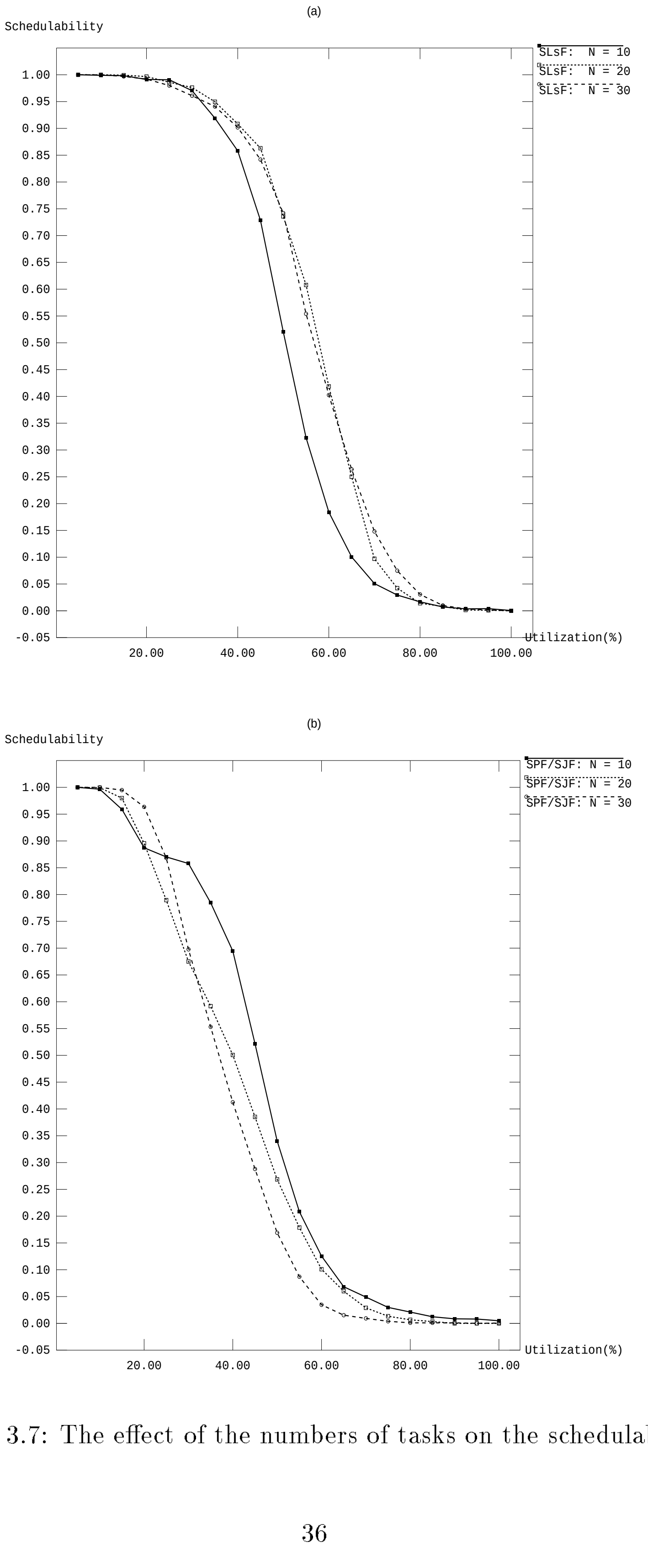


Figure 3.7: The effect of the numbers of tasks on the schedulability

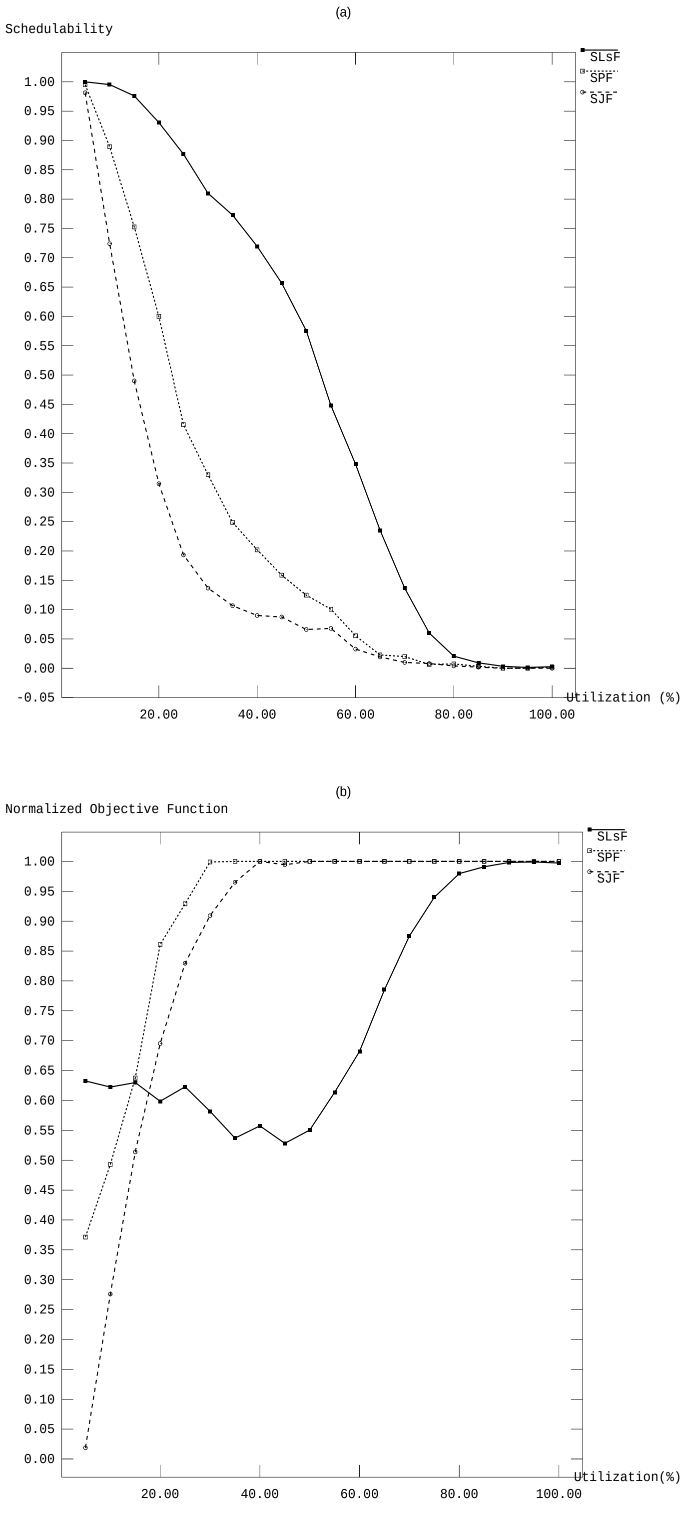


Figure 3.8: The comparison of three algorithms

Chapter 4

Multiprocessor Allocation and Scheduling

In this chapter, we consider the allocation and scheduling of periodic tasks with the hybrid timing constraints on a real-time multiprocessor system. The task characteristics, obtained from the real examples of the real-time applications, are described in Section 4.1. We propose the approach to the allocation and scheduling of these applications in Section 4.2. A simulated annealing algorithm is developed in Section 4.3 to solve the problem in which the solution is based on the results from Chapter 3. In Section 4.4, we evaluate the practicality and show the significance of the algorithm. Instead of randomly generating the *ad hoc* test cases, we apply the algorithm to a real example. The example is the Boeing 777 Aircraft Information Management System (AIMS) with various numbers of processors. The experimental results are shown in Section 4.4.

4.1 Problem Description

Various kinds of periodic task models have been proposed to represent real-time system characteristics. One of them models an application as an independent set of tasks, in which each task is executed once every period under the ready time and deadline constraints. Synchronization (e.g. precedence and mutual exclusion) and communications are simply ignored. Another model takes the precedence relationship and communications into account and models the application as a task graph. In a task graph, tasks are represented as nodes while communications and precedence relationship between tasks are represented as edges. The absolute timing constraints can be imposed on the tasks. The allocation of tasks, and the scheduling of tasks and communications, have to meet their timing constraints as well as the synchronization constraints. The deficiency of task graph modeling is that one cannot specify the relative constraints across task periods. For example, one cannot specify the minimum separation interval between two consecutive executions of the same task.

In Chapter 3 we augmented the real-time system characteristics by considering the relative constraints on the instances of a task. In this chapter we consider the system with inter-task dependency.

4.1.1 Task Characteristics

The problem considered in this chapter has the following characteristics.

- **The Fundamentals:** A task is denoted by the 6-tuple $< p_i, e_i, r_i, d_i, \lambda_i, \eta_i >$ as described in Section 3.1. The timing constraints specified in Equations 3.1 through 3.6 must be satisfied.

- **Asynchronous Communication:** Tasks communicate with each other by sending and receiving data or messages. The frequencies of sending and receiving tasks of a communication can be different. As a consequence, communications between tasks may cross the task periods. When such asynchronous communications occur, the semantics of undersampling is assumed. It means, when two tasks of different frequencies are communicating, the communications are scheduled at the lower rate. For example, if task A (of 10HZ) sends a message to task B (of 5HZ), then in every $200ms$, one of two instances of task A has to send a message to one instance of task B. If the sending and receiving tasks are assigned to the same processor, then a local communication occurs. We assume the time taken by a local communication is negligible. When an interprocessor communication (IPC) occurs, the communication must be scheduled on the communications network between the end of the sending task execution and the start of the receiving task execution. The transmission time required to communicate the message i over the network is denoted by μ_i.

- **Communication Latency:** Each communication is associated with a communication latency which specifies the maximum separation between the start time of the sending task and the completion time of the receiving task.

- **Cyclic Dependency:** Research on the allocation problem has usually focused on acyclic task graphs [Ram90, HS92]. The use of acyclic task graphs excludes the possibility of specifying the cyclic dependency among tasks. For example, consider the following situation in which one instance of task A can not start its execution until it receives data from the last instance of task B. At the instance task A finishes its execution, it sends data to the next instance of task B. Since tasks A and B are periodic, the communication pattern goes on throughout the lifetime of the application. To be able to accommodate this situation, we take cyclic dependency into consideration.

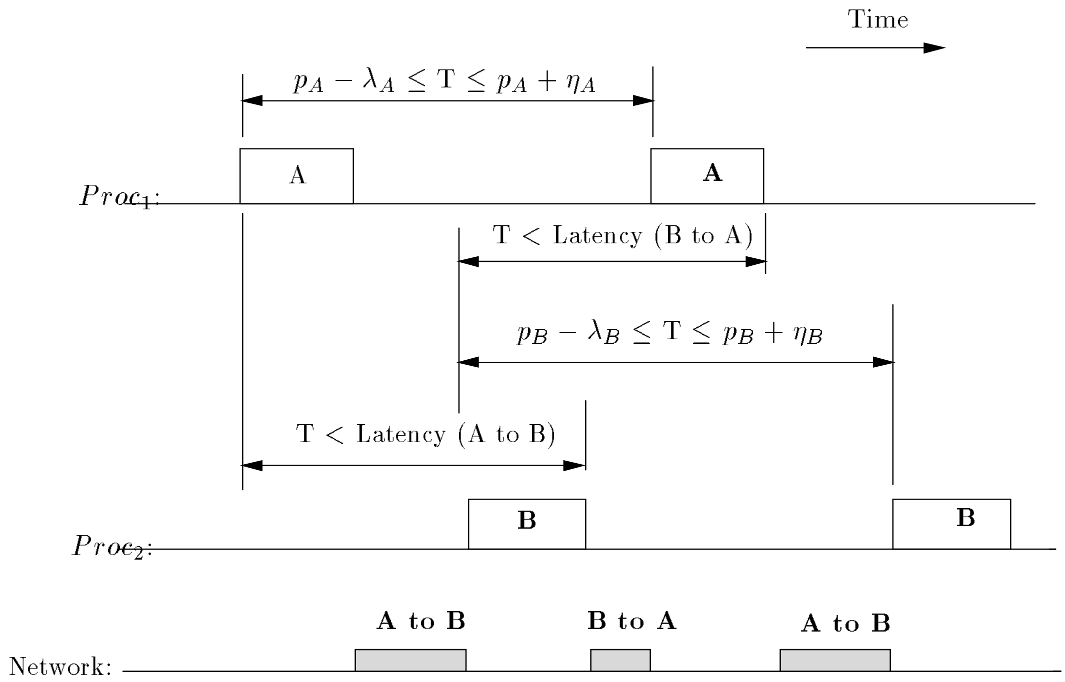


Figure 4.1: Relative Timing Constraints

The timing constraints described above are shown in Figure 4.1. For periodic tasks A and B, the start times of each and every instance of task execution and communication are pre-scheduled such that (1) the execution intervals fall between $p - \lambda$ and $p + \eta$ and (2) the time window between the start time of the sending task and the completion time of the receiving task is less than the latency of the communication. In Figure 4.2, we illustrate examples of all possible communication patterns considered in this dissertation. The description of the communications in the task system is in the form of "*From sender-task-id (of frequency) To receiver-task-id (of frequency)*". If the sender frequency is n times of the receiver frequency and no cyclic dependency is involved, then one of every n instances of the sending task has to communicate with one instance of the receiving task. (Examples of this situation are shown in Figures 4.2.a.1 and 4.2.a.2. Likewise, for the case in which the receiver frequency is n times that of the sender frequency and no cyclic dependency is present, the patterns are shown in Figures 4.2.b.1 and 4.2.b.2. For an asynchronous communication, the sending (receiving) task in low frequency sends (receives) the message to (from) the *nearest* receiving (sending) task as shown in Figure 4.2.a (4.2.b). The cases where cyclic dependency is considered are shown in Figures 4.2.c and 4.2.d.

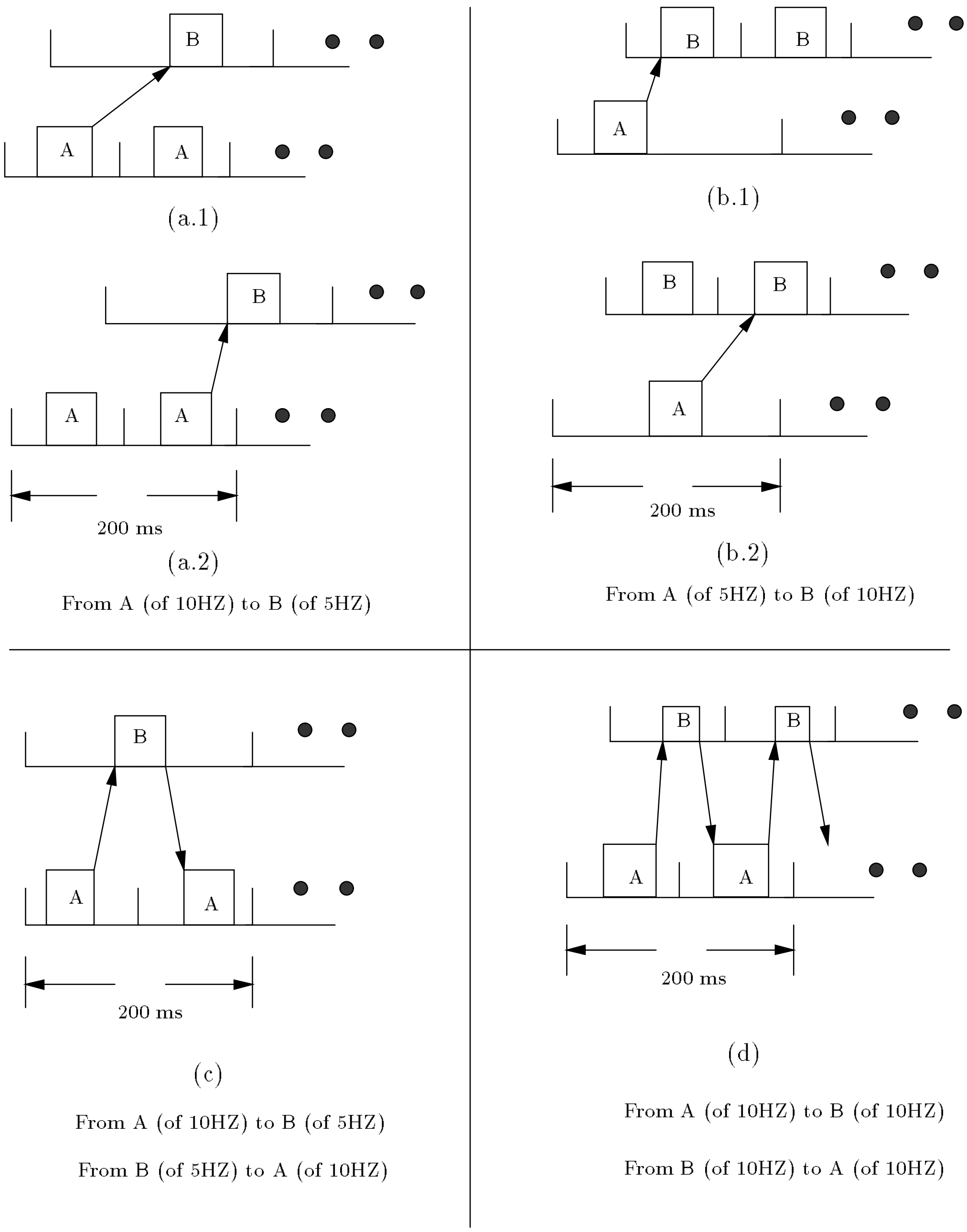


Figure 4.2: Possible Communication Patterns

4.1.2 System Model

A real-time multiprocessor system consists of a number of processors connected by a communications network. The execution of an instance on a processor is nonpreemptable. To provide predictable communication and to avoid contention for the communication channel at the run time, we make the following assumptions. (1) Each IPC occurs at the pre-scheduled time. (2) At most one communication can occur at any given time on the network.

4.1.3 Problem Formulation

We consider the static assignment and scheduling in which a task is the finest granularity object of assignment and an instance is the unit of scheduling. We applied the simulated annealing algorithm [KGV83] to solve the problem of real-time periodic task assignment and scheduling with hybrid timing constraints. In order to make the execution of instances satisfy the specifications and meet the timing constraints, we consider a scheduling frame whose length is the least common multiple (LCM) of all periods of tasks. Given a task set Γ and its communications C, we construct a set of task instances, I, and a set of communications, M. For each task $\tau_i \in \Gamma$, the task is extended to n_i instances, τ_i^1, τ_i^2, $\ldots$, and $\tau_i^{n_i}$, where $n_i = \mathrm{LCM}/p_i$. These n_i instances are included in I. Each communication $\tau_i \mapsto \tau_j \in C$ is extended to $\min(n_i, n_j)$[1] undersampled communications where $n_i = \mathrm{LCM}/p_i$ and $n_j = \mathrm{LCM}/p_j$. These multiple communications are included in M. The extension can be stated as follows.

- If $n_i < n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^1 \mapsto \tau_j^?$, $\tau_i^2 \mapsto \tau_j^?$, $\ldots$, and $\tau_i^{n_i} \mapsto \tau_j^?$.
- If $n_i > n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^? \mapsto \tau_j^1$, $\tau_i^? \mapsto \tau_j^2$, $\ldots$, and $\tau_i^? \mapsto \tau_j^{n_j}$.
- If $n_i = n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^1 \mapsto \tau_j^1$, $\tau_i^2 \mapsto \tau_j^2$, $\ldots$, and $\tau_i^{n_i} \mapsto \tau_j^{n_j}$.

A task ID with a superscript of question mark indicates some instance of the task. For example, $\tau_i^1 \mapsto \tau_j^?$ means that τ_i^1 communicates with some instance of τ_j. We describe how we assign the nearest instance for each communication in Section 4.3.1.

The problem can be formulated as follows. Given a set of task instance, I, its communications M, we find an assignment ϕ, a total ordering σ_m of all instances, and a total ordering σ_c of all communications to minimize

[1] Due to undersampling, when an asynchronous communication is extended to multiple communications, the number of multiple communications is the smaller number of sender and receiver instances.

$$
\begin{aligned}
E(\phi, \sigma_m, \sigma_c) &= \sum_{i,j} \delta(p_i - \lambda_i - s_i^{j+1} + s_i^j) + \sum_{i,j} \delta(s_i^{j+1} - s_i^j - p_i - \eta_i) \\
&+ \sum_{i,j} \delta(f_i^j - d_i^j) + \sum_{i,j,k,l} \delta(F(\tau_i^j \mapsto \tau_k^l, \sigma_c) - s_k^l) \\
&+ \sum_{i,j,k,l} \delta(f_k^l - s_i^j - \text{Latency } (\tau_i \text{ to } \tau_k)) \qquad (4.1)
\end{aligned}
$$

$$
\text{subject to } \; s_i^j \geq r_i^j \text{ and } \; S(\tau_i^j \mapsto \tau_k^l, \sigma_c) \geq f_i^j, \;\; \forall\, \tau_i^j \mapsto \tau_k^l,
$$

where

- s_i^j is the start time of τ_i^j under σ_m.
- f_i^j is the completion time of τ_i^j under σ_m.
- $r_i^j = p_i \times (j-1) + r_i$, and $d_i^j = p_i \times (j-1) + d_i$.
- $\delta(x) = 0$, if $x \leq 0$; and $= x$, if $x > 0$.
- $\phi(\tau_i)$ is the ID of processor which τ_i is assigned to.
- $\tau_i^j \mapsto \tau_k^l$ is the communication from τ_i^j to τ_k^l. If $\phi(\tau_i) = \phi(\tau_k)$, then $\tau_i^j \mapsto \tau_k^l$ is a local communication.
- $S(c, \sigma_c)$ is the start time of communication c on the network under σ_c.
- $F(c, \sigma_c)$ is the completion time of communication c on the network under σ_c.

The minimum value of $E(\phi, \sigma_m, \sigma_c)$ is zero. It occurs when the executions of all instances meet the jitter constraints and all communications meet their latency constraints. A feasible multiprocessor schedule can be obtained by collecting the values of s_i^j and f_i^j, $\forall\, i$ and j. Likewise, a feasible network schedule can be obtained from $S(c, \sigma_c)s$ and $F(c, \sigma_c)s$.

Since the task system is asynchronous and the communication pattern could be in the form of cyclic dependency, we solve the problem of finding a feasible solution $(\phi, \sigma_m, \sigma_c)$ by exploiting the cyclic scheduling technique and embedding the technique into the simulated annealing algorithm.

4.2 The Approach

The basic approach of scheduling a set of synchronous periodic tasks is to consider the execution of all instances within the scheduling frame whose length is the LCM of all periods. The release

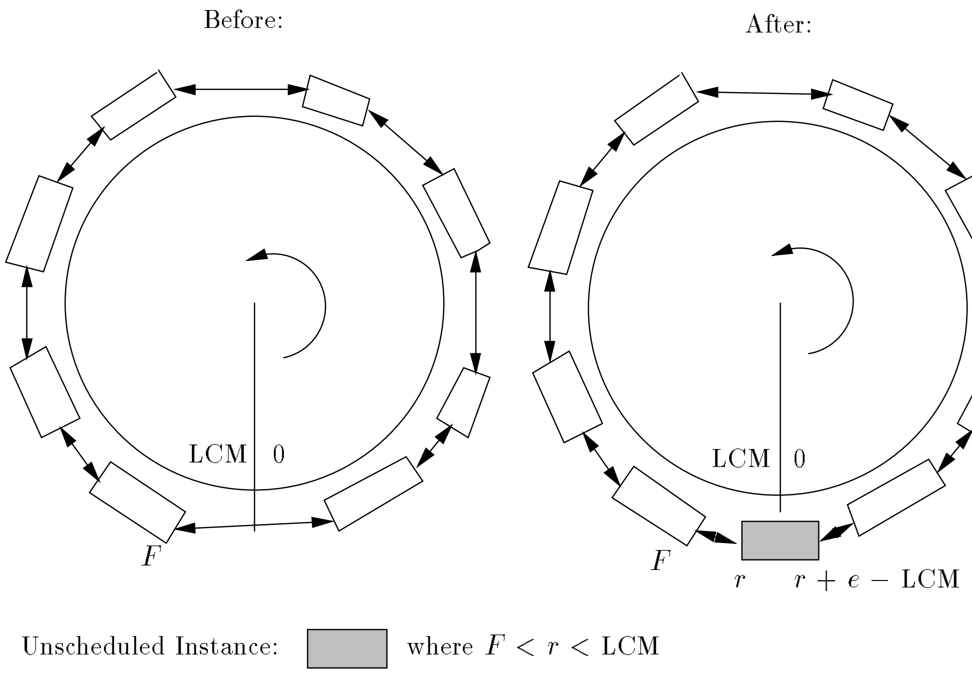


Figure 4.3: Insertion of a new time slot

times of the first periods of all tasks are zero. As long as one instance is scheduled in each period within the frame and these executions meet the timing constraints, a feasible schedule is obtained. In a feasible schedule, all instances complete the executions before the LCM.

On the other hand, in asynchronous task systems, as depicted in Figure 4.2 in which the LCM is $200ms$, the periods of the two tasks are out of phase. It is possible that the completion time of some instance in a feasible schedule exceeds the LCM. To find a feasible schedule for such an asynchronous system, a technique of handling the time value which exceeds the LCM is proposed.

4.2.1 Cyclic Scheduling Technique

The cyclic scheduling technique uses the linked list structure described in Section 3.4. Without loss of generality, we assume the minimum release time among the first periods of all tasks is zero. We keep a linked list for each processor and a separated list for the communication network. Each element in the list represents a time slot assigned to some instance or communication. The fields of a time slot of some processor p: (1) *task id i* and *instance id j* indicate the identifier of the time slot. (2) *start time st* and *finish time ft* indicate the start time and completion time of τ_i^j respectively. (3) *prev ptr* and *next ptr* are the pointers to the preceding and succeeding time slots respectively. The list is arranged in an increasing order of *start_time*. Any two time slots are nonoverlapping.

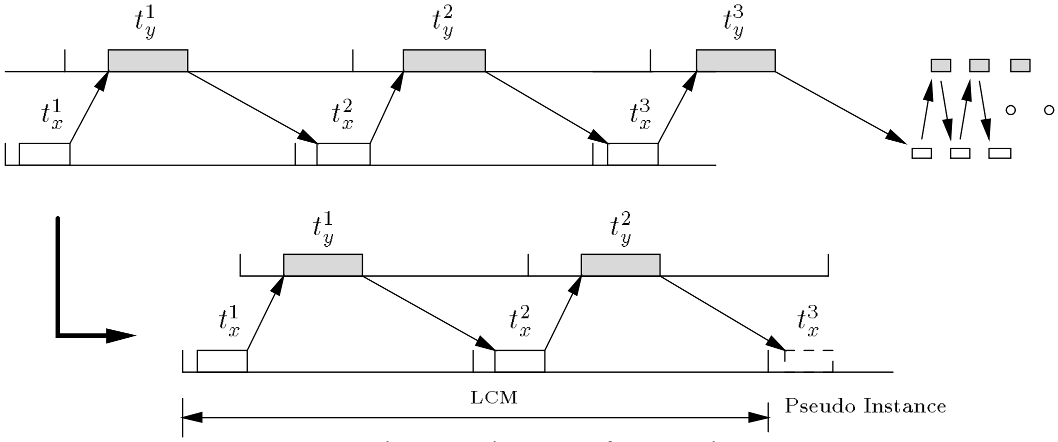


Figure 4.4: The introduction of a pseudo instance

Since the execution of an instance is nonpreemptable, the time difference between $start\_time$ and $finish\_time$ equals the execution time of the task.

Recurrence

Given any solution point $(\phi, \sigma_m, \sigma_c)$, we construct the schedule by inserting time slots in the linked lists. Let σ_m: *task_id* $\times$ *instance_id* $\rightarrow$ *integer*. The insertion of a time slot for τ_i^j precedes that for τ_k^ℓ if $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$.

Recall that Equations 3.23 and 3.24 specify the bounds of the scheduling window for a task instance. Due to the communications, $est(\tau_i^j)$ in Equation 3.23 may not be the earliest time for τ_i^j. We define the *effective start time* as the time when (1) the hybrid constraints are satisfied and (2) τ_i^j receives all necessary data or messages from all the senders.

Given the effective start time r and the assignment of τ_i (i.e. $p = \phi(\tau_i)$), a time slot of processor p is assigned to τ_i^j where $start\_time \geq r$ and $finish\_time - start\_time = e_i$. Note that we have to make sure that the new time slot does not overlap existent time slots. Since (1) the executions of all instances within one scheduling frame recur in the next scheduling frame and (2) it is possible that the time slot for some instance is over LCM, we subtract one LCM from the $start\_time$ or $finish\_time$ if it is greater than LCM. It means the time slot for this task instance will be wrapped to the beginning of the schedule. As shown in Figure 4.3 The $start\_time$ of the new slot is r while the completion time is $r + e -$LCM.

4.2.2 Pseudo Instances

As stated in Section 4.1, we consider the communication pattern in which cyclic dependency exists among tasks. Given a set of tasks, Γ, a set of task instances, I, a set of communications, C, and any solution point, $(\phi, \sigma_m, \sigma_c)$, we introduce pseudo instances. For any task τ_x, if there exists a task τ_y, in which (1) $\sigma_m(\tau_x^i) < \sigma_m(\tau_y^i)$, $\forall\ i$, (2) $n_x = n_y$, and (3) $\tau_x \mapsto \tau_y \in C$ and $\tau_y \mapsto \tau_x \in C$, then a pseudo instance $\tau_x^{n_x+1}$ is added to I. A pseudo instance is always a receiving instance. No insertion of time slots for pseudo instances is needed. For a pseudo instance, only the effective start time is needed. The effective start time of a pseudo instance $\tau_x^{n_x+1}$ in the constructed schedule based on $(\phi, \sigma_m, \sigma_c)$ is checked to see whether it is less than LCM + s_x^1 or not. If yes, then the execution of τ_x^1 for the next scheduling frame may start at $LCM + s_x^1$ which is exactly one LCM away from the execution of τ_x^1 for the current scheduling frame. A graphical illustration of the introduction of pseudo instance to address the synchronous communications of cyclic dependency is given in Figure 4.4 in which $n_x = 2$.

As for the asynchronous communications of cyclic dependency, no pseudo instances are needed. For example, if both $\tau_x \mapsto \tau_y$ and $\tau_y \mapsto \tau_x$ exist and $n_x = n_y \times n$, then for each τ_y^j, where $j = 1$, $2, \ldots, n_y$, we find a sending instance $\tau_x^i \in I$ and a receiving instance $\tau_x^k \in I$ such that (1) $f_x^i \leq s_y^j$, (2) $f_y^j \leq s_x^k$, and (3) $\tau_x^i \mapsto \tau_y^j$ and $\tau_y^j \mapsto \tau_x^k$ are the communications. The relationship between i, j, and k can be stated as

$$(j-1) \times n < i < k \leq j \times n. \tag{4.2}$$

A graphical illustration can be found in Figure 4.5. In the example, the values of i, j, k, and n are 6, 2, 8, 4 respectively. The communications $\tau_x^6 \mapsto \tau_y^2$ and $\tau_y^2 \mapsto \tau_x^8$ are scheduled before and after the scheduling of τ_y^2 respectively.

4.3 The Simulated Annealing Algorithm

Kirkpatrick *et al.* [KGV83] proposed a simulated annealing algorithm for combinatorial optimization problems. Simulated annealing is a global optimization technique. It is derived from the observation that an optimization problem can be identified with a fluid. There exists an analogy between finding an optimal solution of a combinatorial problem with many variables and the slow cooling of a molten metal until it reaches its low energy ground state. Hence, the terms about energy function, temperature, and thermal equilibrium are mostly used. During the search of an optimal solution, the algorithm always accepts the downward moves from the current solution point to the points of lower energy values, while there is a small chance of accepting upward moves to

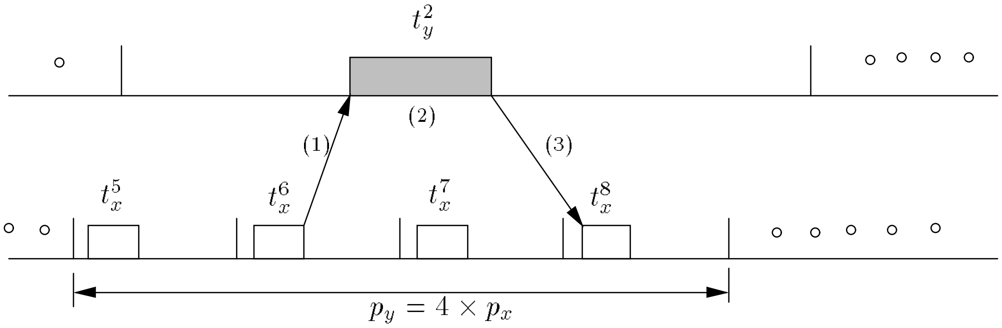


Figure 4.5: Asynchronous communications

the points of higher energy values. The probability of accepting an uphill move is a function of current temperature. The purpose of hill climbing is to escape from a local optimal configuration. If there are no upward or downward moves over a number of iterations, the thermal equilibrium is reached. The temperature then is reduced to a smaller value and the searching continues from the current solution point. The whole process terminates when either (1) the lowest energy point is found or (2) no upward or downward jumps have been taken for a number of successive thermal equilibrium.

The structure of the simulated annealing (SA) algorithm is shown in Figure 4.7. The first step of the algorithm is to randomly choose an assignment ϕ, a total ordering of instances within one scheduling frame, σ_m, and a total ordering of communications for the instances, σ_c. A solution point in the search space of SA is a 3-tuple (ϕ,σ_m,σ_c). The energy of a solution point is computed by equation (4.1). For each solution point P which is infeasible, (i.e. E_p is nonzero), a neighbor finding strategy is invoked to generate a neighbor of P. As stated before, if the energy of the neighbor is lower than the current value, we accept the neighbor as the current solution; otherwise, a probability function (i.e. $exp(\frac{E_p - E_n}{T})$) is evaluated to determine whether to accept the neighbor or not. The parameter of the probability function is the current temperature. As the temperature is decreasing, the chance of accepting an uphill jump (i.e. a solution point with a higher energy level) is smaller. The inner and outer loops are for thermal equilibrium and termination respectively. The number of iterations for the inner loop is also a function of current temperature. The lower the temperature, the bigger the number. Several approaches on how to model the numbers of iterations and how to assign the number for each temperature have been proposed [LH91]. In this dissertation, we consider a simple incremental function. Namely, $N = N + \Delta$ where N is the number of iterations and Δ is a constant. The termination condition for the outer loop is $E_p = 0$. After thermal equilibrium is reached at a temperature, the value of T is decreased. Different approaches of temperature decrease function have been proposed [KGV83]. Here we consider a simple multiplication function (i.e. $T = T \times \alpha$, where $\alpha < 1$).

4.3.1 Evaluation of Energy Value for a Solution Point (ϕ, σ_m, σ_c)

The first step to compute the energy value, stated in Equation 4.1, is to construct multi-processor schedules and a network schedule. The start time and completion time information of each task instance and communication can be collected from these schedules. The time information is then used to compute the energy value.

The construction of the schedules is characterized by the priority assignment of the task instances in the set. The priority assignment algorithm determines the scheduling order among all task instances. When a task instance is chosen to be scheduled, the incoming communications of the instance are scheduled first and then the task instance itself. After all the task instances have been scheduled, the scheduling of the outgoing communications is performed. An algorithmic description about how to compute the energy value for a solution point is given in Figure 4.6. Note that a communication is an incoming communication to a task instance if the frequency of the receiving task instance is equal to or less than that of the sending task instance. For example, $\tau_k^?$ $\mapsto \tau_i^j$ and $\tau_k^j \mapsto \tau_i^j$ are incoming communications to τ_i^j. On the other hand, if the sender frequency is less than the receiver frequency, then the communication is an outgoing communication. (e.g. $\tau_k^j \mapsto \tau_i^?$ is the outgoing communication of τ_k^j).

Priority Assignment of Task Instances: σ_m

In Sections 3.5 and 3.6, we presented the SLsF algorithm and the performance evaluation. The results showed that SLsF outperforms SPF and SJF. In this chapter we use SLsF as the priority assignment algorithm for the task instances in I.

Formally, if $lst(\tau_i^j) < lst(\tau_k^\ell)$, then $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$. And the insertion of a time slot for τ_i^j precedes that for τ_k^ℓ if $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$. The time-based scheduling algorithm for a task instance in Section 3.4 is used to find a time slot for a task instance once the *effective start time* is given. We define the *effective start time* of a task instance as the earliest start time when the incoming communications are taken into account. Let t be the maximum completion time among all the incoming communications of a task instance, then the *effective start time* of the task instance is set to the larger value among t and *est* (as stated in Equation 3.23).

Scheduling the Incoming Communications

There are two kinds of incoming communications. The first kind is called the synchronous communication in which the frequencies of the sender and receiver are identical. The other kind is called the asynchronous communication in which the sending task instance is associated with a question mark. For an asynchronous communication, we have to decide which instance of the sending task should communicate with the receiving task instance. The approach we take is to find the nearest instance of the sending task. By finding the nearest instance, the time difference between start time of the receiving instance and the completion time of the sending instance is the smallest. The chance of violating the latency constraint of a communication will be the smallest.

The nearest instance of a sending task can be found using the following method. Given an incoming communication $\tau_k^? \mapsto \tau_i^j$, and the *effective start time* of τ_i^j, eft we search through the linked list of processor $\phi(\tau_k)$ up to time eft. If there is some instance of τ_k, say τ_k^ℓ, whose completion time is the latest among all scheduled instances of τ_k, then the nearest instance is found. Otherwise, we continue to search through the linked list until an instance of τ_k is found. We set the *effective start time* of the communication to be the completion time of the found instance. We also erase the question mark such that $\tau_k^? \mapsto \tau_i^j$ is changed to $\tau_k^\ell \mapsto \tau_i^j$. For the synchronous communication, the *effective start time* of the communication is simply assigned as the *finish time* of the sending task instance.

The scheduling of the communication is done by inserting a time slot in the linked list for the communications network. The *start time* of the time slot can not be earlier than the *effective start time* of the communication. Once the time slot is inserted, we check the *effective start time* of τ_i^j to make sure that it is not less than the *finish time* of the time slot. If it is, the *effective start time* of τ_i^j is updated to be the *finish time* of the time slot.

If a task instance has more than one incoming communication, the scheduling order among these communications is based on their latency constraints. The larger the latency value, the earlier the communication is scheduled. The incoming communication with the tightest latency constraint is scheduled last. The *effective start time* of the receiving task instance is constantly updated by the scheduling of the incoming communications. It is possible that the scheduling of the later incoming communication increases the *effective start time* of the receiving task instance and makes the early scheduled communication violate its latency constraint if the constraint is tight.

Scheduling the Outgoing Communications

Scheduling of the outgoing communications for the whole task set is performed after all the task instances have been scheduled. The scheduling order among these communications is based on the finish times of the sending task instances. The task instance with the smallest finish time is considered first. When a task instance is taken into account, all of its outgoing communications are scheduled one by one according to their latency constraints. The communication with the tightest latency constraint is scheduled first.

Given an outgoing communication $\tau_i^j \mapsto \tau_k^?$, and the finish time of τ_i^j, f_i^j, the *effective start time* of the communication is set to be f_i^j. Based on the *effective start time*, a time slot in inserted for this communication. Then the nearest instance of receiving task can be found based on the *finish time* of the time slot.

For the example shown in Figure 4.5, the incoming communication marked with "(1)" is scheduled before the scheduling of τ_y^2. The sixth instance of τ_x is chosen as the nearest instance. As for the outgoing communication marked with "(3)", it is scheduled after the scheduling of τ_x^5, τ_x^6, τ_x^7, and τ_x^8. In this example, τ_x^8 is the nearest instance of the outgoing communication.

4.3.2 Neighbor Finding Strategy: ϕ

The neighbor finding strategy is used to find the next solution point once the current solution point is evaluated as infeasible (i.e. energy value is nonnegative). The neighbor space of a solution point is the set of points which can be reached by changing the assignment of one or two tasks. There are several modes of neighbor finding strategy.

- Balance Mode: Randomly move a task from the heavily-loaded processor to the lightest-loaded processor. This move tries to balance the workload of the processors. By balancing the workload, the chance to find a neighbor with a lower energy value is higher.

- Swap Mode: Randomly choose two tasks τ_i and τ_j on processors p and q respectively. Then change ϕ by setting $\phi(\tau_i) = q$ and $\phi(\tau_j) = p$.

- Merge Mode: Pick two tasks and move them to one processor. By merging two tasks to a processor, the workload of the processor is increased. There is an opportunity of increasing the energy level of the new point by increasing the workload of the processor. The purpose of the move is to perturb the system and allow the next move to escape from the local optimum.

- Direct Mode: When the system is in a low-energy state, only a few tasks violate the jitter or latency constraints. Under such a circumstance, it will be more beneficial to change the assignment of these tasks instead of randomly moving other tasks. From the conducted experiments, we find that this mode can accelerate the searching of a feasible solution especially when the system is about to reach the equilibrium.

The selection of the appropriate mode to find a neighbor is based on the current system state. Given a randomly generated initial state (i.e. solution point), the workload discrepancy between the processors may be huge. In the early stage of the simulated annealing, the balance mode is useful to balance the workload. After the processor workload is balanced, the swap mode and the merge mode are frequently used to find a lower energy state until the system reaches near-termination state. In the final stage of the annealing, the direct mode tries to find a feasible solution. The whole process terminates when a feasible solution is found in which the energy value is zero, or when it is not possible to improve over the current solution.

4.4 Experimental Results

We implement the algorithm as the framework of the allocator on MARUTI [GMK+91] [MSA92] [SdSA94], a real-time operating system developed at the University of Maryland, and conducted extensive experiments under various task characteristics. The tests involve the allocation of real-time tasks on a homogeneous distributed system connected by a communication channel.

To test the practicality of the approach and show the significance of the algorithm, we consider a simplified and sanitized version of a real problem. This is derived from actual development work, and is therefore representative of the scheduling requirements of an actual avionics system. The Boeing 777 Aircraft Information Management System (AIMS) is to be running on a multiprocessor system connected by a SafeBus (TM) ultra-reliable bus. The problem is to find the minimum number of processors needed to assign the tasks to these processors and generate the schedule. The objective is to develop an off-line non-preemptable schedule for each processor and one schedule for the SafeBus.

The AIMS consists of 155 tasks and 951 communications between these tasks. The frequencies of the tasks vary from 5HZ to 40HZ. The execution times of the tasks vary from $0ms$ to $16.650ms$. Both λ_i and η_i of a task τ_i are 500μs. The smallest-least-start-time-first (SLsF) scheduling algorithm is used. Tasks communicate with others asynchronously and in cyclic dependency. The transmission times for communications are in the range from 0μs to 447.733μs. The latency constraints of the

	10_Proc	9_Proc	8_Proc	7_Proc	6_Proc
Exec_Time (Sec)	2369	5572	19774	36218	78647
$= Hr : Min : Sec$	0:39:29	1:32:52	5:29:34	10:03:38	21:50:47

Table 4.1: The execution times of the AIMS with different number of processors

communications vary from $68.993ms$ to $200ms$. The LCM of these 155 tasks is $200ms$. When the whole system is extended, the total number of task instances within one scheduling frame is 624 and the number of communications is 1580. The specification of the Boeing 777 AIMS is listed in Appendix A. Note that the AIMS system is expressed in the form of "From <sender-id> <frequency> HZ <sender-execution-time> ms to <recv-id> length <transmission-time> us latency <latency-value> us".

For such a real problem size, pre-analysis is necessary. We calculate the resource utilization index to estimate the minimum number of processors needed to run AIMS. The index is defined as

$$\frac{\sum_{i=1}^{155}(e_i \times n_i)}{LCM}$$

where e_i is the execution of task τ_i and $n_i = \frac{LCM}{p_i}$. The obtained index for AIMS is 5.14. It means there exists no feasible solution for the AIMS if the number of processors in the multiprocessor system is less than 6.

The number of processors which the AIMS is allowed to run on is a parameter to the scheduling problem. We solve the AIMS scheduling problem by starting with 10 processors. After a feasible solution is found, we decrease the number of processors by one and solve the whole problem again. We ran the algorithm on a DECstation 5000. The execution time for the AIMS scheduling problem with different numbers of processors is summarized in Table 4.1. The algorithm is able to find a feasible solution of the AIMS with six processors which is the minimum number of processors according to the resource utilization index. The time to find such a feasible solution is less than one day (approximately 22 hours). A complete solution to the AIMS scheduling problem with six processors is shown in Appendix B.

4.4.1 Discussions

For feasible solutions of the AIMS with various numbers of processors, we calculate the processor utilization ratio (PUR) of each processor. The processor utilization ratio for a processor p is defined as

$$\frac{\sum_{\phi(\tau_i)=p}(e_i \times n_i)}{LCM}.$$

The results are shown in Figure 4.8. The ratios are sorted into a non-decreasing order given a fixed number of processors. The algorithm generates the feasible solutions for the AIMS with 6, 7, 8, 9 and 10 processors respectively. For example, for the 6-processor case, the PURs for the heaviest-loaded and lightest-loaded processors are 0.91 and 0.76 respectively. For the 10-processor cases, the PURs are 0.63 and 0.28 respectively. We find that the ratio difference between the heaviest-loaded processor and the lightest-loaded processor in the 6-processor case is smaller than those in other cases. It means the chance for finding a feasible solution to a more load balanced allocation is higher when the number of processors is smaller.

The detailed schedules for the 6-processor case are shown in Figure 4.9. The results are shown on an interactive graphical interface which is developed for the design of MARUTI. The time scale shown in Figure 4.9 is 100μs. So the LCM is shown as 2000 in the figure. (i.e. $2000 \times 100\mu\text{s} = 200ms$.) This solution consists of seven off-line non-preemptive schedules: one for each processor and one for the SafeBus (TM). Each of these schedules will be one LCM long where an infinite schedule can be produced by repeating these schedules indefinitely. Note that the pseudo instances are introduced to make sure the wrapping around at the end of the LCM-long schedules should satisfy the latency and next-execution-interval requirements across the point of wrap-around. The pseudo instances are not shown in Figure 4.9.

We have presented a general approach for scheduling a set of periodic tasks which communicate with each other. The approach can be extended easily to address additional requirements. For example, the inclusion of resource and memory constraints into the problem can be done by modifying the neighbor-finding strategy. Once a neighbor of the current point is generated, it is checked to ascertain that the constraints on memory etc. are met. If not, the neighbor is discarded and another neighbor is evaluated.

```
Given a solution point P = (φ, σ_m, σ_c)
While there is some unscheduled task instance do
    Find the next unscheduled instance. /* By the SLsF algorithm */
    Let the instance be τ_i^j.
    Sort all the incoming communications of τ_i^j based on
        the latency values into a descending order.
    Schedule each incoming communication starting from
        the biggest-latency one to the tightest-latency one.
    Schedule the instance τ_i^j.
End While.
Mark each instance as un-examined.
While there is some un-examined task instance do
    Find the next un-examined task instance. /* By the finish times */
    Sort all the outgoing communications of the task instance based
        on the latency values into an increasing order.
    Schedule each outgoing communication starting from
        the tightest-latency one to the biggest-latency one.
    Mark the task instance examined.
End While.
Collect the start time and finish time informations for each task
    instance and communication.
Compute the energy value using Equation 4.1.
```

Figure 4.6: The pseudo code for computing the energy value

```
Choose an initial temperature T
Choose randomly a starting point P = (φ, σ_m, σ_c)
E_p := Energy of solution point P
if E_p = 0 then
      output E_p and exit /* E_p = 0 means a feasible solution */
end if
repeat
      repeat
            Choose N, a neighbor of P
            E_n := Energy of solution point N
            if E_n = 0 then
                  output E_n and exit
                  /* E_n = 0 means a feasible solution */
            end if
            if E_n < E_p then
                  P := N
                  E_p := E_n
            else
                  x := (E_p − E_n) / T
                  if e^x ≥ random(0,1) then
                        P := N
                        E_p := E_n
                  end if
            end if
      until thermal equilibrium at T
      T := α × T (where α < 1)
until stopping criterion
```

Figure 4.7: The structure of simulated annealing algorithm.

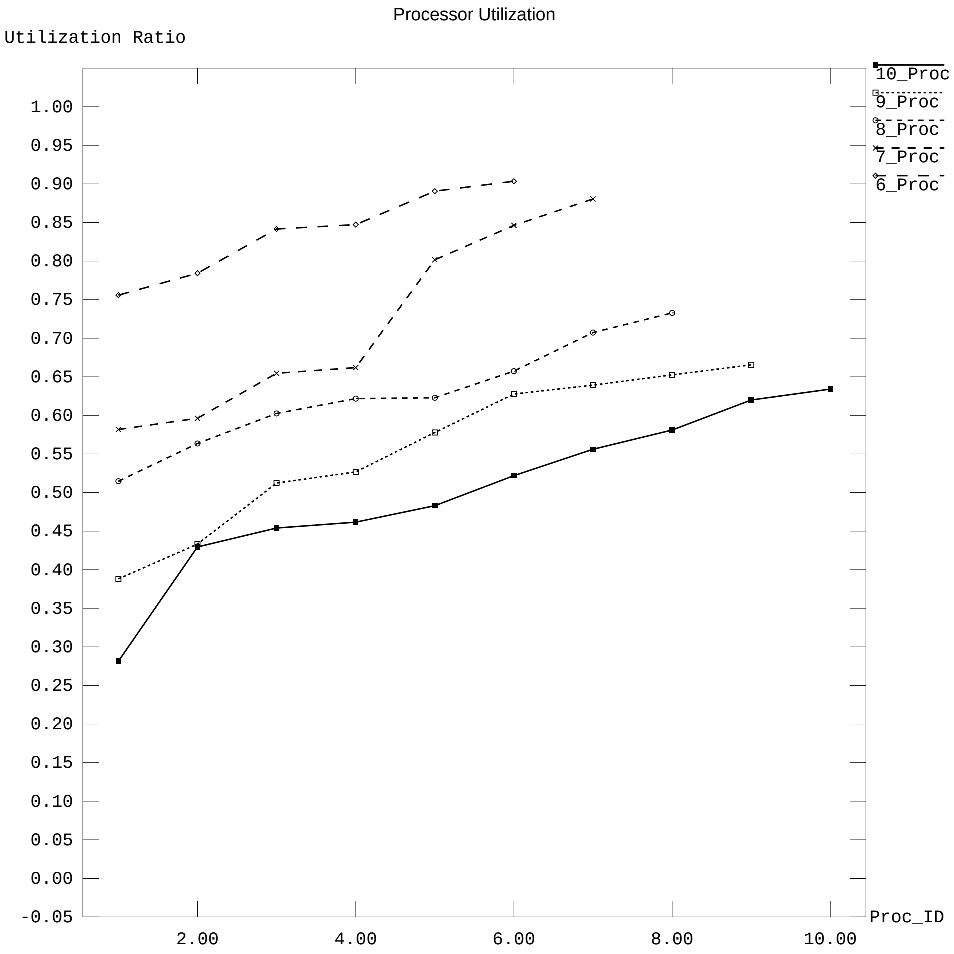


Figure 4.8: Processor Utilization Ratios for different cases

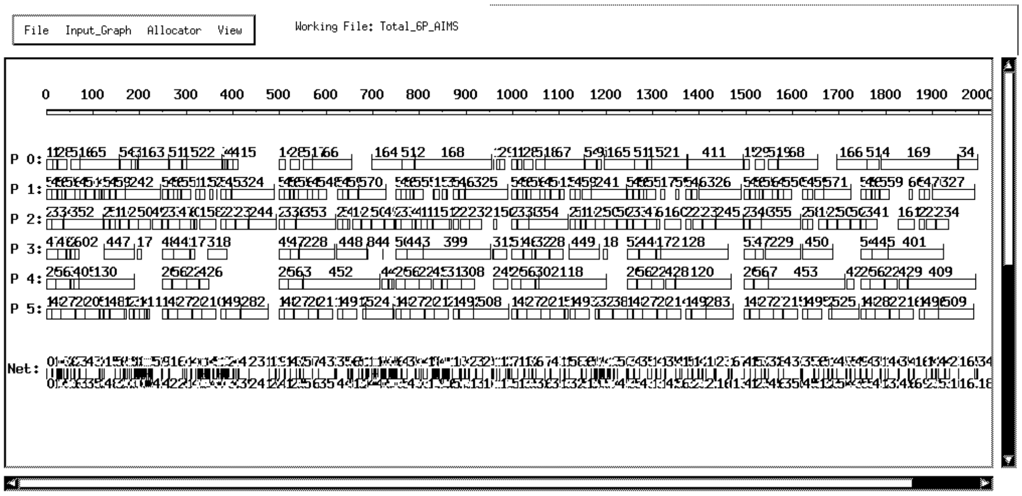


Figure 4.9: The Allocation Results and Schedules for AIMS with 6 processors

Chapter 5

Schedulability-Oriented Replication in Real-Time Systems

In Chapters 3 and 4, we considered the allocation and scheduling of the periodic tasks in real-time systems, in which we assume that a task is the unit of allocation and scheduling. The inter-task dependency is accomplished via communications. In this chapter, we consider each task as a compound object.

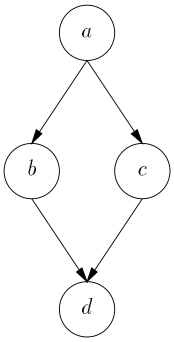
A periodic real-time task can be decomposed into several modules and intermodule communications (IMCs), such that modules of tasks communicate with one another via IMCs. In allocating modules of tasks, we have the option of restricting ourselves to having only one copy of each module. We call this the *assignment problem.* If, on the other hand, a module may have multiple copies, we call this general problem the *replication problem.* In this chapter, we consider the replication problem and present algorithms to find optimally feasible solutions for replicating modules of real-time periodic tasks.

5.1 Introduction

For real-time applications, the objective function of the allocation problem can be in terms of bottleneck processor time [CL87], system hazard [PS89], the degree of reliability [SW89] or the degree of schedulability, etc. For real-time applications where the degree of schedulability is the most critical concern, the essential solution is to find an allocation in which there exists a feasible schedule for the given task set. Ramamritham [Ram90] proposes a global view where the purpose of allocation should directly address the schedulability of processors and communication network. Taking the same global view, we have developed the allocation and scheduling algorithms based on the simulated annealing technique in Chapter 4. The primary focus of the work in Chapter 4 is on the assignment problem.

5.1.1 Schedulability-Oriented Replication

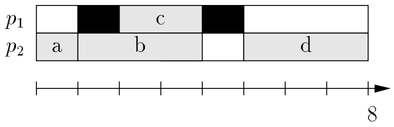
Traditionally, the objective of reliability-oriented replication problems is to increase the degree of fault tolerance [CC90, LAMS92]. If some processors in the distributed system fail, the application may still survive using other copies. In such a model, a module has to communicate with multiple copies of other modules. As a consequence, the degree of schedulability of a reliability-oriented replication problem will be smaller than that of the assignment problem and it is harder for the tasks to meet their deadlines. In this chapter, we consider a *schedulability-oriented* replication problem whose objective is to maximize the degree of schedulability. We adopt the communication model, in which the replication of modules is not for the sake of fault tolerance but for increasing the degree of schedulability. In our model, each module may have more than one copy and a module may start its execution after receiving necessary data from a copy of each of its predecessors. Clearly, in a heterogeneous environment the computation time of a module depends on the processor on which it executes, and the communication time depends on the topology, communication medium, network scheduling algorithm, etc. When a module m is allowed to have only one copy in the system, the time taken by the IPC between m and its successors may be too long. Such a long delay postpones the execution of the successors and causes the task to miss its deadline. Sometimes it will be more beneficial if we replicate m onto multiple processors to reduce the IPC, and to fully utilize the available processors in the system. Such replication may obtain a higher degree of schedulability than an optimal assignment problem. An example illustrating this point is shown in Figure 5.1 in which the number of processors in the system is two and the computation times for modules of a task are listed. We assume the ready time for the task is 0 and the deadline is 7. The IPC between p_1 and p_2 for any two modules takes one unit and the IMC time on the same processor is zero. If module a is assigned to processor p_1 only, then the minimum finish time is 8 as shown in Gantt chart 1. The dark area in the Gantt charts indicates the IPCs. If module a is replicated to processors p_1 and p_2, then we see from Gantt chart 2 that the minimum completion time is 7. As depicted in Figure 5.1, the replication of module a on processor 1 eliminates the IPC between modules a and c. It enables the earlier start of the execution of module c, increases the degree of schedulability and makes the task meet its deadline.



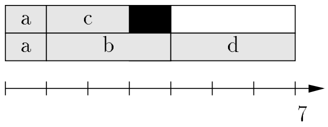
(a) A task graph

$\tilde{v}$ vectors	processor 1	processor 2
module a on	1	1
module b on	3	3
module c on	2	2
module d on	3	5

(b) Computation Vectors



(c) Gantt Chart 1



(d) Gantt Chart 2

Figure 5.1: An example to show how the replication can increase the degree of schedulability

5.1.2 Event-Based Semantics

The specifications of real-time applications have traditionally used code-based semantics. In code-based semantics, the timing constraints are established between blocks of code. Modules of tasks are associated with ready times and deadlines. Recently, Gerber and Hong [GH93] presented a time-constrained event language (TCEL) in which the timing semantics are based on the constraints relationships between observable events. In TCEL, the degree of schedulability may be increased because the timing constraints are only imposed by observable events, and the technique of program slicing is used to decompose the unschedulable codes into schedulable modules. In this chapter, we consider event-based semantics. We assume that 'SEND' and 'RECEIVE' are observable events in the system. Since each IMC is carried out by SEND-RECEIVE, it implies that each IMC is associated with a ready time and a deadline.

Event-based semantics gives a clear separation between scheduling of IMCs and that of modules. Such separation can increase the degree of schedulability. Consider the task graph shown in Figure 5.2, where we contrast the flow chart of a program using code-based semantics with that of the same program using event-based semantics. Each C_i represents a piece of code of the program. In Figure 5.2(a), code-based semantics implies that modules are associated with ready times and deadlines. For this semantics it is usually assumed that the IMC is carried out immediately after the execution of the sender module. For example, consider Send1 which is ready to perform right after the execution of C_1. It is scheduled on the network after the completion of $M1$.

In Figure 5.2(b), instead of imposing timing constraints on each module, event-based semantics imposes the timing constraints on IMCs. We observe that the scheduling of IMCs on the network not only has to meet timing constraints of IMCs but also has to be consistent with the execution of codes of modules. For example, Send1 can not be performed on the network until the completion of C_1. Such a *consistency check*[1] is enforced on each IMC in the event-based semantics. Note that Send1 can be performed at any time after the ready time and it may be performed before the completion time of $M1$.

5.1.3 Main Results

To solve the schedulability-oriented replication problem, we develop a replication technique and embed the technique in a simulated annealing algorithm. The function of the technique consists

[1]The programming-language level support for the consistency check could be in the form of "Not until e_1 do Send1", where e_1 is the execution time for C_1. Given an allocation and a schedule, we apply a consistency check to see if $T_s(Send1) \geq e_1 + \text{start\_time}(M1)$, where $T_s(Send1)$ is the start time of Send1.

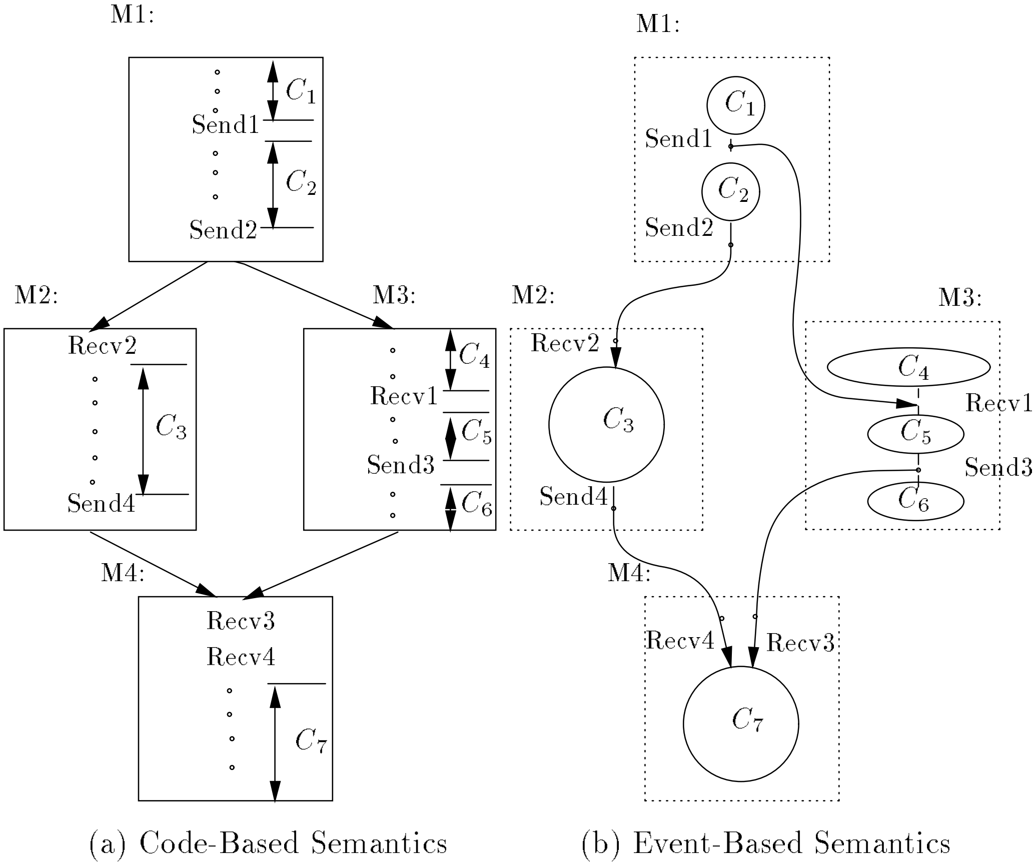


Figure 5.2: The flow charts of a task graph

of two phases, namely (1) identifying bottleneck IPCs and (2) eliminating bottleneck IPCs. Given an allocation and a schedule for modules and IMCs, we say that the bottleneck IPCs are the communications which make the schedule infeasible. Once the bottleneck IPCs are identified, we replicate the sender modules to eliminate the bottleneck IPCs. Experimental results show that such replication leads to a higher degree of schedulability.

In the remainder of this chapter, the task characteristics and the system model are described in Section 5.2. In Section 5.3, the replication problem is mathematically formulated. The replication technique and a simulated annealing algorithm are presented in Section 5.4, and some experimental results are given in Section 5.5.

5.2 Task and System Models

5.2.1 The Task Characteristics

Consider a set of periodic tasks, $\Gamma = \{ \tau_i \mid i = 1, \ldots, n \}$. The logical structure of task τ_i is expressed as a directed acyclic graph where the nodes represent the modules and the edges represent IMCs. The specifications of tasks and their corresponding graphs include the following:

- Task τ_i is associated with a period p_i, ready time r_i, and deadline d_i. All constituent modules of task τ_i have the same period p_i. The execution of a module can not be preempted. The earliest start time of task τ_i is r_i and the finish time can not exceed d_i. The values of r_i and d_i are relative to the start time of each period. We assume that each task starts its period at time zero.

- For each module m_j a computation vector $\tilde{v}_j[1 : n]$ is given. The value of $\tilde{v}_j[p]$ denotes the computation time needed for m_j to be executed on processor p. The computation times required for the same module on two processors can be identical or different depending upon the architecture of the system (i.e. homogeneous or heterogeneous).

- For each edge going from m_j to m_k (we denote it by $c_{j \to k}$), the amount of data to be transferred is specified as $\mu_{j \to k}$. The semantics of data transferring implies that if the sending module on one processor sends an amount of data to the receiving module on another processor then an IPC occurs. A local IMC occurs if the sending and receiving modules are on the same processor. We assume that the time required by a local IMC is negligible compared to the transit time needed for an IPC. Since IMC $\mu_{j \to k}$ is carried out by SEND-RECEIVE, the timing constraints are specified where $r_{j \to k}$ is the ready time for SEND and $d_{j \to k}$ is the deadline for RECEIVE.

5.2.2 System Assumptions

Consider the schedulability-oriented replication problem in which the purpose of replication is different from that of reliability-oriented replication. Our approach is based on the following assumptions:

- All the communications are via a message-passing mechanism. There is no shared memory in the system.

- Modules of a task can be assigned to different processors. Multiple copies of a module on different processors perform the same function and generate the same result as long as they receive the identical data through IMC. As a consequence, there is no need for coordination among the copies.
- The replication process of a module is done before the task starts to execute. There is no run-time overhead for maintaining replicas of a module in the system.

5.2.3 Communication Model

The communication model of schedulability-oriented replication is different from that of reliability-oriented replication. In a reliability-oriented replication problem, the objective is to increase the degree of fault tolerance. To detect fault and maintain data consistency, each module has to receive multiple copies of data from several module instances if its predecessor is replicated in more than one place.

The purpose of schedulability-oriented replication is to increase the degree of schedulability. Under such consideration, there is no need to enforce multiple copy communication between any two modules. We propose the *1-out-of-n* model [CA93] for schedulability-oriented replication. In the model, for each IMC $c_{j\to k}$, a module instance $m_{j@q}$ (i.e. the replica of m_j on processor q) may start its execution if it receives the data from *any one* module instance of its predecessor, module m_k.

5.2.4 System Model

The system architecture can be homogeneous or heterogeneous. Processors are interconnected by a communications network. The execution of modules on a processor is non-preemptive. To provide predictable communication and avoid the contention for the communication channel at the run time, we have the following assumptions. (1) Each IPC occurs at the pre-specified time according to the schedule. (2) At most, one communication can occur at any time on the network. (3) The network is associated with a nominal delay ND for transmitting one unit of data from one processor to another. When an IPC $c_{j\to k}$ occurs, the start time for $c_{j\to k}$ is either the ready time $r_{j\to k}$ if the network is available at that time, or the finish time of the preceding IPC scheduled on the network. The transit time required by an IPC $c_{j\to k}$ can be expressed by $t_{j\to k}$ where $t_{j\to k}$ is the time for transmitting IMC volume and $t_{j\to k} = ND \times \mu_{j\to k}$. When a local IMC occurs, the start time is the ready time. Each IMC is subject to a consistency check. For the recipient module m of

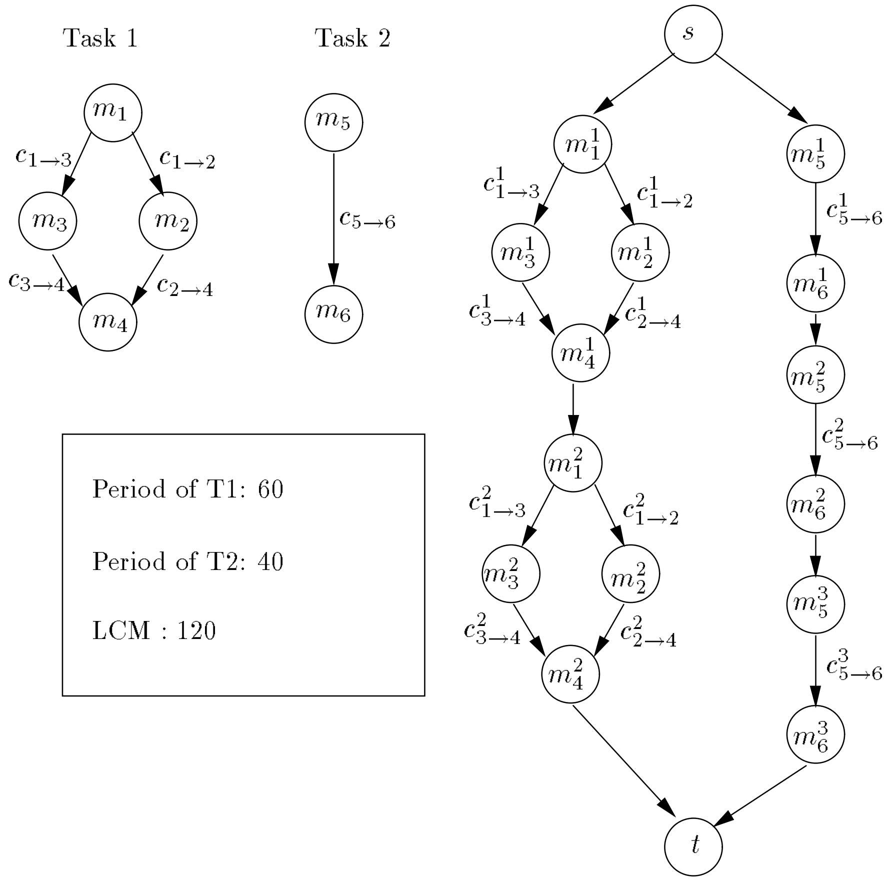


Figure 5.3: An example of Comprehensive Graph

an IMC, the earliest start time for m is the finish time of the data transmission locally or through the network.

5.3 Problem Formulation

We consider each task as a compound object composed of several modules and IMCs. In order to make the execution of modules satisfy the specifications and meet the deadlines, we consider the least common multiple (LCM) of all periods of tasks and repeat each task into multiple instances. Two pseudo nodes s and t are added to form a *comprehensive graph*. We denote the kth instance of module m_j by m_j^k and the kth instance of $c_{i\to j}$ by $c_{i\to j}^k$. An example is presented in Figure 5.3.

The schedulability-oriented module replication problem can be formulated as follows. Given a comprehensive graph G, we find an allocation ϕ, a processor schedule σ_m and a network schedule

σ_c for nodes (except the pseudo nodes) in the graph to minimize

$$E(\phi, \sigma_m, \sigma_c) = \sum_{i,j} \delta(f_i^j - d_i^j) + \sum_{i,j,k} \delta(T_f(c_{i\to j}^k, \sigma_c) - d_{i\to j}^k) \tag{5.1}$$
$$\text{subject to } \sum_{p=1}^{P} \phi(m_k^i, p) \geq 1, \quad \forall\ i,\ k,$$
$$\text{and } \ T_s(c_{i\to j}^k, \sigma_c) \geq r_{i\to j}^k, \quad \forall\ c_{i\to j}^k.$$

where

- f_i^j is the finish time of the jth instance of task τ_i in G under schedule σ_m.
- d_i^j is the deadline of the jth instance of task τ_i, i.e. $d_i^j = p_i \times (j-1) + d_i$.
- $\phi(m_k, p) = 1$, if m_k has a copy on processor p under allocation ϕ; and $= 0$, otherwise.
- $\delta(x) = 0$, if $x \leq 0$; and $= x$, if $x > 0$.
- $T_s(c, \sigma_c)$ is the start time for SEND of IMC c on the network under schedule σ_c.
- $T_f(c, \sigma_c)$ is the finish time for RECEIVE of IMC c on the network under schedule σ_c.
- $r_{i\to j}^k$ is the earliest start time for SEND of $c_{i\to j}^k$, i.e. $r_{i\to j}^k = p_i \times (k-1) + r_{i\to j}$.
- $d_{i\to j}^k$ is the deadline for RECEIVE of $c_{i\to j}^k$, i.e. $d_{i\to j}^k = p_i \times (k-1) + d_{i\to j}$.

The minimum value of $E(\phi, \sigma_m, \sigma_c)$ is zero. It occurs when the executions of all tasks meet the deadlines and all IMCs meet their timing constraints. In Figure 5.1, we show that the replication of some modules can lead to the minimum E value while the optimal assignment solution fails to find even a feasible solution. We solve the schedulability-oriented replication problem by using a simulated annealing algorithm, identifying the bottleneck IPCs and replicating the sender modules of IPCs to eliminate the IPCs.

5.4 The Algorithm

We exploit the simulated annealing technique to solve the schedulability-oriented replication problem. The structure of the simulated annealing (SA) algorithm is shown in Figure 5.4. The first step of the algorithm is to randomly choose an assignment ϕ, a processor schedule σ_m and a network schedule σ_c. A solution point in the search space of SA is a tuple of (ϕ,σ_m,σ_c). The energy (i.e. the reciprocal of the degree of schedulability) of a solution point is computed by equation (5.1).

For each solution point P which is infeasible, (i.e. E_p is nonzero), we use a dynamic programming technique to optimally identify the bottleneck IPCs in polynomial time. The bottleneck IPCs are the communications which make P infeasible. Once the bottleneck IPCs are identified, we try to replicate the sender modules of bottleneck IPCs. If the replication is successfully done, then it eliminates the bottleneck IPCs and the algorithm terminates in a feasible replication. Procedure *Replicate(P)* in Figure 5.4 consists of two phases, namely (1) identifying bottleneck IPCs and (2) elimination of bottleneck IPCs.

5.4.1 Identifying Bottleneck IPCs

Given an assignment ϕ (i.e. only one copy for each module) and schedules σ_m and σ_c, the first step to perform the replication is to identify which modules should be replicated. Since the purpose of replication is to eliminate the bottleneck IPCs, the problem of identifying which modules to replicate is transformed to the problem of identifying bottleneck IPCs.

A processor schedule, σ_m, is a total ordering of module instances in the comprehensive graph G. A relation $m_1 \stackrel{m}{\rightarrow} m_2$ exists if m_1 is scheduled before m_2 under schedule σ_m. Similarly, define $I(\phi)$ to be the set of IPCs in the system under ϕ. A relation $c_1 \stackrel{c}{\rightarrow} c_2$ exists if c_1, $c_2 \in I(\phi)$ and c_1 is scheduled before c_2 under schedule σ_c on the communication network. The problem of identifying bottleneck IPCs is to find a maximum subset $\vartheta \subset I(\phi)$, such that for any $c \in I(\phi) - \vartheta$ if we reduce μ_c (i.e. the amount of data to be transferred by c) to zero and assign c as a local IMC, then the execution of all task instances in G meet their deadlines. We call ϑ the *optimally feasible set* of IPCs. Formally, given $I(\phi)$, we find ϑ such that

- If $c^k_{i \rightarrow j} \in I(\phi) - \vartheta$, then $t^k_{i \rightarrow j} = 0$. Namely, we change a bottleneck IPC into a local IMC.
- $X(\phi, \vartheta, \sigma_m, \sigma_c)$ is feasible. $X(\phi, \vartheta, \sigma_m, \sigma_c)$ denotes the multi-resource calendar (including processors and communications network) generated for the allocation and scheduling of modules and IPCs under ϕ, ϑ, σ_m, and σ_c. An example can be found in Figure 5.8 in Section 5.5.1. $X(\phi, \vartheta, \sigma_m, \sigma_c)$ is feasible means that (1) the execution of all task instances in G meet their deadlines, (2) the scheduling of IMCs does not violate the consistency checks, and (3) all IMCs meet their timing constraints under ϕ, ϑ, σ_m, and σ_c.
- If $X(\phi, \vartheta', \sigma_m, \sigma_c)$ is also feasible for some other ϑ', then either (1) $|\vartheta'| < |\vartheta|$, or (2) $|\vartheta'| = |\vartheta|$ and $T_f(\ell(\vartheta')) \geq T_f(\ell(\vartheta))$, where $T_f(\ell(I))$[2] is the finish time of $\ell(I)$ (i.e. the last IPC in

[2]Without ambiguity, we drop off σ_c from the notation, i.e. $T_f(x) = T_f(x, \sigma_c)$

```
Choose an initial temperature T
Choose randomly a starting point P = (φ, σ_m, σ_c)
E_p := Energy of solution point P
if E_p > 0 then
        Replicate(P)
        if the replication of P meets tasks' deadlines then
                output the result and exit
        end if
else    output E_p and exit /* E_p = 0 means a feasible solution */
end if
repeat
        repeat
                Choose N, a neighbor of P
                E_n := Energy of solution point N
                if E_n > 0 then
                        Replicate(N)
                        if the replication of N meets tasks' deadlines then
                                output the result and exit
                        end if
                else    output E_n and exit
                        /* E_n = 0 means a feasible solution */
                end if
                if E_n < E_p then
                        P := N
                        E_p := E_n
                else
                        x := (E_p - E_n) / T
                        if e^x ≥ random(0,1) then
                                P := N
                                E_p := E_n
                        end if
                end if
        until thermal equilibrium at T
        T := α × T (where α < 1)
until stopping criterion
```

Figure 5.4: The Structure of SA/R: the Simulated Annealing Algorithm with *Replicate*().

I) on the network. We consider ϑ the maximum subset with the earliest completion time on communications network.

To solve the problem, we use a dynamic programming technique to find ϑ. Given σ_c and $I(\phi)$, we can obtain a total ordering of IPCs. Let us assume $c_1 \xrightarrow{c} c_2 \xrightarrow{c} \ldots \xrightarrow{c} c_K$, where $K = |I(\phi)|$. We need the following notations to facilitate the introduction of the technique.

- $g(m,n)$ is any feasible set of n IPCs out of $\{c_1, c_2, \ldots, c_m\}$. Namely, (1) $X(\phi, g(m,n), \sigma_m, \sigma_c)$ is feasible, (2) $g(m,n) \subset \{c_1, c_2, \ldots, c_m\}$, and (3) $|g(m,n)| = n$. If there is no feasible set of cardinality n, then $g(m,n) = \emptyset$, and $T_f(\ell(g(m,n))) = \infty$.
- $\Omega(m,n)$ is the optimally feasible set out of $\{c_1, c_2, \ldots, c_m\}$, where $|\Omega(m,n)| = n$. If there is no feasible set of cardinality n, then $\Omega(m,n) = \emptyset$, and $T_f(\ell(\Omega(m,n))) = \infty$.

Lemma 5.1 *If $X(\phi, g_1(m,n), \sigma_m, \sigma_c)$ and $X(\phi, g_2(m,n), \sigma_m, \sigma_c)$ are feasible, and $T_f(\ell(g_1(m,n)))$ $\leq T_f(\ell(g_2(m,n)))$, then $T_f(\ell(g_1(m,n) \cup \{c_q\})) \leq T_f(\ell(g_2(m,n) \cup \{c_q\}))$, where $q > m$.*

Proof: Since we consider event-based semantics, each c_q has a ready time r_q and a deadline d_q. The start time for c_q on the network is $T_s(c_q) = \max\ (T_f(\ell(g(m,n))), r_q)$.

$$T_f(\ell(g_1(m,n) \cup \{c_q\})) = T_s(c_q) + \mu_q \times ND = \max(T_f(\ell(g_1(m,n))), r_q) + \mu_q \times ND \tag{5.2}$$

$$T_f(\ell(g_2(m,n) \cup \{c_q\})) = T_s(c_q) + \mu_q \times ND = \max(T_f(\ell(g_2(m,n))), r_q) + \mu_q \times ND \tag{5.3}$$

Since $T_f(\ell(g_1(m,n))) \leq T_f(\ell(g_2(m,n)))$, we can see that $T_f(\ell(g_1(m,n) \cup \{c_q\})) \leq T_f(\ell(g_2(m,n) \cup \{c_q\}))$ by comparing equations (5.2) and (5.3).

□

Lemma 5.2 *If $X(\phi, \Omega(m,n), \sigma_m, \sigma_c)$ and $X(\phi, g(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ are feasible, and $g(m,n) \neq \Omega(m,n)$, where $q > m$, then $X(\phi,\Omega(m,n) \cup \{c_q\},\sigma_m, \sigma_c)$ is feasible.*

Proof: We prove the lemma by contradiction. Assume $X(\phi, \Omega(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ is infeasible. Since $X(\phi, \Omega(m,n), \sigma_m, \sigma_c)$ is feasible, it means that only the inclusion of c_q makes $X(\phi, \Omega(m,n) \cup \{c_q\},\ \sigma_m,\ \sigma_c)$ infeasible. There are three cases in which the inclusion of c_q makes $X(\phi, \Omega(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ infeasible:

1. The start time of c_q fails the consistency check.
2. The finish time of c_q violates its timing constraint. Namely, $T_f(c_q) > d_q$.

3. The data transmission delays the execution of its receiving module m_k and causes some task to miss its deadline. (Note that there is no any IPC after c_q in $X(\phi, \Omega(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$.)

Since (1) the start time of c_q in $X(\phi, \Omega(m,n), \sigma_m, \sigma_c)$ is the same as that in $X(\phi, \Omega(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$, and (2) $X(\phi, \Omega(m,n), \sigma_m, \sigma_c)$ is feasible, they imply that case 1 is not true.

Since $|\Omega(m,n)| = |g(m,n)| = n$, so $T_f(\ell(\Omega(m,n))) \leq T_f(\ell(g(m,n)))$. From Lemma 5.1 , we obtain that $T_f(\ell(\Omega(m,n) \cup \{c_q\})) \leq T_f(\ell(g(m,n) \cup \{c_q\}))$. If case 2 is true, the following inequality holds.

$$T_f(\ell(g(m,n) \cup \{c_q\})) \geq T_f(\ell(\Omega(m,n) \cup \{c_q\})) = T_f(c_q) > d_q. \tag{5.4}$$

It means that $X(\phi, g(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ is infeasible. It contradicts the given conditions. So, case 2 does not hold.

Also, based on equation (5.4), we know that the start time of the receiving module m_k under $X(\phi, g(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ is not earlier than that under $X(\phi, \Omega(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$. So, if case 3 is true, then $X(\phi, g(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ is infeasible; it also contradicts the conditions. So, the assumption is not valid, and $X(\phi, \Omega(m,n) \cup \{c_q\}, \sigma_m, \sigma_c)$ is feasible.

□

Theorem 5.1 *If $\Omega(m-1, n-1)$ and $\Omega(m-1, n)$ are given, the construction of $\Omega(m,n)$ can be done by the following equation.*

$$\Omega(m,n) = \begin{cases} \Omega(m-1,n-1) \cup \{c_m\} & \text{if } T_f(\ell(\Omega(m-1,n-1) \cup \{c_m\})) \leq \\ & T_f(\ell(\Omega(m-1,n))) \\ & \text{and } X(\phi, \Omega(m-1,n-1) \cup \{c_m\}, \\ & \sigma_m, \sigma_c) \text{ is feasible,} \\ \Omega(m-1,n) & \text{if } T_f(\ell(\Omega(m-1,n-1) \cup \{c_m\})) > \\ & T_f(\ell(\Omega(m-1,n))) \\ & \text{and } X(\phi, \Omega(m-1,n), \sigma_m, \sigma_c) \text{ is feasible,} \\ \emptyset & \text{otherwise.} \end{cases} \tag{5.5}$$

Proof: There are two cases for the relation between c_m and $\Omega(m,n)$, i.e, $c_m \in \Omega(m,n)$ or $c_m \notin \Omega(m,n)$.

If $c_m \in \Omega(m,n)$, then we need to find $n-1$ IPCs out of $\{c_1, c_2, \ldots, c_{m-1}\}$. Let $\Omega(m,n) = g(m-1,n-1) \cup \{c_m\}$, where $g(m-1,n-1) \neq \Omega(m-1,n-1)$. Since $\Omega(m-1,n-1)$ is given, and from Lemma 5.2, we know that if $X(\phi, g(m-1,n-1) \cup \{c_m\}, \sigma_m, \sigma_c)$ is feasible, then $X(\phi, \Omega(m-1,n-1) \cup \{c_m\}, \sigma_m, \sigma_c)$ is also feasible. And since $T_f(\ell(\Omega(m-1,n-1))) \leq T_f(\ell(g(m-1,n-1)))$, we obtain

```
for m = 0 to K do Ω(m, 0) = ∅
for n = 0 to K do Ω(0, n) = ∅
for m = 1 to K do
        for n = 1 to K do
                if n > m then
                        Ω(m, n) = ∅
                else
                        Compute Ω(m, n) by equation (5.5)
                end if
        end for
end for
```

Figure 5.5: The Algorithm to find $\Omega(K, i)$'s

that $T_f(\ell(\Omega(m-1, n-1) \cup \{c_m\})) \leq T_f(\ell(g(m-1, n-1) \cup \{c_m\}))$ by Lemma 5.1. So, if we replace $g(m-1, n-1)$ with $\Omega(m-1, n-1)$ and assign $\Omega(m, n) = \Omega(m-1, n-1) \cup \{c_m\}$, we obtain another optimally feasible set.

If $c_m \notin \Omega(m, n)$, then we need to find an optimally feasible set with n IPCs out of $\{c_1, c_2, \ldots, c_{m-1}\}$. Namely, $\Omega(m, n) = \Omega(m-1, n)$.

Taking these two cases into consideration, we can choose the one which is feasible with shorter completion time on the network. And if both $\Omega(m-1, n-1) \cup \{c_m\}$ and $\Omega(m-1, n)$ are infeasible, then $\Omega(m, n) = \emptyset$.

□

Let $K = |I(\phi)|$. The algorithm to find $\Omega(K, 0)$, $\Omega(K, 1)$, …, $\Omega(K, K)$ is shown in Figure 5.5. The complexity of the algorithm is $O(K^2 M)$, where M is the number of modules in the comprehensive graph. If $\Omega(K, 0)$, $\Omega(K, 1)$, …, $\Omega(K, K)$ are given, the problem of identifying bottleneck IPCs and finding ϑ can be solved by assigning $\vartheta = \Omega(K, q)$, where $\forall\ i > q$, $\Omega(K, i) = \emptyset$. Since $\Omega(K, q+1)$, $\Omega(K, q+2)$, …, $\Omega(K, K)$ are infeasible, $\Omega(K, q)$ is the maximum subset of $I(\phi)$ which is feasible. Hence, the set of bottleneck IPCs is $I(\phi) - \Omega(K, q)$.

5.4.2 Elimination of Bottleneck IPCs

Once the bottleneck IPCs are identified, we use a replication technique to eliminate them. For each $c^k_{i \to j} \in I(\phi) - \vartheta$, we identify the sending module instance m^k_i on processor p and the receiving module instance m_j^k on processor q. The approach to eliminate $c_{i\to j}^k$ is to have a copy of module m_i^k on processor q, i.e. $\phi(m_i^k, q) = 1$. If the outdegree of the sending module instance m_i^k on processor p in the comprehensive graph G is 1 (i.e. m_i^k only communicates with m_j^k), we remove the module instance by setting $\phi(m_i^k, p) = 0$. It means that we can eliminate bottleneck IPCs by either replication or reassignment of the sender module.

Since the allocation ϕ is changed due to the replication or reassignment, the set of IPCs, $I(\phi)$, is also changed. Formally, $\forall\ c_{i\to j}^k \in G$, $\exists\ p$ and q, such that if $\phi(m_j^k, q) = 1$, then there exists a m_i^k where $\phi(m_i^k, p) = 1$ and m_i^k on processor p sends $\mu_{i\to j}^k$ units of data to m_j^k on processor q. If $p \neq q$ then the IPC to perform $c_{i\to j}^k$ should be included in $I(\phi)$.

If $X(\phi, I(\phi), \sigma_m, \sigma_c)$ is feasible, then we have found a feasible allocation ϕ. The algorithm terminates. Otherwise, we use the following strategy to find a new solution point.

5.4.3 Neighbor Finding Strategy

The neighbor space of a solution point is the set of points which can be reached by changing the assignment of one or two modules, swapping the execution order of two modules in the processor schedule, or swapping the order of two IPCs in the network schedule. In short, there are three modes of neighbor finding strategy.

- Swap Mode: This mode is applicable to ϕ, σ_m, or σ_c. For ϕ, we randomly choose two modules m_i^k and m_j^l on processors p and q respectively. These two modules are not necessarily in the same task. Then we change ϕ by setting $\phi(m_i^k, p) = 0$, $\phi(m_j^l, q) = 0$, $\phi(m_i^k, q) = 1$, and $\phi(m_j^l, p) = 1$. Note that the change of ϕ effects $I(\phi)$. For σ_m or σ_c, we choose two modules or two IPCs and swap their order in the schedules. Since the comprehensive graph itself defines a partial order of the schedules, the swapping of modules and IPCs should not violate the partial order.

- Random Mode: This mode is applicable to ϕ only. We move a module to any other processor and obtain a new ϕ. The move is an attempt to try to find a new solution point with a lower energy level.

- Merge Mode: This mode is applicable to ϕ only. We pick two module instances and move them to one processor. By merging two modules to one processor, we increase the workload of the processor. There is an opportunity of increasing the energy level of the new point by increasing the workload of the processor. The purpose of the move is to perturb the system and allow the next move to escape from the local optimum.

The scheme to apply the appropriate modes to the current solution point is based on the feedback from the energy function. Instead of randomly choosing a mode, we investigate the current energy value $E(\phi, \sigma_m, \sigma_c)$ and find out which mode is to be applied. For example, if the jth instance of task τ_i misses its deadline (i.e. $f_i^j > d_i^j$), we can consider any mode which is applicable to ϕ, σ_m or σ_c. On the other hand, if some IMC violates the timing constraints, then we can apply swap mode to σ_c. These three modes are adequate to find a feasible solution from a starting point, if feasible solutions exist.

5.5 Experimental Results

We implemented the algorithm (SA/R) as the framework of the allocator on MARUTI [GMK+91, MSA92], a real-time operating system developed at the University of Maryland, and conducted extensive experiments with tasks having different characteristics. The tests involve the allocation of real-time tasks on a homogeneous distributed system with seven sites connected by a communication channel. The nominal delay ND of the communication channel falls into the range between 1 and 6 time units. The computation time of each module is uniformly distributed between 1 and 25 time units. The amount of data transfer, μ, for each IMC is between 1 and 10 data units.

To show how the algorithm finds a feasible solution, we present an example allocation problem and its solution in Section 5.5.1. The comprehensive graph for the problem and the allocation result are shown on an interactive graphical display which is developed for the design of MARUTI. There are 50 modules in the comprehensive graph. The allocation result shows that 7 modules are replicated to increase the schedulability of the whole task system and meet the timing constraints. For this problem, if each module is restricted to only one site, then there is no feasible solution to the problem. The significance of the schedulability-oriented replication is demonstrated by this example.

We also compare the performance of the algorithm (SA/R) with that of the simulated annealing (SA) algorithm (i.e. SA/R without *Replicate*()). For the cases where feasible solutions exist for single-copy task assignment, the numbers of iterations needed for SA/R to find feasible solutions are much less than those for SA. For each solution point P, *Replicate*(P) is able to identify the bottleneck IPCs, use the replication technique to eliminate the IPCs if possible, and meet the tasks' deadlines.

To test the practicality of SA/R, we ran experiments on the systems with 10-55 modules, while varying task deadlines, module execution times, IMC volumes, and randomly generating precedence relationships. For each randomly generated problem, the number of trials is 1000. We compare

Modules (M)		10	20	34	55
Best ratio case	Iter. SA/R (I1)	1	1	23	83
	Iter. SA (I2)	99	10442	65883	55,133
	Ratio (I1/I2)	1.010%	0.001%	0.034%	0.151%
Worst ratio case	Iter. SA/R (I3)	65	522	384	2,211
	Iter. SA (I4)	66	899	387	2,212
	Ratio (I3/I4)	98.48%	58.06%	99.22%	99.95%
Avg. ratio case	Iter. SA/R (I5)	15	152	195	1,924
	Iter. SA (I6)	148	2060	4532	9,844
	Ratio (I5/I6)	10.268%	7.388%	4.305%	19.54%
Search Space ($> M^7$)		1×10^7	1.28×10^9	5.25×10^{10}	1.52×10^{12}

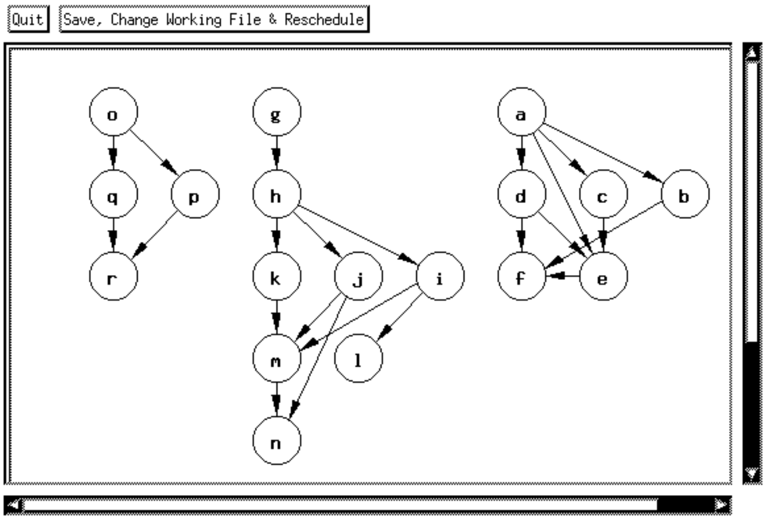
Table 5.1: Experiments Results for SA and SA/R

the number of iterations needed to find a feasible solution for SA and SA/R respectively. The summarized results of best, average, and worst cases for each problem are given in Table 5.1. The size of potential search space for each problem depends on the number of modules in the graph (M), number of processors (p), number of IMCs, and precedence relationships. It is not easy to get an accurate value for that. In Table 5.1, we show the search space for allocation (i.e. ϕ) only. The actual search space is bigger than M^p. From the results in Table 5.1, we see that SA and SA/R perform very well and solve the problems with reasonable time complexity. For example, for the cases in which $M = 55$, the average number of iterations needed to solve the problem by SA is less than 9,900, while the search space is greater than 1.52×10^{12}. Only 6×10^{-7} % of the search space is explored to find a feasible solution. Furthermore, the number of iterations by SA/R is less than that by SA, and SA/R outperforms SA in each case.

5.5.1 An Example Problem

Let us consider an example of a task allocation problem. The real-time distributed system in this example consists of 7 sites with one processor at each site. The sites are connected by a real-time communication channel, for which the nominal delay in this example is 2. The task characteristics and task graph for each task are shown in Figure 5.6. The entry "(4)→ b" indicates 4 units of data sent from modules a to b. Ready time for each task is assumed to be the beginning of each period.

The LCM of three periods in this example is 240. The numbers of instances in the comprehensive graph for tasks 1, 2, and 3 are 3, 2, and 4 respectively. The comprehensive graph is shown in Figure 5.7. There are 50 modules in the graph. The allocation problem is to allocate these 50



Task	p_i	d_i	Module	Exec_time	IMC
1	80	70	a	17	(4)→ b, (7)→ c, (3)→ d, (1)→ e
			b	14	(2)→ f
			c	11	(5)→ e
			d	16	(2)→ e, (5)→ f
			e	9	(2)→ f
			f	10	
2	120	110	g	15	(3)→ h
			h	8	(4)→ i, (9)→ j, (5)→ k
			i	9	(7)→ l, (9)→ m
			j	10	(3)→ m, (8)→ n
			k	11	(5)→ m
			l	8	
			m	9	(2)→ n
			n	10	
3	60	60	o	18	(3)→ p, (3)→ q
			p	18	(3)→ r
			q	24	(3)→ r
			r	18	

Figure 5.6: Graphical Description of the Example Problem

modules to 7 processors such that the energy value in Equation (1) is zero.

The results from SA/R are shown in Figure 5.8, where modules 0, 6, 12, 34, ,38, 42 and 46 are replicated. These replications eliminate the bottleneck IPCs. Figure 5.8 is obtained from the graphical display of MARUTI. For each IPC there are upper and lower IDs. The upper ID indicates the sender module while the lower ID indicates the recipient module. This example illustrates the use of the techniques presented in the chapter.

5.5.2 Discussion

The complexity involved in *Replicate*() is $O(K^2N)$, where K is the number of IPCs and N is the number of modules in the comprehensive graph. The worst case running time for SA/R is of the order $O(K^2N)$ times that for SA if *Replicate*(P) is invoked in each iteration. To avoid the unnecessary invocation of *Replicate*(P) for an infeasible solution point P, we can speed up SA/R by conditionally applying *Replicate*(P) to the solution point P only when E_p is small. For the high-energy point P, the number of tasks violating their deadlines is large. It is difficult for *Replicate*(P) to find a feasible replication for high-energy point P.

From the experiments, we find that *Replicate*(P) plays a significant part in SA/R. In the original SA algorithm (i.e. the one without *Replicate*(P)), when the system is at a low temperature, the energy level is low and the probability of accepting an uphill move is small. In order to escape from the local minimum and find a global minimum, the number of iterations needed at a low temperature is extremely large. The number of iterations has been modeled as a Markov Chain [LH91], where an upper bound on the number of iterations at low temperatures is enforced to avoid infinite chains. In SA/R, *Replicate*(P) is invoked to search for a feasible replication for P if E_p is small. From our experiments we find that for a local minimum point P, SA/R is able to terminate the searching process as soon as a feasible replication is found. For this reason, SA/R outperforms SA.

The inclusion of resource and memory constraints into the replication problem can be solved by modifying the neighbor-finding strategy. Once a neighbor of the current point is generated, it is checked to ascertain that the constraints on memory etc. are met. If not, the neighbor is discarded and another neighbor is evaluated.

5.6 Summary

In this chapter we have focused on a schedulability-oriented replication problem of periodic complex tasks in real-time distributed systems. The purpose of replication is to reduce inter-processor

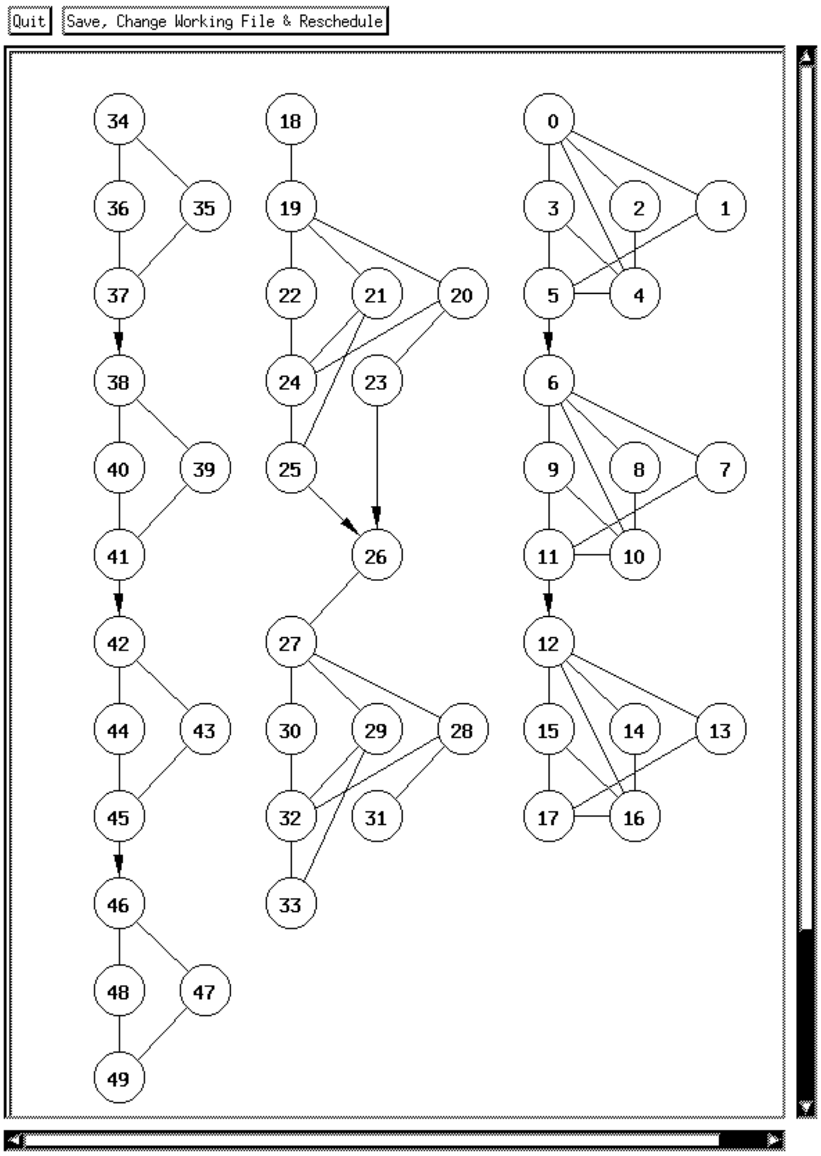


Figure 5.7: The Comprehensive Graph of the Example Problem

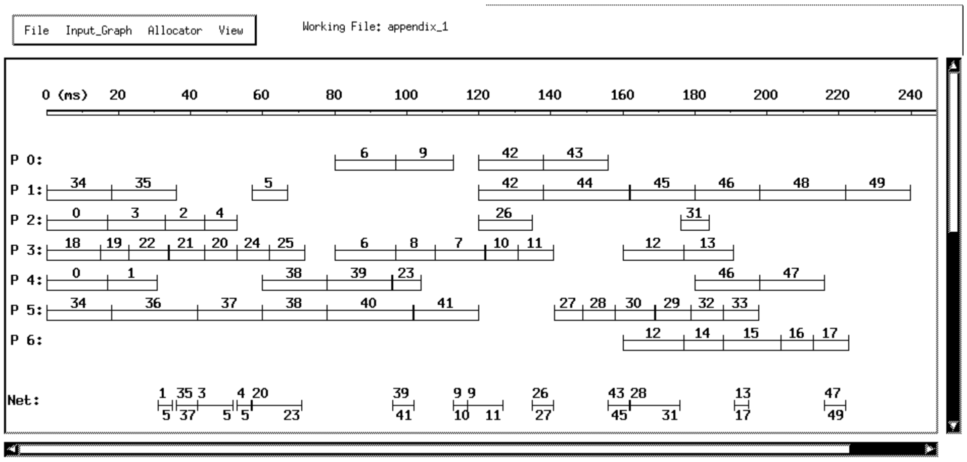


Figure 5.8: Allocation Result of the Example Problem

communication, and to increase the degree of schedulability of a task set.

We have developed a replication technique and embedded the technique in a simulated annealing algorithm to solve the problem. We have presented a two phase replication technique. In the first phase, the bottleneck IPCs, which delay the execution of the recipient modules and cause the tasks to miss their deadlines, are identified by using a dynamic programming technique. The second phase is used to replicate the sending modules to eliminate the bottleneck IPCs. The complexity of the technique is $O(K^2M)$ where K is the number of IPCs and M is the number of modules in the system. We have implemented the algorithm as the framework for the allocator of MARUTI. The experimental results show that the algorithm performs well and finds feasible solutions.

Even though the algorithm is primarily concerned with distributed systems with one communication channel in the network, it can be extended to multiple channels.

Chapter 6

Performance-Oriented Replication in Non-Real-Time Systems

In the previous chapters, we focused on the allocation problems in the real-time systems in which the allocation problems involve the scheduling of tasks and communications. From the results in Chapter 5, we see that the schedulability-oriented replication can lead to higher feasibility in real-time systems. In this chapter, we study how the replication of tasks can reduce the execution cost of a task in the non-real-time systems.

6.1 Introduction

For non-real-time applications, the objective of the allocation problem may be the minimum completion time, processor load balancing, or total cost of execution and communication, etc. For the assignment problem in which the objective is to minimize the total cost of execution and interprocessor communication, Stone [Sto77] and Towsley [Tow86] presented $O(n^3M)$ algorithms for tree-structure and series-parallel graphs, respectively, of M tasks and n processors. For general task graphs, the assignment problem has been proven [MM89] to be NP-complete. Many papers [MLT82, MM89, PP82] presented branch-and-bound methods which yielded an optimal result. Other heuristic methods have been considered by Lo [Lo88] and Price and Krishnaprasad [PK84]. All these works focused on the assignment problem.

Traditionally, the main purpose of replicating a task on multiple processors is to increase the degree of fault tolerance [CC90, LAMS92]. If some processors in a system fail, the application may still survive using other copies. In such a model, a task has to communicate with multiple copies of other tasks. As a consequence, the total cost of execution and communication of the replication problem is bigger than that of the assignment problem. In this chapter, we adopt another communication model in which the replication of a task is not for the sake of fault tolerance but for decreasing of the total cost. In our model, each task may have more than one copy and it may start its execution after receiving necessary data from one copy of each preceding task. Clearly, in a heterogeneous environment the cost of execution of a task depends on the processor on which it executes, and the communication costs depend on the topology, communication medium, protocols used, etc. When a task t is allowed to have only one copy in the system, the sum of the interprocessor communication costs between t and other tasks may be large. Sometimes it will be more beneficial if we replicate t onto multiple processors to reduce the inter-processor communication, and to utilize the available processors in the systems. Such replication may lead to a lower total cost than the optimal assignment problem. An example illustrating this point is presented in Section 6.3.

In the assignment problem, polynomial-time algorithms exist for special cases, such as tree-structure [Sto77] and series-parallel [Tow86] task graphs. This work represents one of the first few attempts at finding special cases for the replication problem. The class of applications we consider in this chapter is computation-intensive applications in which the execution cost of a task is greater than its communication cost. Such applications can be found in many fields, such as digital signal processing, weather forecasting, game searching, etc. We formally define a computation-intensive application in Section 6.2.2. We prove that for the computation-intensive applications, the replication problem is NP-complete, and we present a branch-and-bound algorithm to solve it. The worst-case complexity of the solution is $O(n^2 2^n M)$. Note that the algorithm is able to solve the problem with the complexity of the linear function of M.

We also develop an approximation approach to solve the problem in polynomial time. Given a forker task s with K successors in the SP graph, the method tries to allocate s to processors based on iterative selection. The complexity of the iterative selection for a forker is $O(n^2 K^2)$, while the overall solution for an SP graph is $O(n^4 M^2)$.

In the remainder of this chapter, the series-parallel graph model and the computation model are described in Section 6.2. In Section 6.3, the replication problem is formulated as the minimum cost 0-1 integer programming problem and the proof of NP completeness is given. A branch-and-bound algorithm and numerical results are given in Section 6.4, while the approximation methods and results are given in Section 6.5. The overall algorithm is presented and conclusion remark is drawn in Section 6.6.

6.2 Definitions

6.2.1 Graph Model

A *series-parallel* (SP) graph, $G = (V, E)$, is a directed graph of type p, where $p \in \{T_{unit}, T_{chain}, T_{and}, T_{or}\}$ and G has a source node (of indegree 0) and a sink node (of outdegree 0). An SP graph can be constructed by applying the following rules recursively.

1. A graph $G = (V, E) = (\{v\}, \phi)$ is an SP graph of type T_{unit}. (Node v is the source and the sink of G.)

2. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are SP graphs then $G' = (V', E')$ is an SP graph of type T_{chain}, where $V' = V_1 \cup V_2$ and $E' = E_1 \cup E_2 \cup \{<$sink of G_1, source of $G_2 >\}$.

3. If each graph $G_i = (V_i,E_i)$ with source-sink pair (s_i,t_i), where s_i is of outdegree 1, is an SP graph, $\forall\ i = 1,2,\ldots,n$, and new nodes $s' \notin V_i$ and $t' \notin V_i$, $\forall\ i$ are given then $G' = (V', E')$ is an SP graph of type T_{and}(or type T_{or}), where $V' = V_1 \cup V_2 \cup \ldots \cup V_n \cup \{s', t'\}$ and $E' = E_1 \cup E_2 \cup \ldots \cup E_n \cup \{< s', s_i > \mid \forall\ i = 1,2,\ldots,n\ \} \cup \{< t_i, t' > \mid \forall\ i = 1,2,\ldots,n\ \}$. The source of G', s', is called the *forker* of G'. The sink of G', t', is called the *joiner* of G'. G' is an SP graph of type T_{and}(or type T_{or}) if there exists a *parallel-and* (or *parallel-or*) relation among G_i's.

A convenient way of representing the structure of an SP graph is via a parsing tree [JLT]. The transformation of an SP graph to a parsing tree can be done in a recursive way. There are four kinds of internal nodes in a parsing tree: T_{unit}, T_{chain}, T_{and} and T_{or} nodes. A T_{unit} node has only one child, while a T_{chain} node has more than one child. Every internal node x, along with all its descendant nodes induces a subtree S_x which describes an SP subgraph G_x of G. Each leaf node in S_x corresponds to an SP graph of type T_{unit}. A T_{and}(or T_{or}) node y consists of its type T_{and}(or T_{or}) along with the forker and joiner nodes of G_y. We give an example of an SP graph G, and its parsing tree $T(G)$ in Figure 6.1.

6.2.2 Computational Model

An application program consists of M tasks labeled $m = 1, 2, \ldots, M$. Its behavior is represented by an SP graph where the tasks correspond to the nodes. Each task may be replicated onto more than one processor. A *task instance* $t_{i,p}$ is a replication of task i on processor p. A directed edge $< i, j >$ between nodes i and j exists if the execution of task j follows that of task i. Associated with each edge $< i, j >$ is the communication cost incurred by the application. We are concerned with types of applications where the cost of execution of a task is always greater than the communication overhead it needs. The model is stated as follows.

Given a system S with n processors connected by a communication network, an application is *computation-intensive* if its associated SP graph $G = (V, E)$ on S satisfies the following conditions:

1. $\mu_{i,j}(p, q) \geq 0$,
2. $\sum_{q=1}^{n} \mu_{i,j}(p, q) \leq \min_p(e_{i,p})$, $\forall < i, j > \in E$, and $1 \leq p \leq n$, where

- $\mu_{i,j}(p, q)$ is the communication cost between tasks i and j when they are assigned to processors p and q respectively, and
- $e_{i,p}$ is the execution cost when task i is assigned to processor p.

The first condition states that the communication cost between any two task instances (e.g. $t_{i,p}$ and $t_{j,q}$) is not negative. The second one depicts that for every edge $< i, j >$, the worst-case communication cost between any task instance $t_{i,p}$ and all its successor task instances (i.e. $t_{j,q}$'s, $\forall$ q) is less than the minimum execution cost of task i.

6.2.3 Communication Model

The communication model we considered is different from that of reliability-oriented replication. In the reliability-oriented replication problem, the objective is to increase the degree of fault tolerance. To detect fault and maintain data consistency, each task has to receive multiple copies of data from several task instances if its predecessor is replicated in more than one place.

The purpose of the replication problem considered in this chapter is to decrease the sum of execution and communication costs. Under such consideration, there is no need to enforce plural communication between any two task instances. We propose the **1-out-of-n** communication model. In the model, for each edge $< i, j > \in E$, a task instance $t_{j,q}$ may start its execution if it receives the data from *any one* task instance of its predecessor, task i.

6.3 Problem Formulation and Complexity

Based on the computational model presented in Section 6.2.2, the problem of minimizing the total sum of execution and communication costs for an SP task graph can be approached by replication

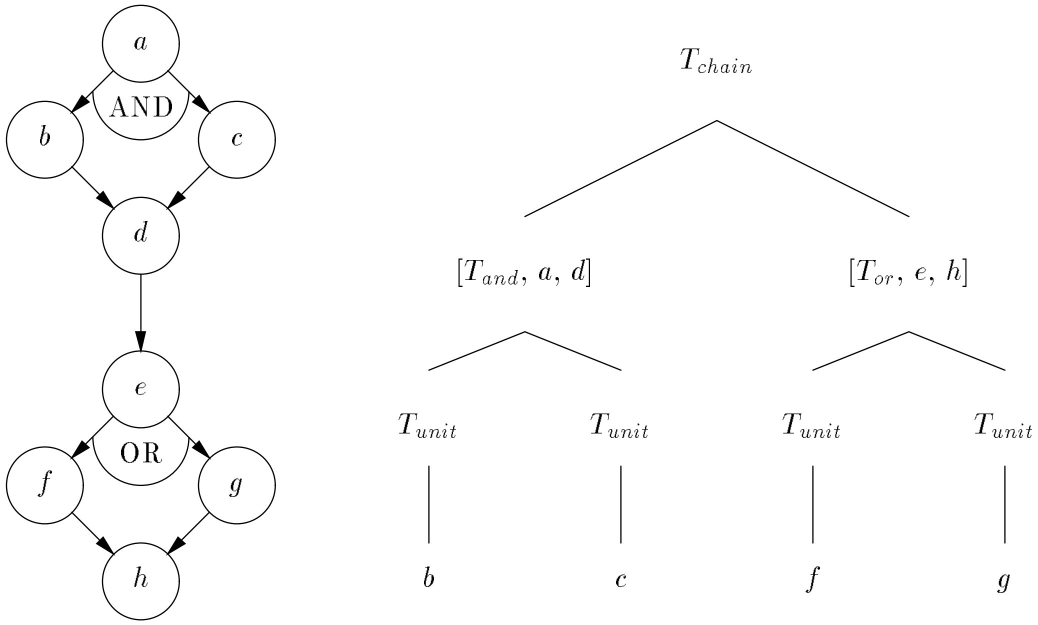
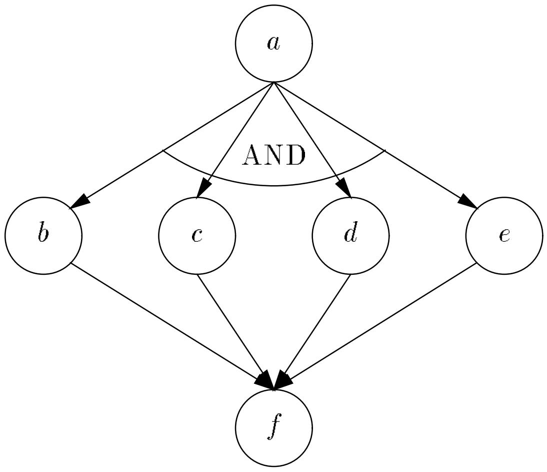


Figure 6.1: An SP graph and its parsing tree

of tasks. An example in which the replication may lead to a lower sum of execution costs and communication costs is given in Figure 6.2, where the number of processors in the system is two, and the execution costs and communication costs are listed in e table and μ table respectively. If each task is allowed to run on at most one processor, then the optimal allocation will be to assign task a to processor 1, b to 1, c to 1, d to 2, e to 2, and f to 1. The minimum cost is 68. However, if each task is allowed to be replicated, (i.e. to replicate task a to processors 1 and 2), then the cost is 67.

We introduce integer variable $X_{i,p}$'s, $\forall\ 1 \leq i \leq M$ and $1 \leq p \leq n$, to formulate the problem where each $X_{i,p} = 1$ if task i is replicated on processor p; and $= 0$, otherwise. We define a binary function $\delta(x)$. If $x > 0$ then $\delta(x) = 1$ else $\delta(x) = 0$. We also associate an **allocated flag** $F(w)$ with each node w in the parsing tree, where $F(w) = 1$ if the allocation for tasks under the subtree S_w is valid; and $= 0$, otherwise. A valid allocation is one that makes the execution of tasks under subtree S_w succeed. A valid allocation is not necessarily the allocation in which each task in S_w is allocated to at least one processor. Some tasks in T_{or} subgraphs may be neglected without effecting the successful execution of an SP graph.

Given an SP graph G, its parsing tree $T(G)$ and any internal node w in $T(G)$, allocated flag $F(w)$ can be recursively computed:



e table	processor 1	processor 2
task a on	5	5
task b on	7	16
task c on	10	20
task d on	25	8
task e on	14	6
task f on	10	13

μ table	$t_{b,1}$	$t_{b,2}$	$t_{c,1}$	$t_{c,2}$	$t_{d,1}$	$t_{d,2}$	$t_{e,1}$	$t_{e,2}$
$t_{a,1}$ to	1	4	1	4	1	4	1	4
$t_{a,2}$ to	4	1	4	1	4	1	4	1
to $t_{f,1}$ from	3	3	3	3	3	3	3	3
to $t_{f,2}$ from	3	3	3	3	3	3	3	3

Optimal Assignment:
$e_{a,1} + \mu_{a,b}(1,1) + \mu_{a,c}(1,1) + \mu_{a,d}(1,2) + \mu_{a,e}(1,2) + e_{b,1} + e_{c,1}$
$+ e_{d,2} + e_{e,2} + \mu_{b,f}(1,1) + \mu_{c,f}(1,1) + \mu_{d,f}(2,1) + \mu_{e,f}(2,1) + e_{f,1} = 68$

Optimal Replication:
$e_{a,1} + e_{a,2} + \mu_{a,b}(1,1) + \mu_{a,c}(1,1) + \mu_{a,d}(2,2) + \mu_{a,e}(2,2) + e_{b,1}$
$+ e_{c,1} + e_{d,2} + e_{e,2} + \mu_{b,f}(1,1) + \mu_{c,f}(1,1) + \mu_{d,f}(2,1) +$
$\mu_{e,f}(2,1) + e_{f,1} = 67$

Figure 6.2: An example to show how the replication can reduce the total cost

1. if w is a T_{unit} node with a child i, then

$$F(w) = F(i) = \delta(\sum_{p=1}^{n} X_{i,p})$$

2. if w is a T_{chain} node with c children, $F(w) = F(child_1) \times F(child_2) \times \ldots \times F(child_c)$.

3. if w is a T_{and} node with forker s, joiner t and c children, then $F(w) = F(s) \times F(t) \times F(child_1) \times F(child_2) \times \ldots \times F(child_c)$.

4. if w is a T_{or} node with forker s, joiner t and c children, then $F(w) = F(s) \times F(t) \times \delta(F(child_1) + F(child_2) + \ldots + F(child_c))$.

The minimum cost replication problem for SP graphs, MCRP-SP, can be formulated as 0-1 integer programming problem, i.e:

$$Z = \text{Minimize } [\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,\ 1\le q\le n} \min_{X_{i,p}=1} (\mu_{i,j}(p,q) * X_{j,q})\ ]$$
$$\text{subject to } F(r) = 1, \text{ where } r \text{ is the root of } T(G) \text{ and } X_{i,p} = 0 \text{ or } 1, \forall i, p. \qquad (6.1)$$

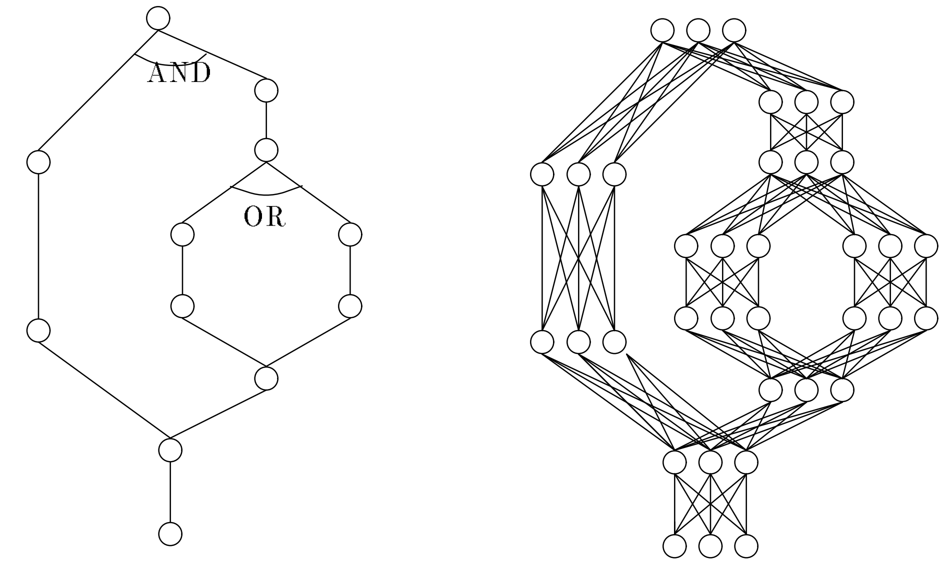
The restricted problem which allows each task to run on at most one processor has the following formulation.

$$Z = \text{Minimize } [\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,p,q} \mu_{i,j} * X_{i,p} * X_{j,q}\ ]$$
$$\text{subject to } \sum_{p=1}^{n} X_{i,p} \le 1 \text{ and } F(r) = 1,$$
$$\text{where } r \text{ is the root of } T(G) \text{ and } X_{i,p} = 0 \text{ or } 1, \forall i, p. \qquad (6.2)$$

The task assignment problem(6.2) for SP graphs of M tasks onto n processors, has been solved in $O(n^3M)$ time [Tow86]. However,the multiprocessor task assignment for general types of task graphs without replication has been reported to be NP-complete [MM89]. As for the MCRP-SP problem, it can be shown to be NP-complete. In this chapter, we present a linear-time algorithm for computation-intensive applications with SP graphs, when the number of processors n is given.

6.3.1 Assignment Graph

Bokhari [Bok87] introduced the assignment graph to solve the task assignment problem (6.2). To prove the NP completeness of problem (6.1) and solve the problem, we also adopt the concept of the assignment graph of an SP graph. The assignment graph of an SP graph can be defined similarly. We draw up an assignment graph for an SP graph in Figure 6.3(a).



(a): An SP graph and its assignment graph.

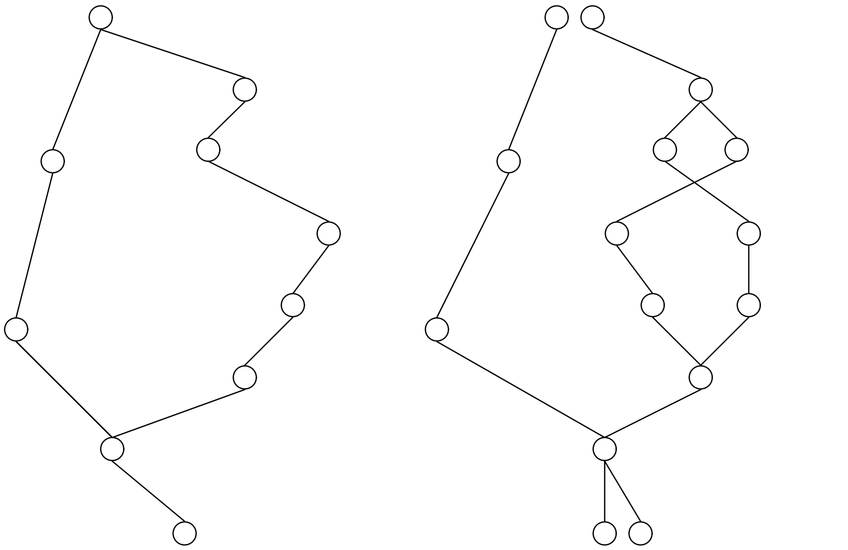


Figure 6.3: (b): An allocation graph and a replication graph of (a)

1. It is a directed graph with weighted nodes and edges.
2. It has $M \times n$ nodes. Each weighted node is labeled with a task instance, $t_{i,p}$.
3. A *layer* i is the collection of n weighted nodes ($t_{i,1}$, $t_{i,2}$, ..., and $t_{i,n}$). Each layer of the graph corresponds to a node in the SP graph. The layer corresponding to the source (sink) is called source (sink) layer.
4. A part of the assignment graph corresponds to an SP subgraph of type T_{chain}, T_{and} or T_{or} is called a T_{chain}, T_{and} or T_{or} *limb* respectively.
5. Communication costs are accounted for by giving the weight $\mu_{i,j}(p,q)$ to the edge going from $t_{i,p}$ to $t_{j,q}$.
6. Execution costs are assigned to the corresponding weighted nodes.

Given an assignment graph, Bokhari [Bok87] solves Problem (6.2) by selecting one weighted node from each layer and including the weighted edges between any two selected nodes. This resulting subgraph is called an *allocation graph.* To solve Problem (6.1), more than one weighted node from each layer may be chosen. Similarly, a *replication graph* for Problem (6.1) can be constructed from an assignment graph by including all selected nodes and edges between these nodes. Examples of an allocation graph and a replication graph are shown in Figure 6.3(b) for an assignment graph shown in Figure 6.3(a). Note that for each node x and each predecessor layer y of x in the replication graph, there is only one edge from y to x.

In a replication graph, each layer may have more than one selected node. Let Variable $\bar{X}_l$ = ($X_{l,1}$, $X_{l,2}$, ..., $X_{l,n}$) be a replication vector for layer l in a replication graph. We define the minimum *activation cost* of vector $\bar{X}_i$ for layer i , $A_i(\bar{X}_i)$, to be the minimum sum of the weights of all possible nodes and edges leading to the selected nodes of layer i in a replication graph. Then the goal of Problem (6.1) can be achieved by computing the minimal value of $\{A_{\text{sink}}(\bar{X}_{\text{sink}}) + \sum_{p=1}^{n} X_{\text{sink},p} * e_{\text{sink},p}\}$ over all possible values of $\bar{X}_{\text{sink}}$.

6.3.2 Complexity

In this section, we can show that Problem (6.1) for a computation-intensive application is NP-complete provided we prove the following:

Lemma 6.1 *For any layer l in the replication graph, the minimum activation cost for two selected nodes $t_{l,p}$ and $t_{l,q}$ will be always greater than that for either node $t_{i,p}$ or $t_{l,q}$ only.*

Proof: The Lemma can be proven by contradiction. Let A_1 be the the minimum activation cost for two nodes $t_{l,p}$ and $t_{l,q}$, and A_2 and A_3 be the minimum costs for $t_{l,p}$ and $t_{l,q}$ respectively. Assume that $A_1 < A_2$ and $A_1 < A_3$. Since A_1 includes the activation cost of node $t_{l,p}$, an activation cost for $t_{l,p}$ only can be obtained from A_1. The obtained value c is not necessarily the minimum value for $t_{l,p}$, hence $A_2 \leq c$. The value c is obtained by removing some weighted nodes and edges from replication graph. This implies that $c < A_1$. From above, we find that $A_2 < A_1$, which contradicts the assumption. The same reasoning can be applied to A_3 and reaches a contradiction. Therefore, the assumptions are incorrect and Lemma 6.1 holds.

□

Lemma 6.1 can be further extended to the cases where more than two weighted nodes are chosen. The conclusion we can draw is that the more nodes are selected from a layer, the bigger the activation cost is.

Lemma 6.2 *Given a computation-intensive application with its SP task graph $G = (V, E)$ and its assignment graph, if node i has outdegree one and edge $< i, j > \in E$, then for any vector $\bar{X}_j$, the minimal activation cost $A_j(\bar{X}_j)$ can be obtained by choosing only one weighted node from layer i. (i.e. $\sum_{p=1}^{n} X_{i,p} = 1$)*

Proof: The Lemma can be proven by contradiction. Since node i has outdegree one and edge $< i, j > \in E$, we know that

$$A_j(\bar{X}_j) = \min_{\bar{X}_i}\{A_i(\bar{X}_i) + \sum_{p=1}^{n} X_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))\}.$$

Let us assume that the above equation reaches a minimal value m when more than one node from layer i is selected and the optimal replication vector is $\bar{X}_i^0$. Since $\sum_{p=1}^{n} X_{i,p} > 1$ for $\bar{X}_i^0$, we may remove one selected node from layer i and obtain a new vector $\bar{X}_i'$. Without loss of generality, let us remove $t_{i,r}$. By removing node $t_{i,r}$, a new value m' is obtained. Since m is the minimum value for layer i, it implies that $m \leq m'$.

From Lemma 6.1, we obtain that $A_i(\bar{X}_i') < A_i(\bar{X}_i^0)$. And for a computation-intensive application, the following holds that $\sum_{q=1}^{n} \mu_{i,j}(p,q) \leq \min_p(e_{i,p})$, $\forall\ 1 \leq p \leq n$. Then,

$$m' = A_i(\bar{X}_i') + \sum_{p=1}^{n} X'_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))$$

$$
\begin{aligned}
&< \; A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X'_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X'_{i,p}=1} (X_{j,q} * \mu_{i,j}(p,q)) \\
&< \; A_i(\bar{X}_i^0) + (\sum_{p=1}^{n} X^0_{i,p} * e_{i,p} \; - \; e_{i,r}) + \sum_{q=1}^{n} \min_{X'_{i,p}=1} (X_{j,q} * \mu_{i,j}(p,q)) \\
&= \; A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1} (X_{j,q} * \mu_{i,j}(p,q))] - e_{i,r} \\
&< \; A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1} (X_{j,q} * \mu_{i,j}(p,q))] - \min_{p}(e_{i,p}) \\
&\leq \; A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1} (X_{j,q} * \mu_{i,j}(p,q))] - \sum_{q=1}^{n} \mu_{i,j}(p,q) \\
&< \; A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} \\
&< \; A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X^0_{i,p}=1} (X_{j,q} * \mu_{i,j}(p,q)) = m.
\end{aligned}
$$

The result, $m' < m$, contradicts our assumption. It means that the assumption is wrong and Lemma 6.2 holds.

□

Lemma 6.3 *Given a computation-intensive application with its SP task graph G, the objective of the minimum cost can be achieved by considering only the replication of the forkers.*

Proof: We proceed to prove the lemma by contradiction. Let the minimum cost for task replication problem be z_0 if only the forkers(i.e. outdegree > 1) are allowed to run on more than one processor. Assume the total cost can be reduced further by replicating some task i which is not a forker. Then there are two possible cases for i:

1. i has outdegree 0.

2. i has outdegree 1.

In case 1, i is the sink of the whole graph. Also i may be the joiner of some SP subgraphs. If i is allowed to run on an extra processor b, which is different from the one which i is initially assigned to (when z_0 is obtained), then the new cost will be $z_0 + e_{i,b} + \sum_{<d,i>\in E} \mu_{d,i}$. Apparently, the new cost is greater than z_0. This contradicts our assumption that the total cost can be reduced further by replicating task i.

In case 2, i has one successor. Let $< i, j > \in E$. From the assumption, we know that the replication of i can reduce the total cost. Hence, the minimum activation cost for task instances in layer j, $A_j(\bar{X}_j)$, is obtained when task i is replicated onto more than one processor. This contradicts Lemma 6.2. Hence, the assumption is incorrect and the objective of the minimum cost can be achieved by considering only the replication of the forkers.

$\square$

Lemma 6.3 tells that, given an SP graph, if we can find out the optimal replication for the forkers, Problem (6.1) for computation-intensive applications can be solved. Now, we show that the problem of finding an optimal replication for the forkers in an SP graph is NP-complete. First, a special form of the replication problem is introduced.

Uni-Cost Task Replication (UCTR) problem is stated as follows:

INSTANCE: Graph $G' = (V', E')$, $V' = V_1' \cup V_2'$, where $\mid V_1' \mid = n$ and $\mid V_2' \mid = m$. If $x \in V_1'$ and $y \in V_2'$ then edge $< x, y > \in E'$ (i.e. $\mid E' \mid = m \times n$). For each $x \in V_1'$, there is an activation cost m. Associated with each edge $< x, y > \in E'$, there is a communication cost $d_{x,y} = n \times m$ or 0. A positive integer $K \leq n \times m$ is also given.

QUESTION: Is there a feasible subset $V_k \subseteq V_1'$ such that, we have

$$[\sum_{x \in V_k} m + \sum_{y \in V_2'} \min_{x \in V_k} (d_{x,y})] \leq K? \tag{6.3}$$

Theorem 6.1 *Uni-Cost Task Replication problem is NP-Complete.*

[**Proof**]: The problem is NP because a subset V_k, if it exists, can be checked to see if the sum of activation costs and communication costs is less than or equal to K. We will now transform the VERTEX COVER [GJ79] problem to this problem. Given any graph $G = (V,E)$ and an integer $C \leq \mid V \mid$, we shall construct a new graph $G' = (V',E')$ and $V' = V_1' \cup V_2'$, such that there exists a VERTEX COVER of size C or less in G if and only if there is a feasible subset of V_1' in G'. Let $\mid V \mid = n$ and $\mid E \mid = m$. To construct G', we create a vertex v_i for each node in V and number the edges in E and create a vertex b_i for each edge $< u,v > \in E$ where u, $v \in V$. We define $K = m \times C$, $V_1' = \{v_1, v_2, \ldots, v_n\}$, $V_2' = \{b_1, b_2, \ldots, b_m\}$ and $E' = \{< v_x,b_y > \mid v_x \in V_1', b_y \in V_2' \}$. Let $d_{v_x,b_y} = 0$, if v_x is an end point of the corresponding edge of vertex b_y; and $= n \times m$, otherwise. An illustration, where $n = 7$ and $m = 9$, is shown in Figure 6.4.

Let us now argue that there exists a vertex cover of size C or less in G if and only if there is a feasible subset of V_1' in G' to satisfy that the sum of activation cost and communication cost is

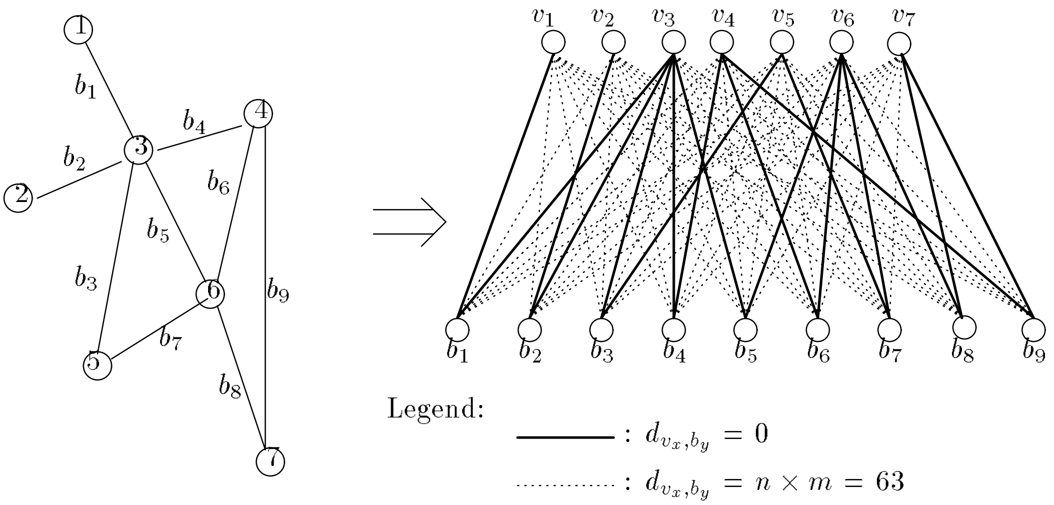


Figure 6.4: An illustration about how to transform a graph to a UCTR instance

$m \times C$ or less. Suppose there is a vertex cover of size C, then for each vertex b_y ($= < u,v >$) in V_2', at least one of u and v belongs to the vertex cover. By selecting all the vertices in the vertex cover into the subset of V_1', we know that the sum in equation (6.3) will be $m \times C$. Since $C \leq n$, it implies that $m \times C \leq n \times m$.

Conversely, for any feasible subset $V_k \subseteq V_1'$ such that the total cost is equal to or less than mC, we can see that the second term of equation (6.3) (i.e. the sum of communication cost) must be zero. Suppose, for some $g_y \in V_2'$, the minimum communication cost between g_y and vertices in V' is nonzero, then the communication cost will be at least $m \times n$. Since $C \leq n$, it implies that $m \times n \geq m \times C$. The total cost in equation (6.3) will be greater than $m \times C$, which is a contradiction. Thus the minimum communication cost between any vertex in V_2' and any vertex in V_k is zero. It means that at least one of two end points of each edge in E belongs to V_k. Since, there is at most C vertices in V_k (the activation cost for each vertex is m), and by selecting the vertices in V_k, we obtain a vertex cover of size C or less in G.

□

Theorem 6.2 *The problem, MCRP-SP for computation-intensive applications, is NP-complete.*

[**Proof**]: From Lemma 6.3 , we know that only the forker in an SP graph of type T_{and} needs to run on more than one processor. Consider the following recognition version of Problem (6.1) for SP graphs of type T_{and}:

Given a system of n processors, an SP graph $G^a = (V^a,E^a)$ of type T_{and}, its assignment graph H and two positive integers m and r. Let r be a multiple of m, $V^a = \{s,\ t,\ 1,2,\ldots,r\}$ and E^a

$= \{< s,i > \mid i = 1,2,\ldots,r\} \cup \{< i,t > \mid i = 1,2,\ldots,r\}$. Task s (t) is the forker (joiner) of G^a. Execution cost $e_{i,p}$ and communication cost $\mu_{i,j}(p,q)$ are defined in H, $\forall < i,j > \in E^a$ and $\forall\ 1 \leq p,q \leq n$. Integer variable $X_{i,p} = 1$ if task i is assigned to processor p; and $= 0$, otherwise. When a positive integer $K \leq r$ is given, is there an assignment of $X_{i,p}$'s, such that

$$[\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,\ 1\leq q\leq n} \min_{X_{i,p}=1} (\mu_{i,j}(p,q) * X_{j,q})\ ] \leq K?$$

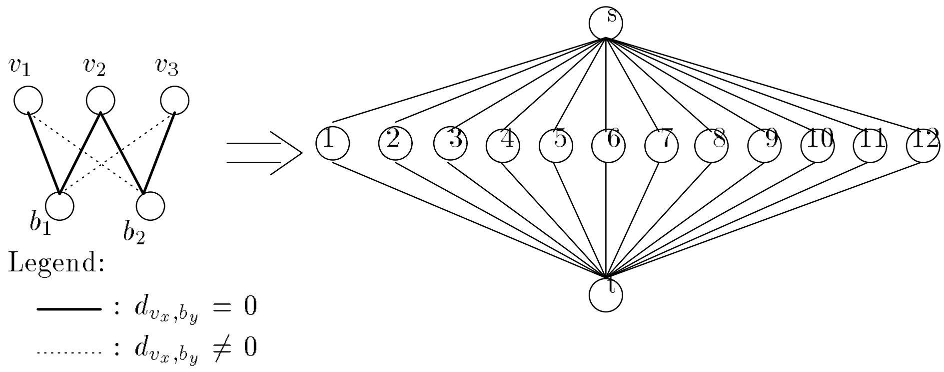
$$\text{where } \sum_{i,p} X_{i,p} = 1,\ \forall i \neq s,\ \text{ and } \sum_{i,p} X_{i,p} \geq 1,\ \text{ if } i = s. \tag{6.4}$$

We shall transform the UCTR problem to this problem. Given any graph $G' = (V_1' \cup V_2', E')$ considered in UCTR problem, we construct an SP graph of type T_{and}, $G^a = (V^a, E^a)$, and its assignment graph H, such that G has a feasible subset of V_1' to allow the sum in equation (6.3) is K or less if and only if there is an assignment of $X_{i,p}$'s for G^a and H to satisfy equation (6.4). Let $\mid V_1' \mid = n$, $\mid V_2' \mid = m$, then the unit cost $f = n \times m$. Assign $r = m \times f$ $(= n \times m^2)$ and $K = n \times m$. The forker and joiner of G^a are s and t respectively. Then $V^a = \{s, t, 1, 2, \ldots, r\}$ and $E^a = \{< s,i > \mid i = 1,2,\ldots,r\} \cup \{< i,t > \mid i = 1,2,\ldots,r\}$. We assign the execution costs and communication costs in H as follows. An illustration, where $m = 2$ and $n = 3$, is shown in Figure 6.5.

- $\forall\ 1 \leq p \leq n$, $e_{s,p} = m$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p \leq n$, if $p = 1$ then $e_{i,p} = 0$ else $e_{i,p} = r$.
- $\forall\ 1 \leq p \leq n$, if $p = 1$ then $e_{t,p} = 0$ else $e_{t,p} = r$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p \leq n$, let $q = (i-1)$ div $(m \times n)$, where div is the integral division. If $d_{v_p,b_{q+1}} \neq 0$ then $\mu_{s,i}(p,1) = 1$ else $\mu_{s,i}(p,1) = 0$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p \leq n$, $\forall\ q \neq 1$, $\mu_{s,i}(p,q) = 0$.
- $\forall\ 1 \leq i \leq r$, $\forall\ 1 \leq p,q \leq n$, $\mu_{i,t}(p,q) = 0$.

It is easy to verify that the SP graph constructed by the above rules is of type T_{and} and computation-intensive. For each node in V_2' of G', we create f nodes in G^a, where the communication cost between each node and source s is either one or zero.

Let us now argue that there exists a feasible subset of V_1' for UCTR problem if and only if there exists a valid assignment of $X_{i,p}$'s such that the total sum in equation (6.4) is K or less. Suppose a feasible subset V_k of V_1' exists such that the sum in equation (6.3) is C $(\leq K)$. Let V_1' be $\{v_1, v_2, \ldots, v_n\}$ Then we can obtain a valid assignment by letting $X_{i,1} = 1$, $X_{i,2} = 0$, $\ldots$, $X_{i,n} =$



e table	p=1	p=2	p=3
$e_{s,p}$ =	2	2	2
$e_{1,p}$ =	0	12	12
$e_{2,p}$ =	0	12	12
$e_{3,p}$ =	0	12	12
$e_{4,p}$ =	0	12	12
$e_{5,p}$ =	0	12	12
$e_{6,p}$ =	0	12	12
$e_{7,p}$ =	0	12	12
$e_{8,p}$ =	0	12	12
$e_{9,p}$ =	0	12	12
$e_{10,p}$ =	0	12	12
$e_{11,p}$ =	0	12	12
$e_{12,p}$ =	0	12	12
$e_{t,p}$ =	0	12	12

μ table	$p = 1$	$p = 2$	$p = 3$
$\mu_{s,1}(p,1) =$	0	0	1
$\mu_{s,2}(p,1) =$	0	0	1
$\mu_{s,3}(p,1) =$	0	0	1
$\mu_{s,4}(p,1) =$	0	0	1
$\mu_{s,5}(p,1) =$	0	0	1
$\mu_{s,6}(p,1) =$	0	0	1
$\mu_{s,7}(p,1) =$	1	0	0
$\mu_{s,8}(p,1) =$	1	0	0
$\mu_{s,9}(p,1) =$	1	0	0
$\mu_{s,10}(p,1) =$	1	0	0
$\mu_{s,11}(p,1) =$	1	0	0
$\mu_{s,12}(p,1) =$	1	0	0
$\mu_{s,i}(p,q) = 0, \forall\ 1 \leq i \leq 12$, and for p = 1,2,3 and q = 2,3			
$\mu_{i,t}(p,q) = 0, \forall\ 1 \leq i \leq 12$, and $\forall\ 1 \leq p,\ q \leq 3$			

Figure 6.5: An illustration about how to transform a UCTR instance to a T_{and} SP graph

0, $\forall$ $1 \leq i \leq r$, and $X_{t,1} = 1$, $X_{t,2} = 0$, ..., $X_{t,n} = 0$, and $X_{s,p} = 1$, if $v_p \in V_k$; and $X_{s,p} = 0$, if $v_p \notin V_k$, $\forall$ $1 \leq p \leq n$. Since each node x in V_2' corresponds to f nodes in G^a, it is sure that the communication cost between node x and any node (v_p) in V_1' is equal to the total communication costs between these f nodes and any task instance of source ($t_{s,p}$) in G^a. By summing up all the costs, we can obtain that the total sum is C. Since $C \leq K \leq n \times m < r$, this is a valid assignment.

Conversely, if there exists an assignment of $X_{i,p}$'s such that the sum in equation (6.4) is K or less, then the following must be true that $X_{i,1} = 1$, $X_{i,2} = 0$, ..., $X_{i,n} = 0$, $\forall$ $1 \leq i \leq r$, and $X_{t,1} = 1$, $X_{t,2} = 0$, ..., $X_{t,n} = 0$. It is because for some $p \neq 1$, if $X_{i,p} = 1$ then the sum must be greater than r, which causes a conflict. Hence the second term in equation (6.4) must be zero. Thus, we may obtain a subset of V_1 for UCTR problem by selecting node $x \in V_1$ if $X_{s,x}$ equals 1. Since the first term in equation (6.3) is equivalent to the first term in equation (6.4), the total sum for UCTR problem will be also K or less then.

$\square$

6.4 Optimal Replication for SP Graphs of Type T_{and}

In this section, we develop the branch-and-bound algorithm to find an optimal solution for T_{and} subgraphs. The non-forker nodes only need to run on one processor. An optimal assignment of non-forker nodes can be done after an optimal replication for forkers is obtained.

6.4.1 A Branch-and-Bound Method for Optimal Replication

Consider a T_{and} SP graph with forker-joiner pair (s,h) shown in Figure 6.6 . There are B subgraphs connected by s and h. These B subgraphs have a parallel-and relationship. Since the joiner h has only one copy in optimal solution (i.e. $\sum_{p=1}^{n} X_{h,p} = 1$), we decompose the minimum cost replication problem $\mathcal{P}$ for a T_{and} SP graph into n subproblems $\mathcal{P}^q$, $q = 1, 2, \ldots, n$, where $\mathcal{P}^q$ is to find the minimum cost when the joiner is assigned to processor q (i.e. $X_{h,q} = 1$).

Given a joiner instance $t_{h,q}$, subgraphs G_b's, $b = 1, 2, \ldots, B$, and the minimum costs $C^b_{p,q}$'s between each forker instance $t_{s,p}$ and joiner instance $t_{h,q}$, $\forall$ $1 \leq p \leq n$ and $1 \leq b \leq B$. we further decompose problem $\mathcal{P}^q$ into n subproblems $\mathcal{P}^q_k$, $k = 1, 2, \ldots, n$, where k is the number of replicated copies that the forker s has. Basically, $\mathcal{P}^q_k$ means the problem of finding an optimal replication for k copies of forker s where the joiner h is assigned to processor q. Since the problem of finding an optimal replication for forker s is NP-complete, we propose a branch-and-bound algorithm for each subproblem $\mathcal{P}^q_k$.

We sort the forker instances according to their execution costs $e_{s,p}$'s into non-decreasing order. Without loss of generality, we assume $e_{s,1} \leq e_{s,2} \leq \ldots \leq e_{s,n}$. We represent all the possible combinations that s may be replicated by a combination tree with $\binom{n}{k}$ leaf nodes. To make the solution efficient, we shall not consider all combinations since it is time-consuming. We apply a least-cost branch-and-bound algorithm to find an optimal solution by traversing a small portion of the combination tree.

During the search, we maintain a variable $\hat{z}$ to record the minimum value known so far. The search is done by the expansion of intermediate nodes. Each intermediate node v at level y represents a combination of y out of n forker instances. The expansion of node v generates at most $n - y$ child nodes, while each child node inherits y forker instances from v and adds one distinct forker instance to itself. For example, if node v is represented by $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y} \succ$, where $i_1 < i_2 < \ldots < i_y$, then $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y}, t_{s,i_y+j} \succ$ represents a possible child node of v, $\forall\ 1 \leq j \leq n - i_y$. A combination tree, where $k = 4$ and $n = 6$, is shown in Figure 6.7. At any intermediate node of a combination tree, we apply an estimation function to compute the least cost this node can achieve. If the estimated cost is greater than $\hat{z}$, then we prune the node and the further expansion of the node is not necessary. Otherwise, we insert this node along with its estimated cost into a queue. The nodes in the queue are sorted into non-decreasing order of their estimated costs, where the first node of the queue is always the next one to be expanded. When the expansion reaches a leaf node, the actual cost of this leaf is computed. If the cost is less than $\hat{z}$, we update $\hat{z}$. The algorithm terminates when the queue is empty.

The Estimation Function

The proposed branch-and-bound algorithm is characterized by the estimation function. Let node v be at level y of the combination tree associated with subproblem $\mathcal{P}_k^q$ and be represented by $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y} \succ$, where $i_1 < i_2 < \ldots < i_y$. Any leaf node that can be reached from node v needs $k - y$ more forker instances. Let $\ell = \prec j_1, j_2, \ldots, j_{k-y} \succ$ be a tuple of $k - y$ instances chosen from the remaining $n - i_y$ instances, where $j_1 < j_2 < \ldots < j_{k-y}$. Let L be the set of all possible ℓ's. Let $g(v)$ be the smallest cost among all leaf nodes that can be reached from node v.

$$g(v) = \sum_{a=1}^{y} e_{s,i_a} \ + \ \min_{\ell \in L}[\sum_{j_x \in \ell} e_{s,j_x} \ + \ \sum_{b=1}^{B} \min_{p=i_1,i_2,\ldots,i_y \text{ or } p \in \ell} (C_{p,q}^b)\ ] \ + \ e_{h,q}.$$

Since the complexity involved in computing $g(v)$ is $\binom{n-i_y}{k-y}$, we use the following estimation function $est(v)$ to approximate $g(v)$:

$$est(v) = \sum_{a=1}^{y} e_{s,i_a} \ + \sum_{j=i_y+1}^{i_y+k-y} e_{s,j} \ + \ \sum_{b=1}^{B} \min_{p=i_1,i_2,\ldots,i_y,i_y+1,i_y+2,\ldots,n} (C_{p,q}^b) \ + \ e_{h,q}. \tag{6.5}$$

Since

$$\sum_{j=i_y+1}^{i_y+k-y} e_{s,j} \leq \sum_{j_x \in \ell} e_{s,j_x} \quad \text{and} \quad \sum_{b=1}^{B} \min_{p=i_y+1,i_y+2,\ldots,n} (C^b_{p,q}) \leq \sum_{b=1}^{B} \min_{p\in\ell} (C^b_{p,q}) \, ,$$

it is easy to see that $est(v) \leq g(v)$. Hence, we use $est(v)$ as the lower bound of the objective function at node v.

The Proposed Algorithm

Three parameters of the branch-and-bound algorithm are joiner instance ($t_{h,q}$), the number of processors that forker s is allowed to run (k), and the up-to-date minimum cost ($\hat{z}$). The algorithm $BB(k, q, \hat{z})$ is shown in Figure 6.10.

The MCRP-SP problem can be solved by invoking $BB(k, q, \hat{z})$ n^2 times with parameters set to different values. $BB(k, q, \hat{z})$ solves the problem $\mathcal{P}^q_k$, while the whole procedure, shown in Figure 6.11, solves $\mathcal{P}$.

6.4.2 Performance Evaluation

The essence of the branch-and-bound algorithm is the expansion of the intermediate nodes. Upon the removal of a node from the queue its children are generated and their estimated values are computed. If the estimation function performs well and gives a tight lower bound of objective function, the number of expanded nodes should be small. An optimal solution can be found as soon as possible.

We conduct two sets of experiments to evaluate the performance of the proposed solution. The performance indices we consider are the number of enqueued intermediate nodes (EIM) and the number of visited leaf nodes (VLF) during the search. We calculate EIM and VLF by inserting one counter for each index at lines 13 and 8 of Figure 6.10 respectively. Each time the execution reaches line 13 (8), EIM (VLF) is incremented by 1.

The first set of experiments is on SP graphs of type T_{and} where the communication cost between any two task instances is arbitrary and is generated by random number generator within the range [1,50]. The execution cost for each task instance is also randomly generated within the same range. The second set of experiments is on SP graphs of type T_{and} with the constrain of computation-intensive applications. We vary the size of the problem by assigning different values to the number of processors in the system (n) and the number of parallel-and subgraphs connected by forker and joiner (B). For each size of the problem (n, B), we randomly generate 50 problem instances and solve them. The results, including the average values of EIM and VLF over the solutions of 50

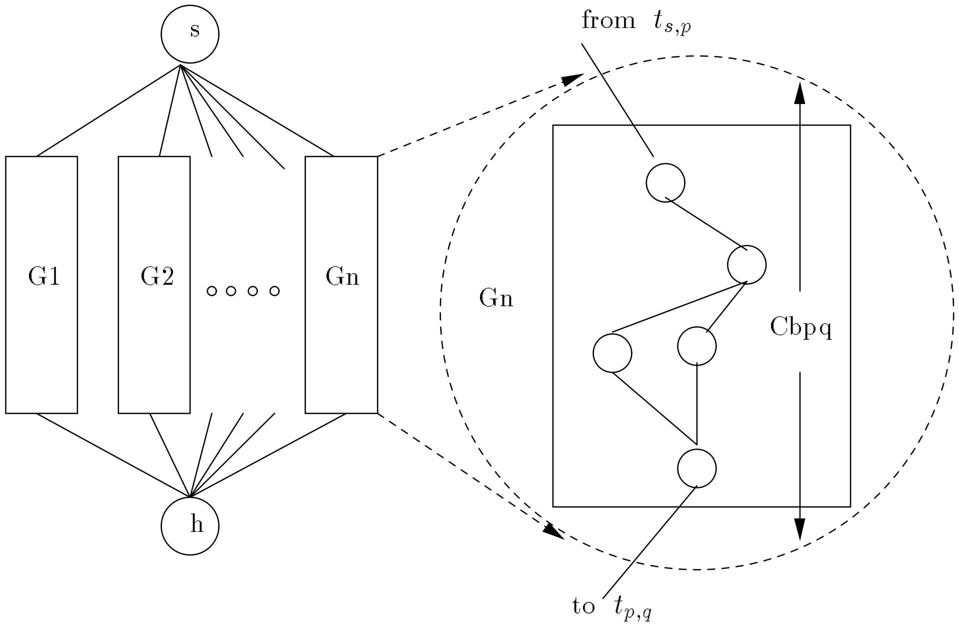


Figure 6.6: A T_{and} SP graph and the graphical interpretation of $C^b_{p,q}$.

problem instances, are summarized in Table 6.1.

From Table 6.1, we find out that the proposed method significantly reduces the number of expansions for intermediate nodes and leaf nodes. For example, for problem size $(n, B) = (20, 40)$, the total number of leaf nodes is 2^{20} (= 1,048,576) if an exhaustive search is applied. However, our algorithm only generates 16,857 nodes on the average, because we apply $est(v)$, $\hat{z}$, and the branch-and-bound approach.

The branch-and-bound approach and the estimation function even perform better for the computation-intensive applications. We can see that EIM and VLF values are much smaller in Set II than those in Set I. In computation-intensive applications an optimal number of replications for the forker is smaller than that in general applications. The $\hat{z}$ value in function $OPT()$ is able to reflect this fact and avoid the unnecessary expansions.

6.5 Sub-Optimal Replication for SP Graphs of Type T_{and}

The branch-and-bound algorithm in section 4.1 yields an optimal solution for T_{and} subgraphs. However, the complexity involved is in exponential time in the worst case. We also attempt to find a near-optimal solution in polynomial time.

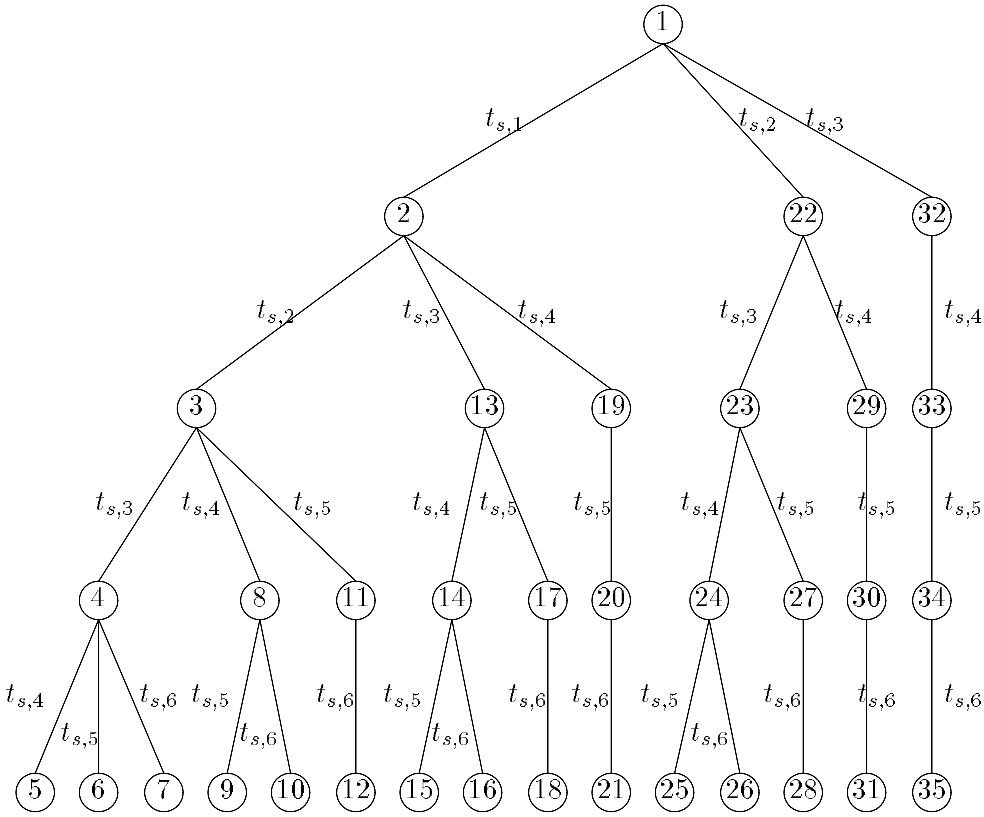


Figure 6.7: A combination tree for the case where $k = 4$ and $n = 6$

6.5.1 Approximation Method

For the problem $\mathcal{P}_k^q$ defined in section 4.1, we exploit an approximation approach to solve it in polynomial time. The approach is based on iterative selection in dynamic programming fashion. Given a joiner instance $t_{h,q}$ and subgraphs G_b, $b = 1, 2, \ldots, B$, and minimum costs $C_{p,q}^b$ between $t_{h,q}$ and $t_{s,p}$, $p = 1, 2, \ldots, n$, and $b = 1, 2, \ldots, B$. We define $Sub(p, b)$ to be the sub-optimal solution for replication of forker s where forker instances $t_{s,1}$, $t_{s,2}$, ..., $t_{s,p}$ and subgraphs G_1, G_2, ..., G_b are taken into consideration.

Strategy 1:

$Sub(p, b)$ can be obtained from $Sub(p-1, b)$ by considering one more forker instance $t_{s,p}$. Strategy 1 consists of two steps. The first step is to initialize $Sub(p, b)$ to be $Sub(p-1, b)$ and to determine if $t_{s,p}$ is to be included into $Sub(p, b)$ or not. If yes, then add $t_{s,p}$ in. The second step is to examine if any instances in $Sub(p-1, b)$ should be removed or not. Due to the possible inclusion of $t_{s,p}$ in the first step, we may obtain a lower cost if we remove some instances $t_{s,i}$'s, $i < p$, and reassign the communications for some graphs G_j's from $t_{s,i}$'s to $t_{s,p}$.

Strategy 2:

$Sub(p, b)$ can also be obtained from $Sub(p, b-1)$ by taking one more subgraph G_b into account. Initially, $Sub(p, b)$ is set to be $Sub(p, b-1)$. The first step is to choose the best forker instance from $t_{s,1}$, $t_{s,2}$, ..., $t_{s,p}$ for G_b. Let the best instance be $t_{s,z}$. The second step is to see if $t_{s,z}$ is in $Sub(p, b)$ or not. If not, a condition is checked to decide whether $t_{s,z}$ should be added in or not. Upon the addition of $t_{s,z}$, we may remove some instances and reassign the communications to achieve a lower cost.

We compare two possible results obtained from the above two strategies and assign the one with lower cost to actual $Sub(p, b)$. By computing in a dynamic programming fashion, $Sub(n, B)$ can be obtained. The algorithm and its graphical interpretation are shown in Figure 6.8.

6.5.2 Performance Evaluation

The complexity involved in each strategy described in section 5.1 is $O(nB)$. Since the solving of $Sub(n, B)$ needs to invoke $n \times B$ times of strategies 1 and 2, the total complexity of solving $Sub(n, B)$ by the approximation method is $O(n^2B^2)$.

We conduct a set of experiments to evaluate the performance of the approximation method. For each problem size (n, B), we randomly generate 50 instances and solve them by using the approximation method and exhaustive searching. We compare the minimum cost obtained from exhaustive searching (EXHAUST) with those from approximation (APPROX) and single assignment solution (SINGLE). The optimal single assignment solution is one in which only one forker instance is allowed. The results are summarized in Table 6.2. From the table, we find that the approximation method yields a tight approximation of the minimum cost. On the contrary, the error range for single copy solution is at least 20%. This again justifies that the replication can lead to a lower cost than an optimal assignment.

6.6 Solution of MCRP-SP for computation-intensive applications

6.6.1 The Solution

Given a computation-intensive application with its SP graph, we generate its parsing tree and assignment graph first. The algorithm finds the minimum weight replication graph from the assignment graph. The optimal solution is then obtained from the minimum weight replication graph.

The algorithm traverses the parsing tree in the postfix order. Namely, during the traversal, an optimal solution of the subtree S_x, induced by an intermediate node x along with all x's descendant nodes, can be found only after the optimal solutions of x's descendant nodes are found. Given an SP graph G and a system S, we know that there is a one-to-one correspondence between each subtree S_x in a parsing tree $T(G)$ and a limb in the assignment graph of G on S. Whenever a child node b of x is visited, the corresponding limb in the assignment graph will be replaced with a a two-layer T_{chain} limb if b is a T_{chain}- or T_{or}-type node; and a one-layer T_{unit} limb if b is a T_{and}-type node. The algorithm is shown in Figure 6.12. A graphical demonstration of how the algorithm solves the problem is shown in Figure 6.9.

Before the replacement of a T_{chain} limb is performed (i.e. x is a T_{chain}-type node), each constituent child limb has been replaced with a T_{unit} or two-layer T_{chain} limb. The shortest path algorithm [Bok87] can be used to compute the weights of the new edges between each node in the source layer and each node in the sink layer of the new T_{chain} limb. The complexity, from lines 05 to 08 of Figure 6.12, in transformation of the limb, corresponding to an intermediate node x with M children, into a two-layer T_{chain} limb is $O(Mn^3)$. An example of illustrating the replacement of a T_{chain} limb is shown from parts (b) to (c) and parts (d) to (e) in Figure 6.9.

For the replacement of a T_{and} limb, we have to compute $C^b_{p,q}$'s. The values can also be computed by the shortest path algorithm. The complexity involved in lines 16 and 17 is $O(Bn^3)$. According to the computational model in section 6.2.2, each task instance s may start its execution if it receives the necessary data from any task instance of its predecessor d. From Lemma 2, we know that the minimum sum of initialization costs of multiple task instances of s will be always from only one task instance of d. Therefore, the initialization of task instance $t_{s,p}$ depends on which task instance of d it communicates. That is why ,in line 19, the communication cost $\mu_{d,s}(i,p)$ is added to the the execution cost of $e_{s,p}$ before $OPT()$ is invoked. And the most significant part of the replacement is to compute the weights on the new edges from the source layer to sink layer. The complexity is $n^2 \times O(OPT())$, which in the worst case is $n^2 2^n$. However, in the average, our OPT function performs pretty well and reduces the complexity significantly. An example of illustrating the replacement of a T_{and} limb is shown from parts (c) to (d) in Figure 6.9.

We also consider to use the approximation method to find the sub-optimal replacement of a T_{and} limb. In that case, function $OPT()$ in line 21 is replaced with $Sub(n, B)$. The total complexity involved is $O(n^4 B^2)$.

Finally, for the replacement of a T_{or} limb, if there are B subgraphs connected between the forker and the joiner, then the complexity will be $O(Bn^2)$ for the new edges and $O(Bn^3)$ for $C^b_{p,q}$'s. An example of illustrating the replacement of a T_{or} limb is shown from parts (a) to (b) in Figure 6.9.

When the traversal reaches the root node of the parsing tree, the result of $FIND()$ will give us either one single layer or two layers, depending on the type of root node. All we have to do is select the lightest of these n (in single layer) or n^2 (in two layers) shortest-path combinations. An optimal replication graph itself is found by combining the shortest paths between the selected nodes that were saved earlier. The whole algorithm has the complexity of

$$O(An^2 2^n) \ + \ \sum_i (R_i n^3) \ + \ \sum_i (C_i n^3)$$

where A is the number of T_{and} limbs, R_i is the number of subgraphs in the ith T_{or} limb, and C_i is the number of layers in the ith T_{chain} limb. This is not greater than $O(Mn^2 2^n)$, where M is the total number of tasks in the SP graph. The complexity of the algorithm is a linear function of M if the number of processors, n, is fixed.

6.6.2 Conclusion Remark

We have focused on MCRP-SP, the optimal replication problem of SP task graphs for computation-intensive applications. The purpose of replication is to reduce inter-processor communication, and to fully utilize the processor power in the system. The SP graph model, which is extensively used in modeling applications in computing systems, is used. The applications considered in this chapter are computation-intensive in which the execution cost of a task is greater than its communication cost. We have proved that MCRP-SP is NP-complete and presented branch-and-bound and approximation methods for SP graphs of type T_{and}. The numerical results show that the algorithm performs very well and avoids a lot of unnecessary searching. Finally, we have presented an algorithm to solve the MCRP-SP problem for computation-intensive applications. The proposed optimal solution has the complexity of $O(n^2 2^n M)$ in the worst case, while the approximation solution is in the complexity of $O(n^4 M^2)$, where n is the number of processors in the system and M is the number of tasks in the graph.

For the applications in which the communication cost between two tasks is greater than the execution cost of a task, the replication can still be used to reduce the total cost. However, in the extreme case where the execution cost of each task is zero, the optimal allocation will be to assign each task to one processor.

$\underline{Rep(p-1,b) \;\rightarrow\; Rep'(p,b)}$:

If $e_{s,p} \leq \sum_{i=1}^{b}([\min_{x \in Rep(p-1,b)}(C^{i}_{x,q})] - C^{i}_{p,q})^{+}$

begin

  $Rep'(p,b) = Rep(p-1,b) \oplus t_{s,p}$

  Reassign&Remove($Rep'(p,b)$)

end

Else $Rep'(p,b) = Rep(p-1,b)$

Legend:

  $(x)^{+} = x$, if $x > 0$.

  $(x)^{+} = 0$, if $x \leq 0$.

$\underline{Rep(p,b-1) \;\rightarrow\; Rep''(p,b)}$:

Let $t_{s,z}$ be the one satisfys

  $\min_{1 \leq i \leq p}(C^{b}_{i,q})$.

If $t_{s,z} \in Rep(p,b-1)$ then

  $Rep''(p,b) = Rep(p,b-1)$

Else

if $e_{s,z} \leq \sum_{i=1}^{b}([\min_{j \in Rep(p,b-1)}(C^{i}_{j,q})] - C^{i}_{z,q})^{+}$

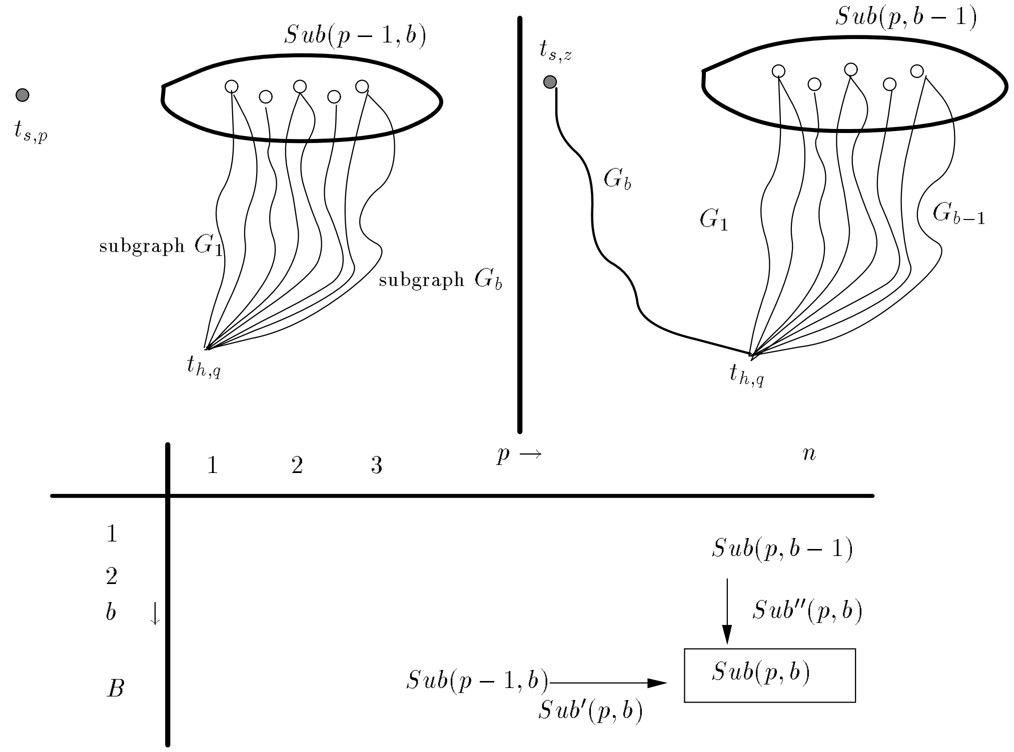
begin

  $Rep''(p,b) = Rep(p-1,b) \oplus t_{s,z}$

  Reassign&Remove($Rep''(p,b)$)

end

Else $Rep''(p,b) = Rep(p,b-1)$



$Sub(p,b) = Min\_Cost(Sub'(p,b), Sub''(p,b))$

Figure 6.8: Pseudo code, graphical demonstration, and dynamic programming table for approximation methods

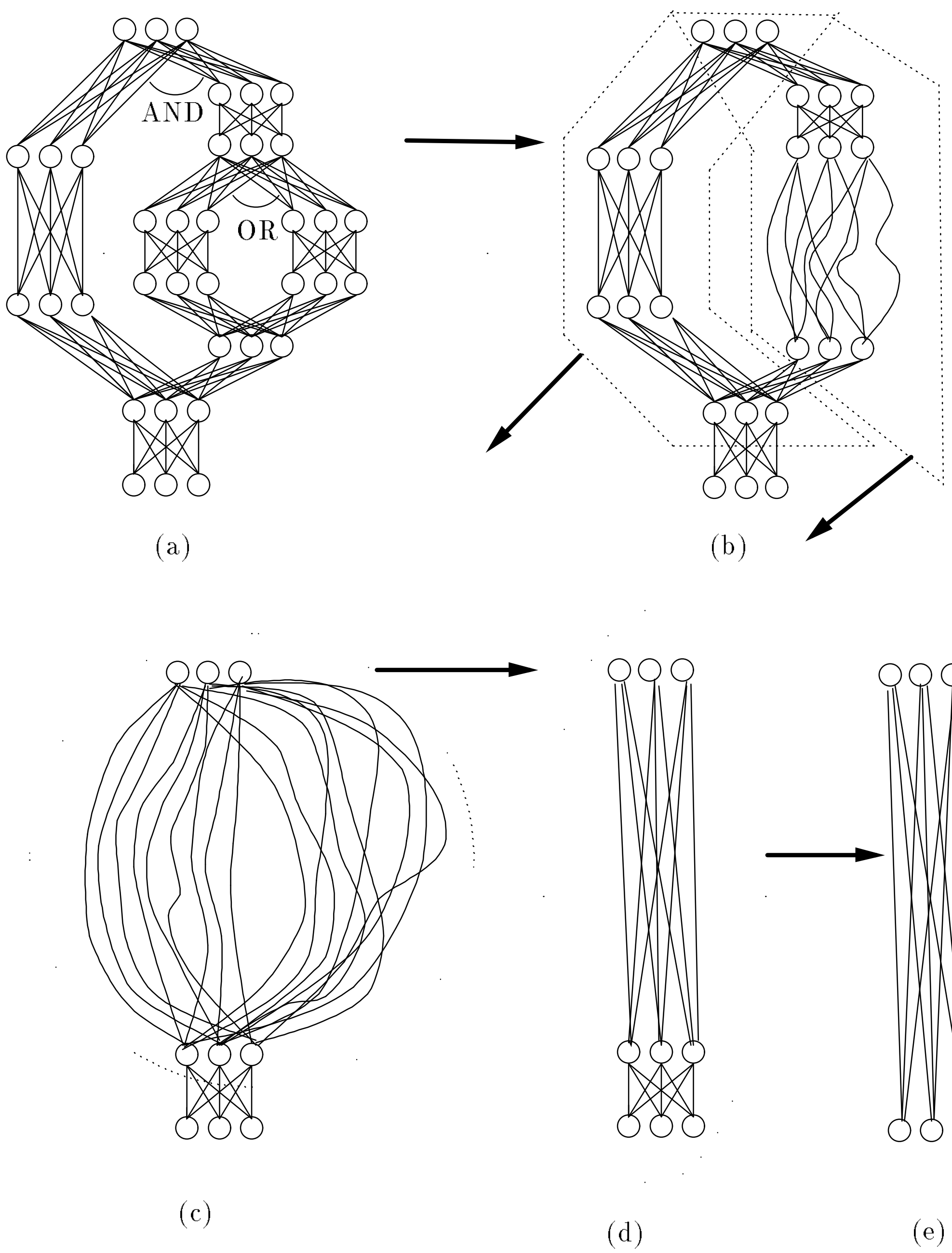


Figure 6.9: A graphical demonstration of how to find an optimal solution for MCRP-SP

Figure 6.10: **Function** $BB(k, q, \hat{z})$: branch-and-bound algorithm for solving problem $\mathcal{P}_k^q$

```
Initialize the queue to be empty;
Insert root node v_0 into the queue;
While the queue is not empty do begin
    Remove the first node u from the queue;
    Generate all child nodes of u ;
    For each generated child node v do begin
        If v is a leaf node (i.e. v is at level k) then
            Compute g(v) by setting L to be φ ;
            Set ẑ = min ( ẑ, g(v));
        else begin /* v is an intermediate node */
            Compute est(v) by (6.5) ;
            If est(v) < ẑ then
                Insert v into the queue according to est(v) ;
        end;
    end;
end;
Return(ẑ).
```

Figure 6.11: **Function** $OPT(C_{p,q}^{b}{}'s,\ e_{s,p}{}'s)$: the optimal solution of MCRP-SP of type T_{and} when $C_{p,q}^{b}$'s and $e_{s,p}$'s are given

```
Sort t_{s,p}'s into a non-decreasing order by values of e_{s,p}'s ;
For q = 1 to n do begin
    Let node v be a leaf node at level 1;
    Set v to be t_{s,1} and k to be 1;
    Compute g(v) by setting L to be φ ;
    Initialize ẑ to be g(v) ;
    For k = 1 to n do
        ẑ = BB(k, q, ẑ) ;
    Set c(q) = ẑ ;
end;
Output the combination with the minimum value
    among c(1), c(2), ..., c(n).
```

Table 6.1: Computation Results for branch-and-bound approach

n	B	Set I EIM[‡]	Set I VLF[‡]	Set II EIM[‡]	Set II VLF[‡]	Total Number of leaves (2^n)
4	20	2	6	4	7	16
	24	3	6	3	6	16
	28	4	7	3	6	16
	32	4	7	3	6	16
	36	4	7	4	7	16
	40	3	6	3	6	16
8	20	36	74	16	51	256
	24	40	75	21	62	256
	28	50	86	26	68	256
	32	63	94	37	78	256
	36	73	96	47	84	256
	40	81	97	50	86	256
12	20	186	558	81	340	4,096
	24	231	639	102	398	4,096
	28	349	839	167	543	4,096
	32	451	967	204	617	4,096
	36	454	984	269	720	4,096
	40	636	1,186	301	780	4,096
16	20	758	3,216	203	1,175	65,536
	24	1,065	4,161	329	1,711	65,536
	28	1,335	4,862	546	2,496	65,536
	32	1,884	6,250	726	3,127	65,536
	36	2,322	7,227	839	3,493	65,536
	40	2,880	8,511	1,179	4,510	65,536
20	20	2,026	12,042	389	3,079	1,048,576
	24	3,579	18,866	761	5,280	1,048,576
	28	5,551	27,018	1,227	7,905	1,048,576
	32	6,405	30,521	1,709	10,357	1,048,576
	36	9,517	40,767	2,681	15,032	1,048,576
	40	11,651	48,087	3,086	16,857	1,048,576

‡: Each value shown is the average value over 50 runs.

Table 6.2: Simulation Results for Approximation Method

n	B	SINGLE‡	APPROX‡	EXHAUST‡	S-error %	A-error %
4	20	2876	2407	2400	20	0.28
	24	3463	2835	2831	22	0.16
	28	4032	3264	3259	24	0.18
	32	4606	3678	3673	25	0.11
	36	5198	4084	4082	27	0.05
	40	5790	4514	4514	28	0.00
8	20	2794	2282	2250	24	1.46
	24	3356	2672	2636	27	1.38
	28	3931	3060	3028	30	1.05
	32	4540	3443	3413	33	0.88
	36	5127	3831	3800	35	0.80
	40	5683	4215	4192	36	0.55
12	20	2767	2213	2161	28	2.42
	24	3359	2592	2542	32	1.99
	28	3912	2996	2941	33	1.88
	32	4491	3364	3299	36	1.97
	36	5063	3736	3676	38	1.62
	40	5610	4101	4043	39	1.43
16	20	2733	2167	2111	29	2.66
	24	3287	2558	2492	32	2.66
	28	3844	2932	2865	34	2.31
	32	4393	3315	3240	36	2.32
	36	4991	3659	3584	39	2.10
	40	5558	4045	3970	40	1.89

‡: Each value shown is the average value over 50 runs.

$$\mathrm{S-error\%} = \frac{\mathrm{SINGLE} - \mathrm{EXHAUST}}{\mathrm{EXHAUST}} \times 100\%.$$

$$\mathrm{A-error\%} = \frac{\mathrm{APPROX} - \mathrm{EXHAUST}}{\mathrm{EXHAUST}} \times 100\%.$$

Figure 6.12: **Algorithm** $FIND(S_x)$: the algorithm for finding the shortest path combinations from the limb which corresponds to the subtree S_x induced by an intermediate node x and all x's descendant nodes in a parsing tree

01 **Case** of the type of intermediate node x:
02 Type T_{chain} :
03 **For** b = the *first* child node of x **to** the *last* one **do**
04 $FIND(S_b)$; /* Now the limb corresponding to S_b is replaced */
05 Replace the limb corresponding to S_x with a two-layer T_{chain} limb where
06 the source (sink) layer of the old limb is the source (sink) layer of new 2-layer limb;
07 Put weights on the edges between source and sink layers equal to the shortest path
08 between the corresponding nodes;
09
10 Type T_{and} : /* Let x = [T_{and}, forker s, joiner h] */
11 Let d be the predecessor of forker s in G (i.e. $< d, s > \in V$);
12 Let B be the number of child nodes of x in the parsing tree;
13 /* I.e. there are B subgraphs connected by s and h */
14 **For** b = the first child node of x **to** the B-th child of x **do**
15 $FIND(S_b)$; /* Now the limb corresponding to S_b is replaced */
16 **For** $p = 1$ **to** n, $q = 1$ **to** n **and** $b = 1$ **to** B **do**
17 Compute the minimum replication cost $C^b_{p,q}$ from $t_{s,p}$ to $t_{h,q}$ w.r.t. child b ;
18 **For** $i = 1$ **to** n **do begin**
19 **For** $p = 1$ **to** n **do** $E_{s,p} = \mu_{d,s}(i,p) + e_{s,p}$;
20 /* $E_{s,p}$ accounts for initialization by $t_{d,i}$ and execution cost itself. */
21 **For** $q = 1$ **to** n **do** $\mu_{d,h}(i,q) = OPT({C^b_{p,q}}'s, E_{s,p}'s)$;
22 /* Create new edges from $t_{d,i}$'s to $t_{h,q}$'s */
23 **end**;
24 Replace the T_{and} limb with a T_{unit} limb, where source layer = sink layer = layer h,
25 and there are new edges from layer d to layer h;
26
27 Type T_{or} : /* Let x = [T_{or}, forker s, joiner h] */
28 Use the same method described above from lines 12 to 17 to compute $C^b_{p,q}$'s ;
29 Replace the T_{or} limb with a two-layer T_{chain} limb, where
30 the source (sink) layer of T_{or} limb is the source (sink) layer of T_{chain} limb and
31 $\mu_{s,h}(p,q) = \min_b(C^b_{p,q})$, $\forall$ p and q ;
32 **end case**;
33 Save the shortest paths between any node in source layer and any node in sink layer for future reference.

Chapter 7

Conclusion

The allocation problem is one of the fundamental issues of using computing systems. Tasks of an application are allocated to the available resources in the systems. The execution of the tasks has to meet the specified resource and synchronization constraints. For real-time applications, the timing constraints add one more dimension to the complexity of the allocation problem.

As a consequence of the timing constraints, the allocation problem in real-time systems has to address the feasibility and schedulability of the allocation. In Chapters 3 and 4, we address this problem by explicitly generating the time-based multiprocessor calendar. A calendar is a time table which maps each task/communication to time lines. The start time and finish time information are used at run time to dispatch the tasks. If the analysis of the calendar satisfies the given resource, synchronization and timing constraints, then a feasible allocation has been obtained.

In general, the specifications of the timing constraints are derived from the requirements and properties of the applications. Traditionally, the absolute timing model is assumed. In the absolute model, the timing constraints are expressed in terms of period, ready time and deadline. Many scheduling techniques are developed under such a model. However, these techniques do not adequately handle the inter-task and/or inter-execution temporal dependencies. In this dissertation, we have considered a hybrid timing model which combines the absolute and relative timing constraints. Solutions to the allocation problem of periodic tasks with the hybrid timing constraints are presented in Chapters 3 and 4.

Another research direction we have taken is to investigate how the replication technique can be applied to improve the proposed allocation scheme. To achieve our objective, we have considered a different computation model in which the purpose of the replication is not for the sake of fault tolerance but for reducing the total execution cost or increasing the degree of schedulability. In Chapters 5 and 6, we have shown how to exploit the replication techniques for real-time and non-real-time problems respectively.

7.1 Future Research

The approach we have taken so far is for the hard-real-time systems in which failure to meet the timing constraints can result in disastrous consequences. Under such consideration, the way to guarantee the critical timing constraints is to allocate and schedule tasks *a priori*. However, the allocation of tasks in soft-real-time may take different approaches. Usually the violation of timing constraints in soft-real-time systems is associated with some penalty function and the objective of the allocation is to minimize the value of the penalty function. One issue that we have not explored is to consider this objective, and algorithms to handle the allocation problem of tasks in soft-real-time systems. We believe that the techniques presented in this dissertation can be extended to incorporate the allocation of tasks in soft-real-time systems.

When we consider applications with relative timing constraints, many open problems exist. In this dissertation, we considered one model of relative timing constraints and task characteristics. As these change, our techniques have to be extended to address them.

Appendix A

The Specification of Boeing 777 AIMS (A sanitized version):

From 11:2:3 5 Hz 0.600 ms to 1:0:1 length 11.967 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 2:0:1 length 11.967 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 3:0:1 length 11.967 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 4:0:1 length 11.967 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 5:17:1 length 49.000 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 6:17:1 length 49.000 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 7:17:1 length 49.000 us latency 200000 us
From 11:2:3 5 Hz 0.600 ms to 8:17:1 length 49.000 us latency 200000 us
From 12:0:1 5 Hz 0.000 ms to 16:16:3 length 16.300 us latency 68993 us
From 12:0:1 5 Hz 0.000 ms to 16:16:5 length 36.333 us latency 100000 us
From 12:0:1 5 Hz 0.000 ms to 17:16:3 length 16.300 us latency 68993 us
From 12:0:1 5 Hz 0.000 ms to 17:16:5 length 36.333 us latency 100000 us
From 12:0:1 5 Hz 0.000 ms to 18:16:3 length 16.300 us latency 68993 us
From 12:0:1 5 Hz 0.000 ms to 18:16:5 length 36.333 us latency 100000 us
From 12:0:1 5 Hz 0.000 ms to 18:29:9 length 383.333 us latency 200000 us
From 12:0:1 5 Hz 0.000 ms to 19:16:3 length 16.300 us latency 68993 us
From 12:0:1 5 Hz 0.000 ms to 19:16:5 length 36.333 us latency 100000 us
From 12:0:1 5 Hz 0.000 ms to 20:2:3 length 5.633 us latency 200000 us
From 12:0:1 5 Hz 0.000 ms to 22:2:3 length 5.633 us latency 200000 us
From 12:0:2 10 Hz 0.000 ms to 18:29:1 length 0.833 us latency 200000 us
From 12:0:3 40 Hz 0.000 ms to 18:29:1 length 0.833 us latency 200000 us
From 12:0:3 40 Hz 0.000 ms to 19:26:22 length 1.667 us latency 50000 us
From 13:0:1 5 Hz 0.000 ms to 16:16:3 length 16.300 us latency 68993 us
From 13:0:1 5 Hz 0.000 ms to 16:16:5 length 36.333 us latency 100000 us
From 13:0:1 5 Hz 0.000 ms to 17:16:3 length 16.300 us latency 68993 us
From 13:0:1 5 Hz 0.000 ms to 17:16:5 length 36.333 us latency 100000 us
From 13:0:1 5 Hz 0.000 ms to 18:16:3 length 16.300 us latency 68993 us
From 13:0:1 5 Hz 0.000 ms to 18:16:5 length 36.333 us latency 100000 us
From 13:0:1 5 Hz 0.000 ms to 18:29:9 length 383.333 us latency 200000 us
From 13:0:1 5 Hz 0.000 ms to 19:16:3 length 16.300 us latency 68993 us
From 13:0:1 5 Hz 0.000 ms to 19:16:5 length 36.333 us latency 100000 us
From 13:0:1 5 Hz 0.000 ms to 20:2:3 length 5.633 us latency 200000 us
From 13:0:1 5 Hz 0.000 ms to 22:2:3 length 5.633 us latency 200000 us
From 13:0:2 10 Hz 0.000 ms to 18:29:1 length 1.667 us latency 200000 us
From 13:0:3 40 Hz 0.000 ms to 18:29:1 length 0.833 us latency 200000 us
From 13:0:3 40 Hz 0.000 ms to 19:26:22 length 1.667 us latency 50000 us
From 14:0:1 5 Hz 0.000 ms to 16:16:3 length 16.300 us latency 68993 us
From 14:0:1 5 Hz 0.000 ms to 16:16:5 length 36.333 us latency 100000 us
From 14:0:1 5 Hz 0.000 ms to 17:16:3 length 16.300 us latency 68993 us
From 14:0:1 5 Hz 0.000 ms to 17:16:5 length 36.333 us latency 100000 us
From 14:0:1 5 Hz 0.000 ms to 18:16:3 length 16.300 us latency 68993 us

From 14:0:1 5 Hz 0.000 ms to 18:16:5 length 36.333 us latency 100000 us
From 14:0:1 5 Hz 0.000 ms to 18:29:9 length 383.333 us latency 200000 us
From 14:0:1 5 Hz 0.000 ms to 19:16:3 length 16.300 us latency 68993 us
From 14:0:1 5 Hz 0.000 ms to 19:16:5 length 36.333 us latency 100000 us
From 14:0:1 5 Hz 0.000 ms to 20:2:3 length 5.633 us latency 200000 us
From 14:0:1 5 Hz 0.000 ms to 22:2:3 length 5.633 us latency 200000 us
From 14:0:2 10 Hz 0.000 ms to 18:29:1 length 2.500 us latency 200000 us
From 14:0:3 40 Hz 0.000 ms to 18:29:1 length 0.833 us latency 200000 us
From 15:0:1 5 Hz 0.000 ms to 16:16:3 length 16.300 us latency 68993 us
From 15:0:1 5 Hz 0.000 ms to 16:16:5 length 36.333 us latency 100000 us
From 15:0:1 5 Hz 0.000 ms to 17:16:3 length 16.300 us latency 68993 us
From 15:0:1 5 Hz 0.000 ms to 17:16:5 length 36.333 us latency 100000 us
From 15:0:1 5 Hz 0.000 ms to 18:16:3 length 16.300 us latency 68993 us
From 15:0:1 5 Hz 0.000 ms to 18:16:5 length 36.333 us latency 100000 us
From 15:0:1 5 Hz 0.000 ms to 18:29:9 length 383.333 us latency 200000 us
From 15:0:1 5 Hz 0.000 ms to 19:16:3 length 16.300 us latency 68993 us
From 15:0:1 5 Hz 0.000 ms to 19:16:5 length 36.333 us latency 100000 us
From 15:0:1 5 Hz 0.000 ms to 20:2:3 length 5.633 us latency 200000 us
From 15:0:1 5 Hz 0.000 ms to 22:2:3 length 5.633 us latency 200000 us
From 15:0:2 10 Hz 0.000 ms to 18:29:1 length 2.500 us latency 200000 us
From 15:0:3 40 Hz 0.000 ms to 18:29:1 length 0.833 us latency 200000 us
From 16:16:1 5 Hz 0.563 ms to 18:29:1 length 4.567 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 12:0:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 13:0:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 14:0:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 15:0:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 16:19:5 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 16:21:5 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 16:22:4 length 6.633 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 17:16:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 17:16:2 length 12.467 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 17:16:3 length 21.633 us latency 93918 us
From 16:16:2 5 Hz 1.126 ms to 17:19:5 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 17:21:5 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 17:22:4 length 6.633 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:16:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:16:2 length 12.467 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:16:3 length 21.633 us latency 93918 us
From 16:16:2 5 Hz 1.126 ms to 18:20:4 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:29:2 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:32:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:33:2 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 18:35:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 19:16:1 length 0.833 us latency 200000 us

From 16:16:2 5 Hz 1.126 ms to 19:16:2 length 12.467 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 19:16:3 length 21.633 us latency 93918 us
From 16:16:2 5 Hz 1.126 ms to 19:20:4 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 19:26:22 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 19:27:1 length 0.833 us latency 200000 us
From 16:16:2 5 Hz 1.126 ms to 19:28:1 length 0.833 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 12:0:1 length 5.800 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 13:0:1 length 5.800 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 14:0:1 length 5.800 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 15:0:1 length 5.800 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 16:16:5 length 4.200 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 17:16:5 length 39.167 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 18:16:5 length 39.167 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 18:29:10 length 9.300 us latency 200000 us
From 16:16:5 5 Hz 3.041 ms to 19:16:5 length 39.167 us latency 200000 us
From 16:17:1 10 Hz 0.980 ms to 17:17:1 length 5.167 us latency 200000 us
From 16:17:1 10 Hz 0.980 ms to 18:17:1 length 5.167 us latency 200000 us
From 16:17:1 10 Hz 0.980 ms to 18:29:9 length 94.667 us latency 200000 us
From 16:17:1 10 Hz 0.980 ms to 19:17:1 length 5.167 us latency 200000 us
From 16:17:1 10 Hz 0.980 ms to 20:2:3 length 447.733 us latency 200000 us
From 16:17:1 10 Hz 0.980 ms to 22:2:3 length 447.733 us latency 200000 us
From 16:19:1 80 Hz 1.180 ms to 16:22:4 length 11.133 us latency 200000 us
From 16:19:1 80 Hz 1.180 ms to 17:22:4 length 11.133 us latency 200000 us
From 16:19:1 80 Hz 1.180 ms to 18:32:1 length 2.900 us latency 200000 us
From 16:19:4 40 Hz 2.000 ms to 16:22:4 length 13.500 us latency 200000 us
From 16:19:4 40 Hz 2.000 ms to 17:22:4 length 13.500 us latency 200000 us
From 16:19:5 5 Hz 0.100 ms to 16:22:4 length 9.300 us latency 200000 us
From 16:19:5 5 Hz 0.100 ms to 17:19:5 length 2.967 us latency 200000 us
From 16:19:5 5 Hz 0.100 ms to 17:22:4 length 9.300 us latency 200000 us
From 16:19:5 5 Hz 0.100 ms to 18:29:10 length 9.300 us latency 200000 us
From 16:21:2 40 Hz 3.150 ms to 16:22:2 length 0.000 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 16:22:2 length 11.967 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 16:22:3 length 5.033 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 17:21:3 length 4.033 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 17:22:2 length 7.933 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 17:22:3 length 5.033 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 18:29:10 length 5.033 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 18:32:1 length 7.933 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 19:26:22 length 14.800 us latency 150000 us
From 16:21:3 20 Hz 2.800 ms to 19:27:1 length 6.867 us latency 200000 us
From 16:21:3 20 Hz 2.800 ms to 19:28:1 length 6.867 us latency 200000 us
From 16:21:4 10 Hz 5.900 ms to 16:22:3 length 0.000 us latency 200000 us
From 16:21:4 10 Hz 5.900 ms to 16:22:4 length 2.367 us latency 200000 us
From 16:21:4 10 Hz 5.900 ms to 17:22:4 length 2.367 us latency 200000 us

From 16:21:5 5 Hz 14.600 ms to 16:22:4 length 0.000 us latency 200000 us
From 16:21:5 5 Hz 14.600 ms to 17:21:5 length 35.800 us latency 200000 us
From 16:21:5 5 Hz 14.600 ms to 18:29:10 length 8.300 us latency 200000 us
From 16:21:5 5 Hz 14.600 ms to 18:32:1 length 16.833 us latency 200000 us
From 16:21:5 5 Hz 14.600 ms to 18:33:2 length 16.833 us latency 200000 us
From 16:21:5 5 Hz 14.600 ms to 19:26:22 length 3.967 us latency 200000 us
From 16:22:2 20 Hz 2.050 ms to 16:21:3 length 0.000 us latency 200000 us
From 16:22:2 20 Hz 2.050 ms to 17:22:2 length 2.433 us latency 200000 us
From 16:22:2 20 Hz 2.050 ms to 17:22:4 length 35.033 us latency 200000 us
From 16:22:2 20 Hz 2.050 ms to 18:33:2 length 3.500 us latency 200000 us
From 16:22:2 20 Hz 2.050 ms to 19:28:1 length 5.033 us latency 200000 us
From 16:22:3 10 Hz 7.650 ms to 16:21:4 length 0.000 us latency 200000 us
From 16:22:3 10 Hz 7.650 ms to 18:29:10 length 33.833 us latency 200000 us
From 16:22:4 5 Hz 12.000 ms to 16:21:3 length 1.833 us latency 200000 us
From 16:22:4 5 Hz 12.000 ms to 16:21:5 length 0.000 us latency 200000 us
From 16:22:4 5 Hz 12.000 ms to 16:22:4 length 3.967 us latency 200000 us
From 16:22:4 5 Hz 12.000 ms to 17:21:3 length 1.833 us latency 200000 us
From 16:22:4 5 Hz 12.000 ms to 17:22:4 length 8.000 us latency 200000 us
From 16:22:4 5 Hz 12.000 ms to 19:28:1 length 6.933 us latency 200000 us
From 17:16:1 5 Hz 0.563 ms to 18:29:1 length 4.567 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 12:0:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 13:0:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 14:0:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 15:0:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 16:16:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 16:16:2 length 12.467 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 16:16:3 length 21.633 us latency 93918 us
From 17:16:2 5 Hz 1.126 ms to 16:19:5 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 16:21:5 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 16:22:4 length 6.633 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 17:19:5 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 17:21:5 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 17:22:4 length 6.633 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:16:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:16:2 length 12.467 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:16:3 length 21.633 us latency 93918 us
From 17:16:2 5 Hz 1.126 ms to 18:20:4 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:29:2 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:32:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:33:2 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 18:35:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 19:16:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 19:16:2 length 12.467 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 19:16:3 length 21.633 us latency 93918 us

From 17:16:2 5 Hz 1.126 ms to 19:20:4 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 19:26:22 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 19:27:1 length 0.833 us latency 200000 us
From 17:16:2 5 Hz 1.126 ms to 19:28:1 length 0.833 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 12:0:1 length 5.800 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 13:0:1 length 5.800 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 14:0:1 length 5.800 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 15:0:1 length 5.800 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 16:16:5 length 39.167 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 17:16:5 length 4.200 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 18:16:5 length 39.167 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 18:29:10 length 9.300 us latency 200000 us
From 17:16:5 5 Hz 3.041 ms to 19:16:5 length 39.167 us latency 200000 us
From 17:17:1 10 Hz 0.980 ms to 16:17:1 length 5.167 us latency 200000 us
From 17:17:1 10 Hz 0.980 ms to 18:17:1 length 5.167 us latency 200000 us
From 17:17:1 10 Hz 0.980 ms to 18:29:9 length 94.667 us latency 200000 us
From 17:17:1 10 Hz 0.980 ms to 19:17:1 length 5.167 us latency 200000 us
From 17:17:1 10 Hz 0.980 ms to 20:2:3 length 447.733 us latency 200000 us
From 17:17:1 10 Hz 0.980 ms to 22:2:3 length 447.733 us latency 200000 us
From 17:19:1 80 Hz 1.180 ms to 16:22:4 length 11.133 us latency 200000 us
From 17:19:1 80 Hz 1.180 ms to 17:22:4 length 11.133 us latency 200000 us
From 17:19:1 80 Hz 1.180 ms to 18:32:1 length 2.900 us latency 200000 us
From 17:19:4 40 Hz 2.000 ms to 16:22:4 length 13.500 us latency 200000 us
From 17:19:4 40 Hz 2.000 ms to 17:22:4 length 13.500 us latency 200000 us
From 17:19:5 5 Hz 0.100 ms to 16:19:5 length 2.967 us latency 200000 us
From 17:19:5 5 Hz 0.100 ms to 16:22:4 length 9.300 us latency 200000 us
From 17:19:5 5 Hz 0.100 ms to 17:22:4 length 9.300 us latency 200000 us
From 17:19:5 5 Hz 0.100 ms to 18:29:10 length 9.300 us latency 200000 us
From 17:21:2 40 Hz 3.150 ms to 17:22:2 length 0.000 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 16:21:3 length 4.033 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 16:22:2 length 7.933 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 16:22:3 length 5.033 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 17:22:2 length 11.967 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 17:22:3 length 5.033 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 18:29:10 length 5.033 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 18:32:1 length 7.933 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 19:26:22 length 14.800 us latency 150000 us
From 17:21:3 20 Hz 2.800 ms to 19:27:1 length 6.867 us latency 200000 us
From 17:21:3 20 Hz 2.800 ms to 19:28:1 length 6.867 us latency 200000 us
From 17:21:4 10 Hz 5.900 ms to 16:22:4 length 2.367 us latency 200000 us
From 17:21:4 10 Hz 5.900 ms to 17:22:3 length 0.000 us latency 200000 us
From 17:21:4 10 Hz 5.900 ms to 17:22:4 length 2.367 us latency 200000 us
From 17:21:5 5 Hz 14.600 ms to 16:21:5 length 35.800 us latency 200000 us
From 17:21:5 5 Hz 14.600 ms to 17:22:4 length 0.000 us latency 200000 us

From 17:21:5 5 Hz 14.600 ms to 18:29:10 length 8.300 us latency 200000 us
From 17:21:5 5 Hz 14.600 ms to 18:32:1 length 16.833 us latency 200000 us
From 17:21:5 5 Hz 14.600 ms to 19:26:22 length 3.967 us latency 200000 us
From 17:22:2 20 Hz 2.050 ms to 16:22:2 length 2.433 us latency 200000 us
From 17:22:2 20 Hz 2.050 ms to 16:22:4 length 35.033 us latency 200000 us
From 17:22:2 20 Hz 2.050 ms to 17:21:3 length 0.000 us latency 200000 us
From 17:22:2 20 Hz 2.050 ms to 18:33:2 length 3.500 us latency 200000 us
From 17:22:2 20 Hz 2.050 ms to 19:28:1 length 5.033 us latency 200000 us
From 17:22:3 10 Hz 7.650 ms to 17:21:4 length 0.000 us latency 200000 us
From 17:22:3 10 Hz 7.650 ms to 18:29:10 length 33.833 us latency 200000 us
From 17:22:4 5 Hz 12.000 ms to 16:21:3 length 1.833 us latency 200000 us
From 17:22:4 5 Hz 12.000 ms to 16:22:4 length 8.000 us latency 200000 us
From 17:22:4 5 Hz 12.000 ms to 17:21:3 length 1.833 us latency 200000 us
From 17:22:4 5 Hz 12.000 ms to 17:21:5 length 0.000 us latency 200000 us
From 17:22:4 5 Hz 12.000 ms to 17:22:4 length 3.967 us latency 200000 us
From 17:22:4 5 Hz 12.000 ms to 19:28:1 length 3.967 us latency 200000 us
From 18:16:1 5 Hz 0.563 ms to 18:29:1 length 0.000 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 12:0:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 13:0:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 14:0:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 15:0:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 16:16:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 16:16:2 length 12.467 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 16:16:3 length 21.633 us latency 93918 us
From 18:16:2 5 Hz 1.126 ms to 16:19:5 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 16:21:5 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 16:22:4 length 6.633 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 17:16:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 17:16:2 length 12.467 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 17:16:3 length 21.633 us latency 93918 us
From 18:16:2 5 Hz 1.126 ms to 17:19:5 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 17:21:5 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 17:22:4 length 6.633 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 18:20:4 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 18:29:2 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 18:31:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 18:32:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 18:33:2 length 1.667 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 18:35:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 19:16:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 19:16:2 length 12.467 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 19:16:3 length 21.633 us latency 93918 us
From 18:16:2 5 Hz 1.126 ms to 19:20:4 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 19:26:22 length 0.833 us latency 200000 us

From 18:16:2 5 Hz 1.126 ms to 19:27:1 length 0.833 us latency 200000 us
From 18:16:2 5 Hz 1.126 ms to 19:28:1 length 0.833 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 12:0:1 length 5.800 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 13:0:1 length 5.800 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 14:0:1 length 5.800 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 15:0:1 length 5.800 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 16:16:5 length 39.167 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 17:16:5 length 39.167 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 18:16:5 length 4.200 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 18:29:10 length 9.300 us latency 200000 us
From 18:16:5 5 Hz 3.041 ms to 19:16:5 length 39.167 us latency 200000 us
From 18:17:1 10 Hz 0.980 ms to 16:17:1 length 5.167 us latency 200000 us
From 18:17:1 10 Hz 0.980 ms to 17:17:1 length 5.167 us latency 200000 us
From 18:17:1 10 Hz 0.980 ms to 18:29:9 length 94.667 us latency 200000 us
From 18:17:1 10 Hz 0.980 ms to 19:17:1 length 5.167 us latency 200000 us
From 18:17:1 10 Hz 0.980 ms to 20:2:3 length 447.733 us latency 200000 us
From 18:17:1 10 Hz 0.980 ms to 22:2:3 length 447.733 us latency 200000 us
From 18:20:4 5 Hz 0.676 ms to 16:22:4 length 9.300 us latency 200000 us
From 18:20:4 5 Hz 0.676 ms to 17:22:4 length 9.300 us latency 200000 us
From 18:20:4 5 Hz 0.676 ms to 18:29:10 length 9.300 us latency 200000 us
From 18:20:4 5 Hz 0.676 ms to 19:20:4 length 2.967 us latency 200000 us
From 18:29:1 40 Hz 2.452 ms to 16:16:1 length 68.567 us latency 200000 us
From 18:29:1 40 Hz 2.452 ms to 17:16:1 length 68.567 us latency 200000 us
From 18:29:1 40 Hz 2.452 ms to 18:16:1 length 68.567 us latency 200000 us
From 18:29:1 40 Hz 2.452 ms to 18:35:4 length 68.567 us latency 200000 us
From 18:29:1 40 Hz 2.452 ms to 19:16:1 length 68.567 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 16:16:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 16:19:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 16:21:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 16:22:4 length 4.033 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 17:16:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 17:19:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 17:21:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 17:22:4 length 4.033 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:16:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:20:4 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:29:10 length 4.033 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:31:1 length 12.100 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:32:3 length 35.767 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:33:2 length 44.367 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:35:1 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 18:35:4 length 16.300 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 19:16:5 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 19:20:4 length 20.167 us latency 200000 us

From 18:29:10 10 Hz 16.650 ms to 19:26:22 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 19:27:1 length 20.167 us latency 200000 us
From 18:29:10 10 Hz 16.650 ms to 19:28:1 length 20.167 us latency 200000 us
From 18:29:2 5 Hz 1.243 ms to 18:29:10 length 8.300 us latency 200000 us
From 18:29:9 5 Hz 4.079 ms to 12:0:1 length 4.100 us latency 200000 us
From 18:29:9 5 Hz 4.079 ms to 13:0:1 length 4.100 us latency 200000 us
From 18:29:9 5 Hz 4.079 ms to 14:0:1 length 4.100 us latency 200000 us
From 18:29:9 5 Hz 4.079 ms to 15:0:1 length 4.100 us latency 200000 us
From 18:29:9 5 Hz 4.079 ms to 16:17:1 length 42.967 us latency 185741 us
From 18:29:9 5 Hz 4.079 ms to 17:17:1 length 42.967 us latency 185741 us
From 18:29:9 5 Hz 4.079 ms to 18:17:1 length 42.967 us latency 185741 us
From 18:29:9 5 Hz 4.079 ms to 19:17:1 length 42.967 us latency 185741 us
From 18:32:3 5 Hz 1.120 ms to 16:22:4 length 5.633 us latency 200000 us
From 18:32:3 5 Hz 1.120 ms to 17:22:4 length 5.633 us latency 200000 us
From 18:32:3 5 Hz 1.120 ms to 18:29:10 length 8.300 us latency 200000 us
From 18:33:2 20 Hz 1.942 ms to 16:22:2 length 67.500 us latency 200000 us
From 18:33:2 20 Hz 1.942 ms to 17:22:2 length 67.500 us latency 200000 us
From 18:33:2 20 Hz 1.942 ms to 18:29:10 length 0.000 us latency 200000 us
From 18:33:2 20 Hz 1.942 ms to 18:33:2 length 8.300 us latency 200000 us
From 18:33:2 20 Hz 1.942 ms to 18:35:2 length 16.600 us latency 200000 us
From 18:35:1 20 Hz 1.230 ms to 16:22:4 length 5.633 us latency 200000 us
From 18:35:1 20 Hz 1.230 ms to 17:22:4 length 5.633 us latency 200000 us
From 18:35:1 20 Hz 1.230 ms to 18:29:1 length 5.633 us latency 200000 us
From 18:35:1 20 Hz 1.230 ms to 18:29:10 length 8.300 us latency 200000 us
From 18:35:1 20 Hz 1.230 ms to 18:33:2 length 5.633 us latency 200000 us
From 18:35:1 20 Hz 1.230 ms to 19:26:22 length 5.633 us latency 200000 us
From 18:35:2 20 Hz 0.827 ms to 16:22:4 length 5.633 us latency 200000 us
From 18:35:2 20 Hz 0.827 ms to 17:22:4 length 5.633 us latency 200000 us
From 18:35:2 20 Hz 0.827 ms to 18:33:2 length 19.267 us latency 200000 us
From 18:35:2 20 Hz 0.827 ms to 19:26:22 length 5.633 us latency 200000 us
From 18:35:4 20 Hz 2.100 ms to 16:22:4 length 17.667 us latency 200000 us
From 18:35:4 20 Hz 2.100 ms to 17:22:4 length 17.667 us latency 200000 us
From 18:35:4 20 Hz 2.100 ms to 18:29:1 length 17.667 us latency 85000 us
From 18:35:4 20 Hz 2.100 ms to 18:29:10 length 16.300 us latency 200000 us
From 18:35:4 20 Hz 2.100 ms to 18:33:2 length 68.567 us latency 200000 us
From 18:35:4 20 Hz 2.100 ms to 19:26:22 length 68.567 us latency 200000 us
From 19:16:1 5 Hz 0.563 ms to 18:29:1 length 4.567 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 12:0:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 13:0:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 14:0:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 15:0:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 16:16:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 16:16:2 length 12.467 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 16:16:3 length 21.633 us latency 93918 us

From 19:16:2 5 Hz 1.126 ms to 16:19:5 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 16:21:5 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 16:22:4 length 6.633 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 17:16:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 17:16:2 length 12.467 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 17:16:3 length 21.633 us latency 93918 us
From 19:16:2 5 Hz 1.126 ms to 17:19:5 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 17:21:5 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 17:22:4 length 6.633 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:16:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:16:2 length 11.633 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:16:3 length 21.633 us latency 93918 us
From 19:16:2 5 Hz 1.126 ms to 18:20:4 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:29:2 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:32:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:33:2 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 18:35:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 19:16:2 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 19:20:4 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 19:26:22 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 19:27:1 length 0.833 us latency 200000 us
From 19:16:2 5 Hz 1.126 ms to 19:28:1 length 0.833 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 12:0:1 length 5.800 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 13:0:1 length 5.800 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 14:0:1 length 5.800 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 15:0:1 length 5.800 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 16:16:5 length 39.167 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 17:16:5 length 39.167 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 18:16:5 length 39.167 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 18:29:10 length 9.300 us latency 200000 us
From 19:16:5 5 Hz 3.041 ms to 19:16:5 length 4.200 us latency 200000 us
From 19:16:6 5 Hz 1.014 ms to 19:26:22 length 0.000 us latency 200000 us
From 19:16:6 5 Hz 1.014 ms to 19:28:1 length 0.000 us latency 200000 us
From 19:17:1 10 Hz 0.980 ms to 16:17:1 length 5.167 us latency 200000 us
From 19:17:1 10 Hz 0.980 ms to 17:17:1 length 5.167 us latency 200000 us
From 19:17:1 10 Hz 0.980 ms to 18:17:1 length 5.167 us latency 200000 us
From 19:17:1 10 Hz 0.980 ms to 18:29:9 length 94.667 us latency 200000 us
From 19:17:1 10 Hz 0.980 ms to 20:2:3 length 447.733 us latency 200000 us
From 19:17:1 10 Hz 0.980 ms to 22:2:3 length 447.733 us latency 200000 us
From 19:18:1 10 Hz 0.750 ms to 12:0:1 length 3.333 us latency 200000 us
From 19:18:1 10 Hz 0.750 ms to 13:0:1 length 3.333 us latency 200000 us
From 19:18:1 10 Hz 0.750 ms to 14:0:1 length 3.333 us latency 200000 us
From 19:18:1 10 Hz 0.750 ms to 15:0:1 length 3.333 us latency 200000 us
From 19:20:4 5 Hz 0.676 ms to 16:22:4 length 9.300 us latency 200000 us

From 19:20:4 5 Hz 0.676 ms to 17:22:4 length 9.300 us latency 200000 us
From 19:20:4 5 Hz 0.676 ms to 18:20:4 length 2.967 us latency 200000 us
From 19:20:4 5 Hz 0.676 ms to 18:29:10 length 9.300 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 16:21:3 length 2.433 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 16:21:4 length 4.033 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 16:21:5 length 13.700 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 16:22:2 length 140.167 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 16:22:3 length 4.033 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 16:22:4 length 13.700 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 17:21:3 length 2.433 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 17:21:4 length 4.033 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 17:21:5 length 13.700 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 17:22:2 length 140.167 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 17:22:3 length 4.033 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 17:22:4 length 13.700 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 18:29:10 length 13.933 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 18:32:1 length 10.500 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 18:33:2 length 24.733 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 18:35:4 length 39.767 us latency 200000 us
From 19:26:22 20 Hz 8.600 ms to 19:27:1 length 14.767 us latency 100000 us
From 19:26:22 20 Hz 8.600 ms to 19:28:1 length 10.500 us latency 100000 us
From 19:27:1 20 Hz 3.550 ms to 12:0:1 length 0.833 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 13:0:1 length 0.833 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 14:0:1 length 0.833 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 16:21:4 length 4.033 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 16:22:3 length 4.033 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 17:21:4 length 4.033 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 17:22:3 length 4.033 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 18:29:10 length 8.300 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 18:32:1 length 4.033 us latency 200000 us
From 19:27:1 20 Hz 3.550 ms to 19:26:22 length 4.033 us latency 100000 us
From 19:28:1 20 Hz 6.550 ms to 16:21:3 length 2.433 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 16:21:4 length 4.033 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 16:21:5 length 4.033 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 16:22:2 length 2.433 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 16:22:3 length 4.033 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 17:21:3 length 2.433 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 17:21:4 length 4.033 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 17:21:5 length 4.033 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 17:22:2 length 2.433 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 17:22:3 length 4.033 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 18:29:10 length 8.300 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 18:32:1 length 8.067 us latency 200000 us
From 19:28:1 20 Hz 6.550 ms to 19:26:22 length 10.500 us latency 100000 us

From 1:0:1 5 Hz 0.000 ms to 11:2:3 length 5.633 us latency 200000 us
From 1:0:1 5 Hz 0.000 ms to 5:16:3 length 16.300 us latency 68993 us
From 1:0:1 5 Hz 0.000 ms to 5:16:5 length 36.333 us latency 100000 us
From 1:0:1 5 Hz 0.000 ms to 6:16:3 length 16.300 us latency 68993 us
From 1:0:1 5 Hz 0.000 ms to 6:16:5 length 36.333 us latency 100000 us
From 1:0:1 5 Hz 0.000 ms to 7:16:3 length 16.300 us latency 68993 us
From 1:0:1 5 Hz 0.000 ms to 7:16:5 length 36.333 us latency 100000 us
From 1:0:1 5 Hz 0.000 ms to 7:29:9 length 383.333 us latency 200000 us
From 1:0:1 5 Hz 0.000 ms to 8:16:3 length 16.300 us latency 68993 us
From 1:0:1 5 Hz 0.000 ms to 8:16:5 length 36.333 us latency 100000 us
From 1:0:1 5 Hz 0.000 ms to 8:30:20 length 51.267 us latency 200000 us
From 1:0:1 5 Hz 0.000 ms to 9:2:3 length 5.633 us latency 200000 us
From 1:0:2 10 Hz 0.000 ms to 7:29:1 length 1.667 us latency 200000 us
From 1:0:3 40 Hz 0.000 ms to 7:29:1 length 0.833 us latency 200000 us
From 1:0:3 40 Hz 0.000 ms to 8:26:22 length 1.667 us latency 50000 us
From 20:2:3 5 Hz 0.600 ms to 12:0:1 length 11.967 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 13:0:1 length 11.967 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 14:0:1 length 11.967 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 15:0:1 length 11.967 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 16:17:1 length 49.000 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 17:17:1 length 49.000 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 18:17:1 length 49.000 us latency 200000 us
From 20:2:3 5 Hz 0.600 ms to 19:17:1 length 49.000 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 12:0:1 length 11.967 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 13:0:1 length 11.967 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 14:0:1 length 11.967 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 15:0:1 length 11.967 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 16:17:1 length 49.000 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 17:17:1 length 49.000 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 18:17:1 length 49.000 us latency 200000 us
From 22:2:3 5 Hz 0.600 ms to 19:17:1 length 49.000 us latency 200000 us
From 2:0:1 5 Hz 0.000 ms to 11:2:3 length 5.633 us latency 200000 us
From 2:0:1 5 Hz 0.000 ms to 5:16:3 length 16.300 us latency 68993 us
From 2:0:1 5 Hz 0.000 ms to 5:16:5 length 36.333 us latency 100000 us
From 2:0:1 5 Hz 0.000 ms to 6:16:3 length 16.300 us latency 68993 us
From 2:0:1 5 Hz 0.000 ms to 6:16:5 length 36.333 us latency 100000 us
From 2:0:1 5 Hz 0.000 ms to 7:16:3 length 16.300 us latency 68993 us
From 2:0:1 5 Hz 0.000 ms to 7:16:5 length 36.333 us latency 100000 us
From 2:0:1 5 Hz 0.000 ms to 7:29:9 length 383.333 us latency 200000 us
From 2:0:1 5 Hz 0.000 ms to 8:16:3 length 16.300 us latency 68993 us
From 2:0:1 5 Hz 0.000 ms to 8:16:5 length 36.333 us latency 100000 us
From 2:0:1 5 Hz 0.000 ms to 8:30:20 length 51.267 us latency 200000 us
From 2:0:1 5 Hz 0.000 ms to 9:2:3 length 5.633 us latency 200000 us
From 2:0:2 10 Hz 0.000 ms to 7:29:1 length 2.500 us latency 200000 us

From 2:0:3 40 Hz 0.000 ms to 7:29:1 length 0.833 us latency 200000 us
From 3:0:1 5 Hz 0.000 ms to 11:2:3 length 5.633 us latency 200000 us
From 3:0:1 5 Hz 0.000 ms to 5:16:3 length 16.300 us latency 68993 us
From 3:0:1 5 Hz 0.000 ms to 5:16:5 length 36.333 us latency 100000 us
From 3:0:1 5 Hz 0.000 ms to 6:16:3 length 16.300 us latency 68993 us
From 3:0:1 5 Hz 0.000 ms to 6:16:5 length 36.333 us latency 100000 us
From 3:0:1 5 Hz 0.000 ms to 7:16:3 length 16.300 us latency 68993 us
From 3:0:1 5 Hz 0.000 ms to 7:16:5 length 36.333 us latency 100000 us
From 3:0:1 5 Hz 0.000 ms to 7:29:9 length 383.333 us latency 200000 us
From 3:0:1 5 Hz 0.000 ms to 8:16:3 length 16.300 us latency 68993 us
From 3:0:1 5 Hz 0.000 ms to 8:16:5 length 36.333 us latency 100000 us
From 3:0:1 5 Hz 0.000 ms to 8:30:20 length 51.267 us latency 200000 us
From 3:0:1 5 Hz 0.000 ms to 9:2:3 length 5.633 us latency 200000 us
From 3:0:2 10 Hz 0.000 ms to 7:29:1 length 1.667 us latency 200000 us
From 3:0:3 40 Hz 0.000 ms to 7:29:1 length 0.833 us latency 200000 us
From 4:0:1 5 Hz 0.000 ms to 11:2:3 length 5.633 us latency 200000 us
From 4:0:1 5 Hz 0.000 ms to 5:16:3 length 16.300 us latency 68993 us
From 4:0:1 5 Hz 0.000 ms to 5:16:5 length 36.333 us latency 100000 us
From 4:0:1 5 Hz 0.000 ms to 6:16:3 length 16.300 us latency 68993 us
From 4:0:1 5 Hz 0.000 ms to 6:16:5 length 36.333 us latency 100000 us
From 4:0:1 5 Hz 0.000 ms to 7:16:3 length 16.300 us latency 68993 us
From 4:0:1 5 Hz 0.000 ms to 7:16:5 length 36.333 us latency 100000 us
From 4:0:1 5 Hz 0.000 ms to 7:29:9 length 383.333 us latency 200000 us
From 4:0:1 5 Hz 0.000 ms to 8:16:3 length 16.300 us latency 68993 us
From 4:0:1 5 Hz 0.000 ms to 8:16:5 length 36.333 us latency 100000 us
From 4:0:1 5 Hz 0.000 ms to 8:30:20 length 51.267 us latency 200000 us
From 4:0:1 5 Hz 0.000 ms to 9:2:3 length 5.633 us latency 200000 us
From 4:0:2 10 Hz 0.000 ms to 7:29:1 length 1.667 us latency 200000 us
From 4:0:3 40 Hz 0.000 ms to 7:29:1 length 0.833 us latency 200000 us
From 4:0:3 40 Hz 0.000 ms to 8:26:22 length 1.667 us latency 50000 us
From 5:16:1 5 Hz 0.563 ms to 7:29:1 length 4.567 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 1:0:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 2:0:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 3:0:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 4:0:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 5:19:5 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 5:21:5 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 5:22:4 length 6.633 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 6:16:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 6:16:2 length 12.467 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 6:16:3 length 21.633 us latency 93918 us
From 5:16:2 5 Hz 1.126 ms to 6:19:5 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 6:21:5 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 6:22:4 length 6.633 us latency 200000 us

From 5:16:2 5 Hz 1.126 ms to 7:16:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:16:2 length 12.467 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:16:3 length 21.633 us latency 93918 us
From 5:16:2 5 Hz 1.126 ms to 7:20:4 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:29:2 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:31:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:32:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:33:2 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 7:35:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:16:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:16:2 length 12.467 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:16:3 length 21.633 us latency 93918 us
From 5:16:2 5 Hz 1.126 ms to 8:20:4 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:26:22 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:27:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:28:1 length 0.833 us latency 200000 us
From 5:16:2 5 Hz 1.126 ms to 8:30:20 length 21.000 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 1:0:1 length 5.800 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 2:0:1 length 5.800 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 3:0:1 length 5.800 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 4:0:1 length 5.800 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 5:16:5 length 4.200 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 6:16:5 length 39.167 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 7:16:5 length 39.167 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 7:29:10 length 9.300 us latency 200000 us
From 5:16:5 5 Hz 3.041 ms to 8:16:5 length 39.167 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 11:2:3 length 447.733 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 6:17:1 length 5.167 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 7:17:1 length 5.167 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 7:29:9 length 94.667 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 8:17:1 length 5.167 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 8:30:20 length 353.067 us latency 200000 us
From 5:17:1 10 Hz 0.980 ms to 9:2:3 length 447.733 us latency 200000 us
From 5:19:1 80 Hz 1.180 ms to 5:22:4 length 11.133 us latency 200000 us
From 5:19:1 80 Hz 1.180 ms to 6:22:4 length 11.133 us latency 200000 us
From 5:19:1 80 Hz 1.180 ms to 7:32:1 length 2.900 us latency 200000 us
From 5:19:1 80 Hz 1.180 ms to 8:30:20 length 21.300 us latency 200000 us
From 5:19:2 10 Hz 2.400 ms to 8:30:20 length 43.133 us latency 200000 us
From 5:19:3 20 Hz 1.900 ms to 8:30:20 length 54.100 us latency 200000 us
From 5:19:4 40 Hz 2.000 ms to 5:22:4 length 13.500 us latency 200000 us
From 5:19:4 40 Hz 2.000 ms to 6:22:4 length 13.500 us latency 200000 us
From 5:19:4 40 Hz 2.000 ms to 8:30:20 length 41.433 us latency 200000 us
From 5:19:5 5 Hz 0.100 ms to 5:22:4 length 9.300 us latency 200000 us
From 5:19:5 5 Hz 0.100 ms to 6:19:5 length 2.967 us latency 200000 us

From 5:19:5 5 Hz 0.100 ms to 6:22:4 length 9.300 us latency 200000 us
From 5:19:5 5 Hz 0.100 ms to 7:29:10 length 9.300 us latency 200000 us
From 5:19:5 5 Hz 0.100 ms to 8:30:20 length 11.133 us latency 200000 us
From 5:21:2 40 Hz 3.150 ms to 5:22:2 length 0.000 us latency 200000 us
From 5:21:2 40 Hz 3.150 ms to 8:30:20 length 1.833 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 5:22:2 length 11.967 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 5:22:3 length 5.033 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 6:21:3 length 4.033 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 6:22:2 length 7.933 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 6:22:3 length 5.033 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 7:29:10 length 5.033 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 7:32:1 length 7.933 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 8:26:22 length 14.800 us latency 150000 us
From 5:21:3 20 Hz 2.800 ms to 8:27:1 length 6.867 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 8:28:1 length 6.867 us latency 200000 us
From 5:21:3 20 Hz 2.800 ms to 8:30:20 length 36.100 us latency 200000 us
From 5:21:4 10 Hz 5.900 ms to 5:22:3 length 0.000 us latency 200000 us
From 5:21:4 10 Hz 5.900 ms to 5:22:4 length 2.367 us latency 200000 us
From 5:21:4 10 Hz 5.900 ms to 6:22:4 length 2.367 us latency 200000 us
From 5:21:4 10 Hz 5.900 ms to 8:30:20 length 18.933 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 5:22:4 length 0.000 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 6:21:5 length 35.800 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 7:29:10 length 8.300 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 7:32:1 length 16.833 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 7:33:2 length 16.833 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 8:26:22 length 3.967 us latency 200000 us
From 5:21:5 5 Hz 14.600 ms to 8:30:20 length 1.833 us latency 200000 us
From 5:22:2 20 Hz 2.050 ms to 5:21:3 length 0.000 us latency 200000 us
From 5:22:2 20 Hz 2.050 ms to 6:22:2 length 2.433 us latency 200000 us
From 5:22:2 20 Hz 2.050 ms to 6:22:4 length 35.033 us latency 200000 us
From 5:22:2 20 Hz 2.050 ms to 7:33:2 length 3.500 us latency 200000 us
From 5:22:2 20 Hz 2.050 ms to 8:28:1 length 5.033 us latency 200000 us
From 5:22:2 20 Hz 2.050 ms to 8:30:20 length 4.733 us latency 200000 us
From 5:22:3 10 Hz 7.650 ms to 5:21:4 length 0.000 us latency 200000 us
From 5:22:3 10 Hz 7.650 ms to 7:29:10 length 33.833 us latency 200000 us
From 5:22:3 10 Hz 7.650 ms to 8:30:20 length 6.567 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 5:21:3 length 1.833 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 5:21:5 length 0.000 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 5:22:4 length 3.967 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 6:21:3 length 1.833 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 6:22:4 length 8.000 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 8:28:1 length 6.933 us latency 200000 us
From 5:22:4 5 Hz 12.000 ms to 8:30:20 length 1.833 us latency 200000 us
From 6:16:1 5 Hz 0.563 ms to 7:29:1 length 4.567 us latency 200000 us

From 6:16:2 5 Hz 1.126 ms to 1:0:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 2:0:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 3:0:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 4:0:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 5:16:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 5:16:2 length 12.467 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 5:16:3 length 21.633 us latency 93918 us
From 6:16:2 5 Hz 1.126 ms to 5:19:5 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 5:21:5 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 5:22:4 length 6.633 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 6:19:5 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 6:21:5 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 6:22:4 length 6.633 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:16:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:16:2 length 12.467 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:16:3 length 21.633 us latency 93918 us
From 6:16:2 5 Hz 1.126 ms to 7:20:4 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:29:2 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:31:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:32:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:33:2 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 7:35:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:16:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:16:2 length 12.467 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:16:3 length 21.633 us latency 93918 us
From 6:16:2 5 Hz 1.126 ms to 8:20:4 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:26:22 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:27:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:28:1 length 0.833 us latency 200000 us
From 6:16:2 5 Hz 1.126 ms to 8:30:20 length 21.000 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 1:0:1 length 5.800 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 2:0:1 length 5.800 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 3:0:1 length 5.800 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 4:0:1 length 5.800 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 5:16:5 length 39.167 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 6:16:5 length 4.200 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 7:16:5 length 39.167 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 7:29:10 length 9.300 us latency 200000 us
From 6:16:5 5 Hz 3.041 ms to 8:16:5 length 39.167 us latency 200000 us
From 6:17:1 10 Hz 0.980 ms to 11:2:3 length 447.733 us latency 200000 us
From 6:17:1 10 Hz 0.980 ms to 5:17:1 length 5.167 us latency 200000 us
From 6:17:1 10 Hz 0.980 ms to 7:17:1 length 5.167 us latency 200000 us
From 6:17:1 10 Hz 0.980 ms to 7:29:9 length 94.667 us latency 200000 us
From 6:17:1 10 Hz 0.980 ms to 8:17:1 length 5.167 us latency 200000 us

From 6:17:1 10 Hz 0.980 ms to 8:30:20 length 353.067 us latency 200000 us
From 6:17:1 10 Hz 0.980 ms to 9:2:3 length 447.733 us latency 200000 us
From 6:19:1 80 Hz 1.180 ms to 5:22:4 length 11.133 us latency 200000 us
From 6:19:1 80 Hz 1.180 ms to 6:22:4 length 11.133 us latency 200000 us
From 6:19:1 80 Hz 1.180 ms to 7:32:1 length 2.900 us latency 200000 us
From 6:19:1 80 Hz 1.180 ms to 8:30:20 length 21.300 us latency 200000 us
From 6:19:2 10 Hz 2.400 ms to 8:30:20 length 43.133 us latency 200000 us
From 6:19:3 20 Hz 1.900 ms to 8:30:20 length 54.100 us latency 200000 us
From 6:19:4 40 Hz 2.000 ms to 5:22:4 length 13.500 us latency 200000 us
From 6:19:4 40 Hz 2.000 ms to 6:22:4 length 13.500 us latency 200000 us
From 6:19:4 40 Hz 2.000 ms to 8:30:20 length 41.433 us latency 200000 us
From 6:19:5 5 Hz 0.100 ms to 5:19:5 length 2.967 us latency 200000 us
From 6:19:5 5 Hz 0.100 ms to 5:22:4 length 9.300 us latency 200000 us
From 6:19:5 5 Hz 0.100 ms to 6:22:4 length 9.300 us latency 200000 us
From 6:19:5 5 Hz 0.100 ms to 7:29:10 length 9.300 us latency 200000 us
From 6:19:5 5 Hz 0.100 ms to 8:30:20 length 11.133 us latency 200000 us
From 6:21:2 40 Hz 3.150 ms to 6:22:2 length 0.000 us latency 200000 us
From 6:21:2 40 Hz 3.150 ms to 8:30:20 length 1.833 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 5:21:3 length 4.033 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 5:22:2 length 7.933 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 5:22:3 length 5.033 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 6:22:2 length 11.967 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 6:22:3 length 5.033 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 7:29:10 length 5.033 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 7:32:1 length 7.933 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 8:26:22 length 14.800 us latency 150000 us
From 6:21:3 20 Hz 2.800 ms to 8:27:1 length 6.867 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 8:28:1 length 6.867 us latency 200000 us
From 6:21:3 20 Hz 2.800 ms to 8:30:20 length 36.100 us latency 200000 us
From 6:21:4 10 Hz 5.900 ms to 5:22:4 length 2.367 us latency 200000 us
From 6:21:4 10 Hz 5.900 ms to 6:22:3 length 0.000 us latency 200000 us
From 6:21:4 10 Hz 5.900 ms to 6:22:4 length 2.367 us latency 200000 us
From 6:21:4 10 Hz 5.900 ms to 8:30:20 length 18.933 us latency 200000 us
From 6:21:5 5 Hz 14.600 ms to 5:21:5 length 35.800 us latency 200000 us
From 6:21:5 5 Hz 14.600 ms to 6:22:4 length 0.000 us latency 200000 us
From 6:21:5 5 Hz 14.600 ms to 7:29:10 length 8.300 us latency 200000 us
From 6:21:5 5 Hz 14.600 ms to 7:32:1 length 16.833 us latency 200000 us
From 6:21:5 5 Hz 14.600 ms to 8:26:22 length 3.967 us latency 200000 us
From 6:21:5 5 Hz 14.600 ms to 8:30:20 length 1.833 us latency 200000 us
From 6:22:2 20 Hz 2.050 ms to 5:22:2 length 2.433 us latency 200000 us
From 6:22:2 20 Hz 2.050 ms to 5:22:4 length 35.033 us latency 200000 us
From 6:22:2 20 Hz 2.050 ms to 6:21:3 length 0.000 us latency 200000 us
From 6:22:2 20 Hz 2.050 ms to 7:33:2 length 3.500 us latency 200000 us
From 6:22:2 20 Hz 2.050 ms to 8:28:1 length 5.033 us latency 200000 us

From 6:22:2 20 Hz 2.050 ms to 8:30:20 length 4.733 us latency 200000 us
From 6:22:3 10 Hz 7.650 ms to 6:21:4 length 0.000 us latency 200000 us
From 6:22:3 10 Hz 7.650 ms to 7:29:10 length 33.833 us latency 200000 us
From 6:22:3 10 Hz 7.650 ms to 8:30:20 length 6.567 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 5:21:3 length 1.833 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 5:22:4 length 8.000 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 6:21:3 length 1.833 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 6:21:5 length 0.000 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 6:22:4 length 3.967 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 8:28:1 length 3.967 us latency 200000 us
From 6:22:4 5 Hz 12.000 ms to 8:30:20 length 1.833 us latency 200000 us
From 7:16:1 5 Hz 0.563 ms to 7:29:1 length 0.000 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 1:0:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 2:0:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 3:0:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 4:0:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 5:16:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 5:16:2 length 12.467 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 5:16:3 length 21.633 us latency 93918 us
From 7:16:2 5 Hz 1.126 ms to 5:19:5 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 5:21:5 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 5:22:4 length 6.633 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 6:16:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 6:16:2 length 12.467 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 6:16:3 length 21.633 us latency 93918 us
From 7:16:2 5 Hz 1.126 ms to 6:19:5 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 6:21:5 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 6:22:4 length 6.633 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 7:20:4 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 7:29:2 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 7:31:1 length 1.667 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 7:32:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 7:33:2 length 1.667 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 7:35:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:16:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:16:2 length 12.467 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:16:3 length 21.633 us latency 93918 us
From 7:16:2 5 Hz 1.126 ms to 8:20:4 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:26:22 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:27:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:28:1 length 0.833 us latency 200000 us
From 7:16:2 5 Hz 1.126 ms to 8:30:20 length 21.000 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 1:0:1 length 5.800 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 2:0:1 length 5.800 us latency 200000 us

From 7:16:5 5 Hz 3.041 ms to 3:0:1 length 5.800 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 4:0:1 length 5.800 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 5:16:5 length 39.167 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 6:16:5 length 39.167 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 7:16:5 length 4.200 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 7:29:10 length 9.300 us latency 200000 us
From 7:16:5 5 Hz 3.041 ms to 8:16:5 length 39.167 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 11:2:3 length 447.733 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 5:17:1 length 5.167 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 6:17:1 length 5.167 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 7:29:9 length 94.667 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 8:17:1 length 5.167 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 8:30:20 length 353.067 us latency 200000 us
From 7:17:1 10 Hz 0.980 ms to 9:2:3 length 447.733 us latency 200000 us
From 7:20:1 40 Hz 0.850 ms to 8:30:20 length 6.867 us latency 200000 us
From 7:20:2 10 Hz 0.800 ms to 8:30:20 length 47.933 us latency 200000 us
From 7:20:3 20 Hz 1.150 ms to 8:30:20 length 23.800 us latency 200000 us
From 7:20:4 5 Hz 0.676 ms to 5:22:4 length 9.300 us latency 200000 us
From 7:20:4 5 Hz 0.676 ms to 6:22:4 length 9.300 us latency 200000 us
From 7:20:4 5 Hz 0.676 ms to 7:29:10 length 9.300 us latency 200000 us
From 7:20:4 5 Hz 0.676 ms to 8:20:4 length 2.967 us latency 200000 us
From 7:20:4 5 Hz 0.676 ms to 8:30:20 length 32.033 us latency 200000 us
From 7:29:1 40 Hz 2.452 ms to 5:16:1 length 68.567 us latency 200000 us
From 7:29:1 40 Hz 2.452 ms to 6:16:1 length 68.567 us latency 200000 us
From 7:29:1 40 Hz 2.452 ms to 7:16:1 length 68.567 us latency 200000 us
From 7:29:1 40 Hz 2.452 ms to 7:35:4 length 68.567 us latency 200000 us
From 7:29:1 40 Hz 2.452 ms to 8:16:1 length 68.567 us latency 200000 us
From 7:29:1 40 Hz 2.452 ms to 8:30:20 length 14.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 1:0:1 length 0.833 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 2:0:1 length 0.833 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 3:0:1 length 0.833 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 5:16:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 5:19:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 5:21:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 5:22:4 length 4.033 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 6:16:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 6:19:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 6:21:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 6:22:4 length 4.033 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:16:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:20:4 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:29:10 length 4.033 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:31:1 length 12.100 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:32:3 length 32.267 us latency 200000 us

From 7:29:10 10 Hz 16.650 ms to 7:33:2 length 44.367 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:35:1 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 7:35:4 length 16.300 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 8:16:5 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 8:20:4 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 8:26:22 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 8:27:1 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 8:28:1 length 20.167 us latency 200000 us
From 7:29:10 10 Hz 16.650 ms to 8:30:20 length 63.767 us latency 200000 us
From 7:29:2 5 Hz 1.243 ms to 7:29:10 length 8.300 us latency 200000 us
From 7:29:9 5 Hz 4.079 ms to 1:0:1 length 4.100 us latency 200000 us
From 7:29:9 5 Hz 4.079 ms to 2:0:1 length 4.100 us latency 200000 us
From 7:29:9 5 Hz 4.079 ms to 3:0:1 length 4.100 us latency 200000 us
From 7:29:9 5 Hz 4.079 ms to 4:0:1 length 4.100 us latency 200000 us
From 7:29:9 5 Hz 4.079 ms to 5:17:1 length 42.967 us latency 185741 us
From 7:29:9 5 Hz 4.079 ms to 6:17:1 length 42.967 us latency 185741 us
From 7:29:9 5 Hz 4.079 ms to 7:17:1 length 42.967 us latency 185741 us
From 7:29:9 5 Hz 4.079 ms to 8:17:1 length 42.967 us latency 185741 us
From 7:31:1 10 Hz 1.550 ms to 8:30:20 length 6.767 us latency 200000 us
From 7:32:3 5 Hz 1.120 ms to 5:22:4 length 5.633 us latency 200000 us
From 7:32:3 5 Hz 1.120 ms to 6:22:4 length 5.633 us latency 200000 us
From 7:32:3 5 Hz 1.120 ms to 7:29:10 length 8.300 us latency 200000 us
From 7:32:3 5 Hz 1.120 ms to 8:30:20 length 9.133 us latency 200000 us
From 7:33:2 20 Hz 1.942 ms to 5:22:2 length 67.500 us latency 200000 us
From 7:33:2 20 Hz 1.942 ms to 6:22:2 length 67.500 us latency 200000 us
From 7:33:2 20 Hz 1.942 ms to 7:29:10 length 0.000 us latency 200000 us
From 7:33:2 20 Hz 1.942 ms to 7:33:2 length 8.300 us latency 200000 us
From 7:33:2 20 Hz 1.942 ms to 7:35:2 length 16.600 us latency 200000 us
From 7:33:2 20 Hz 1.942 ms to 8:30:20 length 3.500 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 5:22:4 length 5.633 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 6:22:4 length 5.633 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 7:29:1 length 5.633 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 7:29:10 length 8.300 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 7:33:2 length 5.633 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 8:26:22 length 5.633 us latency 200000 us
From 7:35:1 20 Hz 1.230 ms to 8:30:20 length 8.133 us latency 200000 us
From 7:35:2 20 Hz 0.827 ms to 5:22:4 length 5.633 us latency 200000 us
From 7:35:2 20 Hz 0.827 ms to 6:22:4 length 5.633 us latency 200000 us
From 7:35:2 20 Hz 0.827 ms to 7:33:2 length 19.267 us latency 200000 us
From 7:35:2 20 Hz 0.827 ms to 8:26:22 length 5.633 us latency 200000 us
From 7:35:2 20 Hz 0.827 ms to 8:30:20 length 28.433 us latency 200000 us
From 7:35:4 20 Hz 2.100 ms to 5:22:4 length 17.667 us latency 200000 us
From 7:35:4 20 Hz 2.100 ms to 6:22:4 length 17.667 us latency 200000 us
From 7:35:4 20 Hz 2.100 ms to 7:29:1 length 17.667 us latency 85000 us

From 7:35:4 20 Hz 2.100 ms to 7:29:10 length 16.300 us latency 200000 us
From 7:35:4 20 Hz 2.100 ms to 7:33:2 length 68.567 us latency 200000 us
From 7:35:4 20 Hz 2.100 ms to 8:26:22 length 68.567 us latency 200000 us
From 7:35:4 20 Hz 2.100 ms to 8:30:20 length 35.333 us latency 200000 us
From 8:16:1 5 Hz 0.563 ms to 7:29:1 length 4.567 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 1:0:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 2:0:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 3:0:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 4:0:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 5:16:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 5:16:2 length 12.467 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 5:16:3 length 21.633 us latency 93918 us
From 8:16:2 5 Hz 1.126 ms to 5:19:5 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 5:21:5 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 5:22:4 length 6.633 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 6:16:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 6:16:2 length 12.467 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 6:16:3 length 21.633 us latency 93918 us
From 8:16:2 5 Hz 1.126 ms to 6:19:5 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 6:21:5 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 6:22:4 length 6.633 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:16:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:16:2 length 11.633 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:16:3 length 21.633 us latency 93918 us
From 8:16:2 5 Hz 1.126 ms to 7:20:4 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:29:2 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:31:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:32:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:33:2 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 7:35:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 8:16:2 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 8:20:4 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 8:26:22 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 8:27:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 8:28:1 length 0.833 us latency 200000 us
From 8:16:2 5 Hz 1.126 ms to 8:30:20 length 22.367 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 1:0:1 length 5.800 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 2:0:1 length 5.800 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 3:0:1 length 5.800 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 4:0:1 length 5.800 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 5:16:5 length 39.167 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 6:16:5 length 39.167 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 7:16:5 length 39.167 us latency 200000 us
From 8:16:5 5 Hz 3.041 ms to 7:29:10 length 9.300 us latency 200000 us

From 8:16:5 5 Hz 3.041 ms to 8:16:5 length 4.200 us latency 200000 us
From 8:16:6 5 Hz 1.014 ms to 8:26:22 length 0.000 us latency 200000 us
From 8:16:6 5 Hz 1.014 ms to 8:28:1 length 0.000 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 11:2:3 length 447.733 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 5:17:1 length 5.167 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 6:17:1 length 5.167 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 7:17:1 length 5.167 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 7:29:9 length 94.667 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 8:30:20 length 353.067 us latency 200000 us
From 8:17:1 10 Hz 0.980 ms to 9:2:3 length 447.733 us latency 200000 us
From 8:18:1 10 Hz 0.750 ms to 1:0:1 length 3.333 us latency 200000 us
From 8:18:1 10 Hz 0.750 ms to 2:0:1 length 3.333 us latency 200000 us
From 8:18:1 10 Hz 0.750 ms to 3:0:1 length 3.333 us latency 200000 us
From 8:18:1 10 Hz 0.750 ms to 4:0:1 length 3.333 us latency 200000 us
From 8:20:1 40 Hz 0.850 ms to 8:30:20 length 6.867 us latency 200000 us
From 8:20:2 10 Hz 0.800 ms to 8:30:20 length 47.933 us latency 200000 us
From 8:20:3 20 Hz 1.150 ms to 8:30:20 length 23.800 us latency 200000 us
From 8:20:4 5 Hz 0.676 ms to 5:22:4 length 9.300 us latency 200000 us
From 8:20:4 5 Hz 0.676 ms to 6:22:4 length 9.300 us latency 200000 us
From 8:20:4 5 Hz 0.676 ms to 7:20:4 length 2.967 us latency 200000 us
From 8:20:4 5 Hz 0.676 ms to 7:29:10 length 9.300 us latency 200000 us
From 8:20:4 5 Hz 0.676 ms to 8:30:20 length 32.033 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 5:21:3 length 2.433 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 5:21:4 length 4.033 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 5:21:5 length 13.700 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 5:22:2 length 140.167 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 5:22:3 length 4.033 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 5:22:4 length 13.700 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 6:21:3 length 2.433 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 6:21:4 length 4.033 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 6:21:5 length 13.700 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 6:22:2 length 140.167 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 6:22:3 length 4.033 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 6:22:4 length 13.700 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 7:29:10 length 13.933 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 7:32:1 length 10.500 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 7:33:2 length 24.733 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 7:35:4 length 39.767 us latency 200000 us
From 8:26:22 20 Hz 8.600 ms to 8:27:1 length 14.767 us latency 100000 us
From 8:26:22 20 Hz 8.600 ms to 8:28:1 length 10.500 us latency 100000 us
From 8:26:22 20 Hz 8.600 ms to 8:30:20 length 62.700 us latency 200000 us
From 8:27:1 20 Hz 3.550 ms to 5:21:4 length 4.033 us latency 200000 us
From 8:27:1 20 Hz 3.550 ms to 5:22:3 length 4.033 us latency 200000 us
From 8:27:1 20 Hz 3.550 ms to 6:21:4 length 4.033 us latency 200000 us

From 8:27:1 20 Hz 3.550 ms to 6:22:3 length 4.033 us latency 200000 us
From 8:27:1 20 Hz 3.550 ms to 7:29:10 length 8.300 us latency 200000 us
From 8:27:1 20 Hz 3.550 ms to 7:32:1 length 4.033 us latency 200000 us
From 8:27:1 20 Hz 3.550 ms to 8:26:22 length 4.033 us latency 100000 us
From 8:27:1 20 Hz 3.550 ms to 8:30:20 length 8.067 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 5:21:3 length 2.433 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 5:21:4 length 4.033 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 5:21:5 length 4.033 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 5:22:2 length 2.433 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 5:22:3 length 4.033 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 6:21:3 length 2.433 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 6:21:4 length 4.033 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 6:21:5 length 4.033 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 6:22:2 length 2.433 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 6:22:3 length 4.033 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 7:29:10 length 8.300 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 7:32:1 length 8.067 us latency 200000 us
From 8:28:1 20 Hz 6.550 ms to 8:26:22 length 10.500 us latency 100000 us
From 8:28:1 20 Hz 6.550 ms to 8:30:20 length 25.333 us latency 200000 us
From 8:30:20 20 Hz 9.155 ms to 7:29:10 length 8.300 us latency 200000 us
From 8:30:20 20 Hz 9.155 ms to 7:31:1 length 169.200 us latency 200000 us
From 8:30:20 20 Hz 9.155 ms to 7:35:4 length 293.700 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 1:0:1 length 11.967 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 2:0:1 length 11.967 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 3:0:1 length 11.967 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 4:0:1 length 11.967 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 5:17:1 length 49.000 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 6:17:1 length 49.000 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 7:17:1 length 49.000 us latency 200000 us
From 9:2:3 5 Hz 0.600 ms to 8:17:1 length 49.000 us latency 200000 us

Appendix B

The Allocation and Scheduling Results of the Problem given in Appendix A:

The schedule for processor 0 is:

13: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
12: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
4: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
14: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
18:35: 1 starts at 0 ms 0 us and finishs at 1 ms 231 us
7:17: 1 starts at 1 ms 231 us and finishs at 2 ms 212 us
18:35: 4 starts at 2 ms 455 us and finishs at 4 ms 555 us
6:22: 2 starts at 5 ms 151 us and finishs at 7 ms 201 us
19:26:22 starts at 7 ms 201 us and finishs at 15 ms 802 us
6:19: 2 starts at 15 ms 802 us and finishs at 18 ms 203 us
5:17: 1 starts at 18 ms 203 us and finishs at 19 ms 184 us
5:16: 1 starts at 19 ms 184 us and finishs at 19 ms 748 us
19:28: 1 starts at 19 ms 795 us and finishs at 26 ms 346 us
6:21: 3 starts at 26 ms 346 us and finishs at 29 ms 146 us
18:17: 1 starts at 29 ms 146 us and finishs at 30 ms 127 us
7:32: 1 starts at 37 ms 778 us and finishs at 37 ms 778 us
9: 2: 3 starts at 37 ms 778 us and finishs at 38 ms 379 us
6:16: 1 starts at 38 ms 379 us and finishs at 38 ms 943 us
8:16: 2 starts at 38 ms 943 us and finishs at 40 ms 70 us
7:16: 2 starts at 40 ms 70 us and finishs at 41 ms 197 us
18:35: 1 starts at 49 ms 875 us and finishs at 51 ms 106 us
18:35: 4 starts at 52 ms 330 us and finishs at 54 ms 430 us
6:22: 2 starts at 55 ms 26 us and finishs at 57 ms 76 us
19:26:22 starts at 57 ms 78 us and finishs at 65 ms 679 us
19:28: 1 starts at 69 ms 670 us and finishs at 76 ms 221 us
7:16: 3 starts at 76 ms 221 us and finishs at 76 ms 221 us
6:21: 3 starts at 76 ms 232 us and finishs at 79 ms 32 us
18:29:10 starts at 79 ms 32 us and finishs at 95 ms 682 us
17:19: 5 starts at 95 ms 859 us and finishs at 95 ms 960 us
18:20: 4 starts at 96 ms 529 us and finishs at 97 ms 205 us
18:32: 3 starts at 97 ms 205 us and finishs at 98 ms 326 us
13: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
12: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
4: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
14: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
18:35: 1 starts at 99 ms 750 us and finishs at 100 ms 981 us
7:17: 1 starts at 100 ms 981 us and finishs at 101 ms 962 us
18:35: 4 starts at 102 ms 205 us and finishs at 104 ms 305 us
6:22: 2 starts at 104 ms 901 us and finishs at 106 ms 951 us
19:26:22 starts at 106 ms 955 us and finishs at 115 ms 556 us
6:19: 2 starts at 115 ms 556 us and finishs at 117 ms 957 us
5:17: 1 starts at 118 ms 1 us and finishs at 118 ms 982 us
19:28: 1 starts at 119 ms 545 us and finishs at 126 ms 96 us
18:32: 1 starts at 126 ms 96 us and finishs at 126 ms 96 us
6:21: 3 starts at 126 ms 115 us and finishs at 128 ms 915 us
18:17: 1 starts at 128 ms 915 us and finishs at 129 ms 896 us
6:22: 3 starts at 129 ms 896 us and finishs at 137 ms 547 us

6:22: 4 starts at 137 ms 547 us and finishs at 149 ms 547 us
18:35: 1 starts at 149 ms 625 us and finishs at 150 ms 856 us
18:35: 4 starts at 152 ms 80 us and finishs at 154 ms 180 us
6:22: 2 starts at 154 ms 776 us and finishs at 156 ms 826 us
19:26:22 starts at 156 ms 832 us and finishs at 165 ms 433 us
19:28: 1 starts at 169 ms 420 us and finishs at 175 ms 971 us
6:21: 3 starts at 175 ms 998 us and finishs at 178 ms 798 us
18:29:10 starts at 178 ms 917 us and finishs at 195 ms 567 us
18:29: 9 starts at 195 ms 567 us and finishs at 199 ms 646 us
6:22: 3 starts at 230 ms 127 us and finishs at 237 ms 778 us
The schedule for processor 1 is:
1: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
12: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
14: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
13: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
6:19: 1 starts at 0 ms 0 us and finishs at 1 ms 180 us
5:19: 1 starts at 1 ms 180 us and finishs at 2 ms 360 us
7:20: 1 starts at 2 ms 360 us and finishs at 3 ms 211 us
8:20: 1 starts at 3 ms 211 us and finishs at 4 ms 62 us
6:19: 4 starts at 4 ms 62 us and finishs at 6 ms 62 us
8:20: 3 starts at 6 ms 62 us and finishs at 7 ms 212 us
7:35: 1 starts at 7 ms 212 us and finishs at 8 ms 443 us
6:19: 3 starts at 8 ms 443 us and finishs at 10 ms 343 us
6:17: 1 starts at 10 ms 343 us and finishs at 11 ms 324 us
8:16: 1 starts at 11 ms 324 us and finishs at 11 ms 888 us
19:16: 1 starts at 11 ms 888 us and finishs at 12 ms 452 us
6:19: 1 starts at 12 ms 469 us and finishs at 13 ms 649 us
5:19: 1 starts at 13 ms 649 us and finishs at 14 ms 829 us
7:35: 4 starts at 14 ms 829 us and finishs at 16 ms 929 us
12: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
14: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
13: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
6:19: 1 starts at 24 ms 938 us and finishs at 26 ms 118 us
5:19: 1 starts at 26 ms 118 us and finishs at 27 ms 298 us
7:20: 1 starts at 27 ms 298 us and finishs at 28 ms 149 us
8:20: 1 starts at 28 ms 149 us and finishs at 29 ms 0 us
6:19: 4 starts at 29 ms 0 us and finishs at 31 ms 0 us
17:17: 1 starts at 31 ms 949 us and finishs at 32 ms 930 us
4: 0: 1 starts at 32 ms 930 us and finishs at 32 ms 930 us
17:16: 2 starts at 32 ms 930 us and finishs at 34 ms 57 us
7:35: 2 starts at 34 ms 972 us and finishs at 35 ms 800 us
12: 0: 1 starts at 36 ms 500 us and finishs at 36 ms 500 us
6:19: 1 starts at 37 ms 407 us and finishs at 38 ms 587 us
5:19: 1 starts at 38 ms 587 us and finishs at 39 ms 767 us
8:30:20 starts at 39 ms 767 us and finishs at 48 ms 922 us
12: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
14: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
13: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
6:19: 1 starts at 49 ms 876 us and finishs at 51 ms 56 us
5:19: 1 starts at 51 ms 56 us and finishs at 52 ms 236 us
7:20: 1 starts at 52 ms 236 us and finishs at 53 ms 87 us

8:20: 1 starts at 53 ms 87 us and finishs at 53 ms 938 us
6:19: 4 starts at 53 ms 938 us and finishs at 55 ms 938 us
8:20: 3 starts at 55 ms 938 us and finishs at 57 ms 88 us
7:35: 1 starts at 57 ms 88 us and finishs at 58 ms 319 us
6:19: 3 starts at 58 ms 319 us and finishs at 60 ms 219 us
6:19: 1 starts at 62 ms 345 us and finishs at 63 ms 525 us
5:19: 1 starts at 63 ms 525 us and finishs at 64 ms 705 us
7:35: 4 starts at 64 ms 773 us and finishs at 66 ms 873 us
6:21: 4 starts at 66 ms 873 us and finishs at 72 ms 774 us
12: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
14: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
13: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
6:19: 1 starts at 74 ms 814 us and finishs at 75 ms 994 us
5:19: 1 starts at 75 ms 994 us and finishs at 77 ms 174 us
7:20: 1 starts at 77 ms 174 us and finishs at 78 ms 25 us
8:20: 1 starts at 78 ms 25 us and finishs at 78 ms 876 us
6:19: 4 starts at 78 ms 876 us and finishs at 80 ms 876 us
13: 0: 1 starts at 82 ms 969 us and finishs at 82 ms 969 us
18:16: 2 starts at 83 ms 77 us and finishs at 84 ms 204 us
7:35: 2 starts at 84 ms 989 us and finishs at 85 ms 817 us
19:16: 3 starts at 85 ms 817 us and finishs at 85 ms 817 us
18:16: 3 starts at 85 ms 817 us and finishs at 85 ms 817 us
6:19: 1 starts at 87 ms 283 us and finishs at 88 ms 463 us
5:19: 1 starts at 88 ms 463 us and finishs at 89 ms 643 us
8:30:20 starts at 89 ms 813 us and finishs at 98 ms 968 us
1: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
12: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
14: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
13: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
6:19: 1 starts at 99 ms 752 us and finishs at 100 ms 932 us
5:19: 1 starts at 100 ms 932 us and finishs at 102 ms 112 us
7:20: 1 starts at 102 ms 112 us and finishs at 102 ms 963 us
8:20: 1 starts at 102 ms 963 us and finishs at 103 ms 814 us
6:19: 4 starts at 103 ms 814 us and finishs at 105 ms 814 us
8:20: 3 starts at 105 ms 814 us and finishs at 106 ms 964 us
7:35: 1 starts at 106 ms 964 us and finishs at 108 ms 195 us
6:19: 3 starts at 108 ms 195 us and finishs at 110 ms 95 us
6:17: 1 starts at 110 ms 95 us and finishs at 111 ms 76 us
6:19: 1 starts at 112 ms 221 us and finishs at 113 ms 401 us
5:19: 1 starts at 113 ms 401 us and finishs at 114 ms 581 us
7:35: 4 starts at 114 ms 717 us and finishs at 116 ms 817 us
17:22: 3 starts at 116 ms 817 us and finishs at 124 ms 468 us
12: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
14: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
13: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
6:19: 1 starts at 124 ms 690 us and finishs at 125 ms 870 us
5:19: 1 starts at 125 ms 870 us and finishs at 127 ms 50 us
7:20: 1 starts at 127 ms 50 us and finishs at 127 ms 901 us
8:20: 1 starts at 127 ms 901 us and finishs at 128 ms 752 us
6:19: 4 starts at 128 ms 752 us and finishs at 130 ms 752 us
17:17: 1 starts at 131 ms 713 us and finishs at 132 ms 694 us

7:35: 2 starts at 134 ms 947 us and finishs at 135 ms 775 us
6:19: 1 starts at 137 ms 159 us and finishs at 138 ms 339 us
5:19: 1 starts at 138 ms 339 us and finishs at 139 ms 519 us
8:30:20 starts at 139 ms 859 us and finishs at 149 ms 14 us
12: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
14: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
13: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
6:19: 1 starts at 149 ms 628 us and finishs at 150 ms 808 us
5:19: 1 starts at 150 ms 808 us and finishs at 151 ms 988 us
7:20: 1 starts at 151 ms 988 us and finishs at 152 ms 839 us
8:20: 1 starts at 152 ms 839 us and finishs at 153 ms 690 us
6:19: 4 starts at 153 ms 690 us and finishs at 155 ms 690 us
8:20: 3 starts at 155 ms 690 us and finishs at 156 ms 840 us
7:35: 1 starts at 156 ms 840 us and finishs at 158 ms 71 us
6:19: 3 starts at 158 ms 71 us and finishs at 159 ms 971 us
6:19: 1 starts at 162 ms 97 us and finishs at 163 ms 277 us
5:19: 1 starts at 163 ms 277 us and finishs at 164 ms 457 us
7:35: 4 starts at 164 ms 661 us and finishs at 166 ms 761 us
6:21: 4 starts at 166 ms 761 us and finishs at 172 ms 662 us
12: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
14: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
13: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
6:19: 1 starts at 174 ms 566 us and finishs at 175 ms 746 us
5:19: 1 starts at 175 ms 746 us and finishs at 176 ms 926 us
7:20: 1 starts at 176 ms 926 us and finishs at 177 ms 777 us
8:20: 1 starts at 177 ms 777 us and finishs at 178 ms 628 us
6:19: 4 starts at 178 ms 628 us and finishs at 180 ms 628 us
7:35: 2 starts at 184 ms 905 us and finishs at 185 ms 733 us
6:19: 1 starts at 187 ms 35 us and finishs at 188 ms 215 us
5:19: 1 starts at 188 ms 215 us and finishs at 189 ms 395 us
8:30:20 starts at 189 ms 905 us and finishs at 199 ms 60 us
17:22: 3 starts at 216 ms 929 us and finishs at 224 ms 580 us
The schedule for processor 2 is:
2: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
3: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
17:19: 1 starts at 0 ms 0 us and finishs at 1 ms 180 us
7:29: 1 starts at 1 ms 180 us and finishs at 3 ms 632 us
8:26:22 starts at 3 ms 632 us and finishs at 12 ms 233 us
17:19: 1 starts at 12 ms 469 us and finishs at 13 ms 649 us
7:20: 3 starts at 13 ms 649 us and finishs at 14 ms 799 us
19:17: 1 starts at 14 ms 799 us and finishs at 15 ms 780 us
18:33: 2 starts at 15 ms 802 us and finishs at 17 ms 745 us
16:22: 2 starts at 17 ms 745 us and finishs at 19 ms 795 us
5:21: 3 starts at 19 ms 795 us and finishs at 22 ms 595 us
5:22: 2 starts at 22 ms 613 us and finishs at 24 ms 663 us
2: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
3: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
17:19: 1 starts at 24 ms 938 us and finishs at 26 ms 118 us
7:29: 1 starts at 26 ms 119 us and finishs at 28 ms 571 us
5:19: 2 starts at 28 ms 571 us and finishs at 30 ms 972 us
8:18: 1 starts at 30 ms 972 us and finishs at 31 ms 722 us

11: 2: 3 starts at 31 ms 722 us and finishs at 32 ms 323 us
19:27: 1 starts at 32 ms 930 us and finishs at 36 ms 480 us
17:19: 1 starts at 37 ms 407 us and finishs at 38 ms 587 us
17:22: 2 starts at 38 ms 587 us and finishs at 40 ms 637 us
17:21: 3 starts at 40 ms 646 us and finishs at 43 ms 446 us
16:21: 4 starts at 43 ms 446 us and finishs at 49 ms 347 us
2: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
3: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
17:19: 1 starts at 49 ms 876 us and finishs at 51 ms 56 us
7:29: 1 starts at 51 ms 58 us and finishs at 53 ms 510 us
8:26:22 starts at 53 ms 510 us and finishs at 62 ms 111 us
17:19: 1 starts at 62 ms 345 us and finishs at 63 ms 525 us
7:20: 3 starts at 63 ms 525 us and finishs at 64 ms 675 us
22: 2: 3 starts at 64 ms 675 us and finishs at 65 ms 276 us
18:33: 2 starts at 65 ms 746 us and finishs at 67 ms 689 us
16:22: 2 starts at 67 ms 689 us and finishs at 69 ms 739 us
5:21: 3 starts at 69 ms 739 us and finishs at 72 ms 539 us
5:22: 2 starts at 72 ms 539 us and finishs at 74 ms 589 us
6:16: 3 starts at 74 ms 589 us and finishs at 74 ms 589 us
6:19: 5 starts at 74 ms 700 us and finishs at 74 ms 801 us
2: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
3: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
17:19: 1 starts at 74 ms 814 us and finishs at 75 ms 994 us
7:29: 1 starts at 75 ms 997 us and finishs at 78 ms 449 us
8:20: 4 starts at 78 ms 449 us and finishs at 79 ms 125 us
7:29: 2 starts at 79 ms 125 us and finishs at 80 ms 369 us
18:29: 2 starts at 80 ms 369 us and finishs at 81 ms 613 us
16:16: 2 starts at 81 ms 613 us and finishs at 82 ms 740 us
19:27: 1 starts at 82 ms 812 us and finishs at 86 ms 362 us
18:16: 1 starts at 86 ms 362 us and finishs at 86 ms 926 us
17:19: 1 starts at 87 ms 283 us and finishs at 88 ms 463 us
17:22: 2 starts at 88 ms 611 us and finishs at 90 ms 661 us
17:21: 3 starts at 90 ms 661 us and finishs at 93 ms 461 us
19:20: 4 starts at 95 ms 843 us and finishs at 96 ms 519 us
2: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
3: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
17:19: 1 starts at 99 ms 752 us and finishs at 100 ms 932 us
7:29: 1 starts at 100 ms 936 us and finishs at 103 ms 388 us
8:26:22 starts at 103 ms 388 us and finishs at 111 ms 989 us
17:19: 1 starts at 112 ms 221 us and finishs at 113 ms 401 us
7:20: 3 starts at 113 ms 401 us and finishs at 114 ms 551 us
19:17: 1 starts at 114 ms 551 us and finishs at 115 ms 532 us
18:33: 2 starts at 115 ms 690 us and finishs at 117 ms 633 us
16:22: 2 starts at 117 ms 633 us and finishs at 119 ms 683 us
5:21: 3 starts at 119 ms 683 us and finishs at 122 ms 483 us
5:22: 2 starts at 122 ms 483 us and finishs at 124 ms 533 us
2: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
3: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
17:19: 1 starts at 124 ms 690 us and finishs at 125 ms 870 us
7:29: 1 starts at 125 ms 875 us and finishs at 128 ms 327 us
5:19: 2 starts at 128 ms 327 us and finishs at 130 ms 728 us

8:18: 1 starts at 130 ms 728 us and finishs at 131 ms 478 us
19:27: 1 starts at 132 ms 694 us and finishs at 136 ms 244 us
17:19: 1 starts at 137 ms 159 us and finishs at 138 ms 339 us
17:22: 2 starts at 138 ms 635 us and finishs at 140 ms 685 us
17:21: 3 starts at 140 ms 685 us and finishs at 143 ms 485 us
16:21: 4 starts at 143 ms 485 us and finishs at 149 ms 386 us
2: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
3: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
17:19: 1 starts at 149 ms 628 us and finishs at 150 ms 808 us
7:29: 1 starts at 150 ms 814 us and finishs at 153 ms 266 us
8:26:22 starts at 153 ms 266 us and finishs at 161 ms 867 us
17:19: 1 starts at 162 ms 97 us and finishs at 163 ms 277 us
7:20: 3 starts at 163 ms 277 us and finishs at 164 ms 427 us
18:33: 2 starts at 165 ms 634 us and finishs at 167 ms 577 us
16:22: 2 starts at 167 ms 577 us and finishs at 169 ms 627 us
5:21: 3 starts at 169 ms 627 us and finishs at 172 ms 427 us
5:22: 2 starts at 172 ms 427 us and finishs at 174 ms 477 us
2: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
3: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
17:19: 1 starts at 174 ms 566 us and finishs at 175 ms 746 us
7:29: 1 starts at 175 ms 753 us and finishs at 178 ms 205 us
19:27: 1 starts at 182 ms 576 us and finishs at 186 ms 126 us
17:19: 1 starts at 187 ms 35 us and finishs at 188 ms 215 us
17:22: 2 starts at 188 ms 659 us and finishs at 190 ms 709 us
17:21: 3 starts at 190 ms 709 us and finishs at 193 ms 509 us
The schedule for processor 3 is:
4: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
1: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
18:29: 1 starts at 0 ms 3 us and finishs at 2 ms 455 us
5:19: 3 starts at 2 ms 455 us and finishs at 4 ms 355 us
8:20: 2 starts at 4 ms 355 us and finishs at 5 ms 156 us
19:16: 6 starts at 5 ms 156 us and finishs at 6 ms 171 us
8:16: 6 starts at 6 ms 171 us and finishs at 7 ms 186 us
8:28: 1 starts at 12 ms 374 us and finishs at 18 ms 925 us
8:17: 1 starts at 19 ms 556 us and finishs at 20 ms 537 us
4: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
1: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
18:29: 1 starts at 24 ms 942 us and finishs at 27 ms 394 us
8:27: 1 starts at 27 ms 394 us and finishs at 30 ms 944 us
16:17: 1 starts at 30 ms 944 us and finishs at 31 ms 925 us
7:29: 9 starts at 34 ms 628 us and finishs at 38 ms 707 us
4: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
1: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
18:29: 1 starts at 49 ms 881 us and finishs at 52 ms 333 us
5:19: 3 starts at 52 ms 333 us and finishs at 54 ms 233 us
16:22: 3 starts at 54 ms 233 us and finishs at 61 ms 884 us
8:28: 1 starts at 62 ms 258 us and finishs at 68 ms 809 us
14: 0: 1 starts at 68 ms 809 us and finishs at 68 ms 809 us
2: 0: 1 starts at 72 ms 60 us and finishs at 72 ms 60 us
3: 0: 1 starts at 72 ms 159 us and finishs at 72 ms 159 us
4: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us

1: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
18:29: 1 starts at 74 ms 820 us and finishs at 77 ms 272 us
8:27: 1 starts at 77 ms 276 us and finishs at 80 ms 826 us
5:21: 5 starts at 80 ms 826 us and finishs at 95 ms 427 us
16:19: 5 starts at 95 ms 748 us and finishs at 95 ms 849 us
6:16: 5 starts at 95 ms 849 us and finishs at 98 ms 890 us
4: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
1: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
18:29: 1 starts at 99 ms 759 us and finishs at 102 ms 211 us
5:19: 3 starts at 102 ms 211 us and finishs at 104 ms 111 us
8:20: 2 starts at 104 ms 111 us and finishs at 104 ms 912 us
18:16: 5 starts at 104 ms 912 us and finishs at 107 ms 953 us
17:16: 5 starts at 108 ms 21 us and finishs at 111 ms 62 us
8:28: 1 starts at 112 ms 139 us and finishs at 118 ms 690 us
8:17: 1 starts at 119 ms 354 us and finishs at 120 ms 335 us
4: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
1: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
18:29: 1 starts at 124 ms 698 us and finishs at 127 ms 150 us
8:27: 1 starts at 127 ms 158 us and finishs at 130 ms 708 us
16:17: 1 starts at 130 ms 708 us and finishs at 131 ms 689 us
17:21: 5 starts at 131 ms 689 us and finishs at 146 ms 290 us
4: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
1: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
18:29: 1 starts at 149 ms 637 us and finishs at 152 ms 89 us
5:19: 3 starts at 152 ms 89 us and finishs at 153 ms 989 us
16:22: 3 starts at 154 ms 60 us and finishs at 161 ms 711 us
8:28: 1 starts at 162 ms 20 us and finishs at 168 ms 571 us
4: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
1: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
18:29: 1 starts at 174 ms 576 us and finishs at 177 ms 28 us
8:27: 1 starts at 177 ms 40 us and finishs at 180 ms 590 us
5:22: 4 starts at 180 ms 590 us and finishs at 192 ms 590 us
The schedule for processor 4 is:
3: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
15: 0: 3 starts at 0 ms 0 us and finishs at 0 ms 0 us
16:19: 4 starts at 0 ms 0 us and finishs at 2 ms 0 us
6:21: 2 starts at 2 ms 0 us and finishs at 5 ms 151 us
19:18: 1 starts at 5 ms 151 us and finishs at 5 ms 901 us
6:16: 2 starts at 5 ms 901 us and finishs at 7 ms 28 us
15: 0: 3 starts at 24 ms 938 us and finishs at 24 ms 938 us
16:19: 4 starts at 24 ms 938 us and finishs at 26 ms 938 us
6:21: 2 starts at 26 ms 938 us and finishs at 30 ms 89 us
16:21: 3 starts at 30 ms 89 us and finishs at 32 ms 889 us
7:33: 2 starts at 32 ms 889 us and finishs at 34 ms 832 us
15: 0: 3 starts at 49 ms 876 us and finishs at 49 ms 876 us
16:19: 4 starts at 49 ms 876 us and finishs at 51 ms 876 us
6:21: 2 starts at 51 ms 876 us and finishs at 55 ms 27 us
7:29:10 starts at 55 ms 27 us and finishs at 71 ms 677 us
7:31: 1 starts at 71 ms 842 us and finishs at 73 ms 392 us
1: 0: 1 starts at 73 ms 392 us and finishs at 73 ms 392 us
7:20: 4 starts at 73 ms 392 us and finishs at 74 ms 68 us

20: 2: 3 starts at 74 ms 68 us and finishs at 74 ms 669 us
15: 0: 3 starts at 74 ms 814 us and finishs at 74 ms 814 us
16:19: 4 starts at 74 ms 814 us and finishs at 76 ms 814 us
6:21: 2 starts at 76 ms 814 us and finishs at 79 ms 965 us
16:21: 3 starts at 79 ms 967 us and finishs at 82 ms 767 us
7:33: 2 starts at 82 ms 906 us and finishs at 84 ms 849 us
15: 0: 1 starts at 84 ms 849 us and finishs at 84 ms 849 us
7:32: 3 starts at 84 ms 849 us and finishs at 85 ms 970 us
17:16: 3 starts at 85 ms 970 us and finishs at 85 ms 970 us
7:16: 5 starts at 85 ms 970 us and finishs at 89 ms 11 us
5:16: 5 starts at 89 ms 11 us and finishs at 92 ms 52 us
18:31: 1 starts at 95 ms 809 us and finishs at 95 ms 809 us
16:16: 5 starts at 95 ms 875 us and finishs at 98 ms 916 us
3: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
15: 0: 3 starts at 99 ms 752 us and finishs at 99 ms 752 us
16:19: 4 starts at 99 ms 752 us and finishs at 101 ms 752 us
6:21: 2 starts at 101 ms 752 us and finishs at 104 ms 903 us
19:18: 1 starts at 104 ms 903 us and finishs at 105 ms 653 us
16:21: 5 starts at 105 ms 653 us and finishs at 120 ms 254 us
15: 0: 3 starts at 124 ms 690 us and finishs at 124 ms 690 us
16:19: 4 starts at 124 ms 690 us and finishs at 126 ms 690 us
6:21: 2 starts at 126 ms 690 us and finishs at 129 ms 841 us
16:21: 3 starts at 129 ms 845 us and finishs at 132 ms 645 us
7:33: 2 starts at 132 ms 864 us and finishs at 134 ms 807 us
16:22: 4 starts at 134 ms 807 us and finishs at 146 ms 807 us
15: 0: 3 starts at 149 ms 628 us and finishs at 149 ms 628 us
16:19: 4 starts at 149 ms 628 us and finishs at 151 ms 628 us
6:21: 2 starts at 151 ms 628 us and finishs at 154 ms 779 us
7:29:10 starts at 154 ms 847 us and finishs at 171 ms 497 us
7:31: 1 starts at 171 ms 662 us and finishs at 173 ms 212 us
15: 0: 3 starts at 174 ms 566 us and finishs at 174 ms 566 us
16:19: 4 starts at 174 ms 566 us and finishs at 176 ms 566 us
6:21: 2 starts at 176 ms 566 us and finishs at 179 ms 717 us
16:21: 3 starts at 179 ms 723 us and finishs at 182 ms 523 us
7:33: 2 starts at 182 ms 822 us and finishs at 184 ms 765 us
6:21: 5 starts at 184 ms 765 us and finishs at 199 ms 366 us
17:22: 4 starts at 207 ms 28 us and finishs at 219 ms 28 us
The schedule for processor 5 is:
2: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
15: 0: 2 starts at 0 ms 0 us and finishs at 0 ms 0 us
16:19: 1 starts at 0 ms 0 us and finishs at 1 ms 180 us
5:19: 4 starts at 1 ms 180 us and finishs at 3 ms 180 us
17:21: 2 starts at 3 ms 180 us and finishs at 6 ms 331 us
17:19: 4 starts at 6 ms 331 us and finishs at 8 ms 331 us
16:21: 2 starts at 8 ms 331 us and finishs at 11 ms 482 us
7:20: 2 starts at 11 ms 482 us and finishs at 12 ms 283 us
16:19: 1 starts at 12 ms 469 us and finishs at 13 ms 649 us
5:21: 2 starts at 13 ms 649 us and finishs at 16 ms 800 us
17:16: 1 starts at 16 ms 800 us and finishs at 17 ms 364 us
18:35: 2 starts at 17 ms 886 us and finishs at 18 ms 714 us
5:16: 2 starts at 18 ms 714 us and finishs at 19 ms 841 us

19:16: 2 starts at 19 ms 841 us and finishs at 20 ms 968 us
7:16: 1 starts at 20 ms 968 us and finishs at 21 ms 532 us
16:16: 1 starts at 21 ms 532 us and finishs at 22 ms 96 us
16:19: 1 starts at 24 ms 938 us and finishs at 26 ms 118 us
5:19: 4 starts at 26 ms 118 us and finishs at 28 ms 118 us
17:21: 2 starts at 28 ms 118 us and finishs at 31 ms 269 us
17:19: 4 starts at 31 ms 269 us and finishs at 33 ms 269 us
16:21: 2 starts at 33 ms 269 us and finishs at 36 ms 420 us
16:19: 1 starts at 37 ms 407 us and finishs at 38 ms 587 us
5:21: 2 starts at 38 ms 587 us and finishs at 41 ms 738 us
17:21: 4 starts at 41 ms 738 us and finishs at 47 ms 639 us
16:19: 1 starts at 49 ms 876 us and finishs at 51 ms 56 us
5:19: 4 starts at 51 ms 56 us and finishs at 53 ms 56 us
17:21: 2 starts at 53 ms 56 us and finishs at 56 ms 207 us
17:19: 4 starts at 56 ms 207 us and finishs at 58 ms 207 us
16:21: 2 starts at 58 ms 207 us and finishs at 61 ms 358 us
16:19: 1 starts at 62 ms 345 us and finishs at 63 ms 525 us
5:21: 2 starts at 63 ms 525 us and finishs at 66 ms 676 us
18:35: 2 starts at 67 ms 761 us and finishs at 68 ms 589 us
5:21: 4 starts at 68 ms 589 us and finishs at 74 ms 490 us
5:16: 3 starts at 74 ms 490 us and finishs at 74 ms 490 us
8:16: 3 starts at 74 ms 490 us and finishs at 74 ms 490 us
5:19: 5 starts at 74 ms 490 us and finishs at 74 ms 591 us
16:19: 1 starts at 74 ms 814 us and finishs at 75 ms 994 us
5:19: 4 starts at 75 ms 994 us and finishs at 77 ms 994 us
17:21: 2 starts at 77 ms 994 us and finishs at 81 ms 145 us
17:19: 4 starts at 81 ms 145 us and finishs at 83 ms 145 us
16:21: 2 starts at 83 ms 145 us and finishs at 86 ms 296 us
16:16: 3 starts at 86 ms 296 us and finishs at 86 ms 296 us
16:19: 1 starts at 87 ms 283 us and finishs at 88 ms 463 us
5:21: 2 starts at 88 ms 463 us and finishs at 91 ms 614 us
5:22: 3 starts at 91 ms 614 us and finishs at 99 ms 265 us
2: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
15: 0: 2 starts at 99 ms 750 us and finishs at 99 ms 750 us
16:19: 1 starts at 99 ms 752 us and finishs at 100 ms 932 us
5:19: 4 starts at 100 ms 932 us and finishs at 102 ms 932 us
17:21: 2 starts at 102 ms 932 us and finishs at 106 ms 83 us
17:19: 4 starts at 106 ms 83 us and finishs at 108 ms 83 us
16:21: 2 starts at 108 ms 83 us and finishs at 111 ms 234 us
7:20: 2 starts at 111 ms 234 us and finishs at 112 ms 35 us
16:19: 1 starts at 112 ms 221 us and finishs at 113 ms 401 us
5:21: 2 starts at 113 ms 401 us and finishs at 116 ms 552 us
18:35: 2 starts at 117 ms 705 us and finishs at 118 ms 533 us
8:16: 5 starts at 118 ms 533 us and finishs at 121 ms 574 us
19:16: 5 starts at 121 ms 574 us and finishs at 124 ms 615 us
16:19: 1 starts at 124 ms 690 us and finishs at 125 ms 870 us
5:19: 4 starts at 125 ms 870 us and finishs at 127 ms 870 us
17:21: 2 starts at 127 ms 870 us and finishs at 131 ms 21 us
17:19: 4 starts at 131 ms 21 us and finishs at 133 ms 21 us
16:21: 2 starts at 133 ms 21 us and finishs at 136 ms 172 us
16:19: 1 starts at 137 ms 159 us and finishs at 138 ms 339 us

5:21: 2 starts at 138 ms 339 us and finishs at 141 ms 490 us
17:21: 4 starts at 141 ms 498 us and finishs at 147 ms 399 us
16:19: 1 starts at 149 ms 628 us and finishs at 150 ms 808 us
5:19: 4 starts at 150 ms 808 us and finishs at 152 ms 808 us
17:21: 2 starts at 152 ms 808 us and finishs at 155 ms 959 us
17:19: 4 starts at 155 ms 959 us and finishs at 157 ms 959 us
16:21: 2 starts at 157 ms 959 us and finishs at 161 ms 110 us
16:19: 1 starts at 162 ms 97 us and finishs at 163 ms 277 us
5:21: 2 starts at 163 ms 277 us and finishs at 166 ms 428 us
18:35: 2 starts at 167 ms 649 us and finishs at 168 ms 477 us
5:21: 4 starts at 168 ms 477 us and finishs at 174 ms 378 us
16:19: 1 starts at 174 ms 566 us and finishs at 175 ms 746 us
5:19: 4 starts at 175 ms 746 us and finishs at 177 ms 746 us
17:21: 2 starts at 177 ms 746 us and finishs at 180 ms 897 us
17:19: 4 starts at 180 ms 897 us and finishs at 182 ms 897 us
16:21: 2 starts at 182 ms 897 us and finishs at 186 ms 48 us
16:19: 1 starts at 187 ms 35 us and finishs at 188 ms 215 us
5:21: 2 starts at 188 ms 215 us and finishs at 191 ms 366 us
5:22: 3 starts at 191 ms 386 us and finishs at 199 ms 37 us
The schedule for the communications network is:
1: 0: 3 sends to 7:29: 1 starts at 0 ms 0 us and finishs at 0 ms 1 us
4: 0: 3 sends to 7:29: 1 starts at 0 ms 1 us and finishs at 0 ms 2 us
13: 0: 3 sends to 18:29: 1 starts at 0 ms 2 us and finishs at 0 ms 3 us
14: 0: 3 sends to 18:29: 1 starts at 0 ms 3 us and finishs at 0 ms 4 us
15: 0: 3 sends to 18:29: 1 starts at 0 ms 4 us and finishs at 0 ms 5 us
12: 0: 3 sends to 18:29: 1 starts at 0 ms 5 us and finishs at 0 ms 6 us
1: 0: 3 sends to 8:26:22 starts at 0 ms 6 us and finishs at 0 ms 8 us
4: 0: 3 sends to 8:26:22 starts at 0 ms 8 us and finishs at 0 ms 10 us
12: 0: 3 sends to 19:26:22 starts at 0 ms 10 us and finishs at 0 ms 12 us
13: 0: 3 sends to 19:26:22 starts at 0 ms 12 us and finishs at 0 ms 14 us
1: 0: 2 sends to 7:29: 1 starts at 0 ms 14 us and finishs at 0 ms 16 us
3: 0: 2 sends to 7:29: 1 starts at 0 ms 16 us and finishs at 0 ms 18 us
14: 0: 2 sends to 18:29: 1 starts at 0 ms 18 us and finishs at 0 ms 21 us
15: 0: 2 sends to 18:29: 1 starts at 0 ms 21 us and finishs at 0 ms 24 us
2: 0: 2 sends to 7:29: 1 starts at 0 ms 24 us and finishs at 0 ms 27 us
4: 0: 2 sends to 7:29: 1 starts at 0 ms 27 us and finishs at 0 ms 29 us
12: 0: 2 sends to 18:29: 1 starts at 0 ms 29 us and finishs at 0 ms 30 us
13: 0: 2 sends to 18:29: 1 starts at 0 ms 30 us and finishs at 0 ms 32 us
16:19: 1 sends to 18:32: 1 starts at 1 ms 180 us and finishs at 1 ms 183 us
16:19: 1 sends to 16:22: 4 starts at 1 ms 183 us and finishs at 1 ms 195 us
18:35: 1 sends to 18:29: 1 starts at 1 ms 231 us and finishs at 1 ms 237 us
18:35: 1 sends to 18:33: 2 starts at 1 ms 237 us and finishs at 1 ms 243 us
7:17: 1 sends to 6:17: 1 starts at 2 ms 212 us and finishs at 2 ms 218 us
5:19: 1 sends to 6:22: 4 starts at 2 ms 360 us and finishs at 2 ms 372 us
5:19: 1 sends to 7:32: 1 starts at 2 ms 372 us and finishs at 2 ms 375 us
18:29: 1 sends to 18:35: 4 starts at 2 ms 455 us and finishs at 2 ms 524 us
7:17: 1 sends to 8:30:20 starts at 2 ms 524 us and finishs at 2 ms 878 us
18:29: 1 sends to 17:16: 1 starts at 2 ms 878 us and finishs at 2 ms 947 us
18:29: 1 sends to 19:16: 1 starts at 2 ms 947 us and finishs at 3 ms 16 us
18:29: 1 sends to 16:16: 1 starts at 3 ms 16 us and finishs at 3 ms 85 us
5:19: 4 sends to 8:30:20 starts at 3 ms 180 us and finishs at 3 ms 222 us

5:19: 4 sends to 6:22: 4 starts at 3 ms 222 us and finishs at 3 ms 236 us
7:29: 1 sends to 7:35: 4 starts at 3 ms 632 us and finishs at 3 ms 701 us
5:19: 3 sends to 8:30:20 starts at 4 ms 355 us and finishs at 4 ms 410 us
18:35: 4 sends to 18:29: 1 starts at 4 ms 555 us and finishs at 4 ms 573 us
18:35: 4 sends to 18:33: 2 starts at 4 ms 573 us and finishs at 4 ms 642 us
6:21: 2 sends to 6:22: 2 starts at 5 ms 151 us and finishs at 5 ms 151 us
8:20: 2 sends to 8:30:20 starts at 5 ms 156 us and finishs at 5 ms 204 us
19:18: 1 sends to 12: 0: 1 starts at 5 ms 901 us and finishs at 5 ms 905 us
19:16: 6 sends to 19:28: 1 starts at 6 ms 171 us and finishs at 6 ms 171 us
19:16: 6 sends to 19:26:22 starts at 6 ms 171 us and finishs at 6 ms 171 us
6:16: 2 sends to 6:22: 4 starts at 7 ms 28 us and finishs at 7 ms 35 us
6:16: 2 sends to 7:32: 1 starts at 7 ms 35 us and finishs at 7 ms 36 us
6:16: 2 sends to 7:35: 1 starts at 7 ms 36 us and finishs at 7 ms 37 us
6:16: 2 sends to 8:26:22 starts at 7 ms 37 us and finishs at 7 ms 38 us
6:16: 2 sends to 8:27: 1 starts at 7 ms 38 us and finishs at 7 ms 39 us
6:16: 2 sends to 8:28: 1 starts at 7 ms 39 us and finishs at 7 ms 40 us
6:16: 2 sends to 8:30:20 starts at 7 ms 40 us and finishs at 7 ms 61 us
6:16: 2 sends to 8:16: 1 starts at 7 ms 61 us and finishs at 7 ms 62 us
7:29: 1 sends to 8:16: 1 starts at 7 ms 62 us and finishs at 7 ms 131 us
6:16: 2 sends to 5:16: 2 starts at 7 ms 131 us and finishs at 7 ms 144 us
6:16: 2 sends to 5:16: 1 starts at 7 ms 144 us and finishs at 7 ms 145 us
8:16: 6 sends to 8:26:22 starts at 7 ms 186 us and finishs at 7 ms 186 us
6:22: 2 sends to 8:28: 1 starts at 7 ms 201 us and finishs at 7 ms 207 us
7:29: 1 sends to 5:16: 1 starts at 7 ms 207 us and finishs at 7 ms 276 us
7:35: 1 sends to 7:29: 1 starts at 8 ms 443 us and finishs at 8 ms 449 us
7:35: 1 sends to 8:26:22 starts at 8 ms 449 us and finishs at 8 ms 455 us
6:17: 1 sends to 7:17: 1 starts at 11 ms 324 us and finishs at 11 ms 330 us
6:17: 1 sends to 5:17: 1 starts at 11 ms 330 us and finishs at 11 ms 336 us
16:21: 2 sends to 16:22: 2 starts at 11 ms 482 us and finishs at 11 ms 482 us
8:16: 1 sends to 7:29: 1 starts at 11 ms 888 us and finishs at 11 ms 893 us
8:26:22 sends to 6:22: 2 starts at 12 ms 233 us and finishs at 12 ms 374 us
8:26:22 sends to 8:28: 1 starts at 12 ms 374 us and finishs at 12 ms 385 us
8:26:22 sends to 7:35: 4 starts at 12 ms 385 us and finishs at 12 ms 425 us
8:26:22 sends to 5:21: 4 starts at 12 ms 425 us and finishs at 12 ms 430 us
8:26:22 sends to 6:21: 4 starts at 12 ms 430 us and finishs at 12 ms 435 us
7:20: 2 sends to 8:30:20 starts at 12 ms 435 us and finishs at 12 ms 483 us
19:16: 1 sends to 18:29: 1 starts at 12 ms 483 us and finishs at 12 ms 488 us
7:20: 3 sends to 8:30:20 starts at 14 ms 799 us and finishs at 14 ms 823 us
19:17: 1 sends to 18:17: 1 starts at 15 ms 780 us and finishs at 15 ms 786 us
19:26:22 sends to 18:33: 2 starts at 15 ms 802 us and finishs at 15 ms 827 us
19:26:22 sends to 16:21: 4 starts at 15 ms 827 us and finishs at 15 ms 832 us
19:26:22 sends to 17:21: 4 starts at 15 ms 832 us and finishs at 15 ms 837 us
16:21: 3 sends to 17:22: 3 starts at 16 ms 567 us and finishs at 16 ms 573 us
17:21: 4 sends to 17:22: 3 starts at 16 ms 573 us and finishs at 16 ms 573 us
17:21: 3 sends to 17:22: 3 starts at 16 ms 573 us and finishs at 16 ms 579 us
19:26:22 sends to 17:22: 3 starts at 16 ms 579 us and finishs at 16 ms 584 us
19:27: 1 sends to 17:22: 3 starts at 16 ms 584 us and finishs at 16 ms 589 us
19:28: 1 sends to 17:22: 3 starts at 16 ms 589 us and finishs at 16 ms 594 us
5:21: 2 sends to 5:22: 2 starts at 16 ms 800 us and finishs at 16 ms 800 us
5:21: 2 sends to 8:30:20 starts at 16 ms 800 us and finishs at 16 ms 802 us
7:35: 4 sends to 7:29: 1 starts at 16 ms 929 us and finishs at 16 ms 947 us

7:35: 4 sends to 8:26:22 starts at 16 ms 947 us and finishs at 17 ms 16 us
17:16: 1 sends to 18:29: 1 starts at 17 ms 364 us and finishs at 17 ms 369 us
19:26:22 sends to 16:22: 2 starts at 17 ms 745 us and finishs at 17 ms 886 us
18:33: 2 sends to 18:35: 2 starts at 17 ms 886 us and finishs at 17 ms 903 us
6:19: 2 sends to 8:30:20 starts at 18 ms 203 us and finishs at 18 ms 247 us
18:35: 2 sends to 18:33: 2 starts at 18 ms 714 us and finishs at 18 ms 734 us
18:35: 2 sends to 19:26:22 starts at 18 ms 734 us and finishs at 18 ms 740 us
8:28: 1 sends to 6:22: 2 starts at 18 ms 925 us and finishs at 18 ms 928 us
8:28: 1 sends to 8:26:22 starts at 18 ms 928 us and finishs at 18 ms 939 us
8:28: 1 sends to 5:21: 3 starts at 18 ms 939 us and finishs at 18 ms 942 us
17:22: 4 sends to 17:21: 5 starts at 19 ms 28 us and finishs at 19 ms 28 us
17:22: 4 sends to 17:21: 3 starts at 19 ms 28 us and finishs at 19 ms 30 us
17:22: 4 sends to 19:28: 1 starts at 19 ms 30 us and finishs at 19 ms 34 us
5:17: 1 sends to 6:17: 1 starts at 19 ms 184 us and finishs at 19 ms 190 us
5:17: 1 sends to 8:30:20 starts at 19 ms 190 us and finishs at 19 ms 544 us
5:17: 1 sends to 8:17: 1 starts at 19 ms 544 us and finishs at 19 ms 550 us
6:17: 1 sends to 8:17: 1 starts at 19 ms 550 us and finishs at 19 ms 556 us
7:17: 1 sends to 8:17: 1 starts at 19 ms 556 us and finishs at 19 ms 562 us
5:16: 1 sends to 7:29: 1 starts at 19 ms 748 us and finishs at 19 ms 753 us
16:22: 2 sends to 16:21: 3 starts at 19 ms 795 us and finishs at 19 ms 795 us
16:22: 2 sends to 19:28: 1 starts at 19 ms 795 us and finishs at 19 ms 801 us
19:26:22 sends to 16:21: 3 starts at 19 ms 801 us and finishs at 19 ms 804 us
5:17: 1 sends to 11: 2: 3 starts at 19 ms 804 us and finishs at 20 ms 252 us
5:16: 2 sends to 4: 0: 1 starts at 20 ms 252 us and finishs at 20 ms 253 us
5:16: 2 sends to 6:16: 2 starts at 20 ms 253 us and finishs at 20 ms 266 us
5:16: 2 sends to 6:22: 4 starts at 20 ms 266 us and finishs at 20 ms 273 us
5:16: 2 sends to 7:31: 1 starts at 20 ms 273 us and finishs at 20 ms 274 us
5:16: 2 sends to 7:32: 1 starts at 20 ms 274 us and finishs at 20 ms 275 us
5:16: 2 sends to 7:33: 2 starts at 20 ms 275 us and finishs at 20 ms 276 us
5:16: 2 sends to 7:35: 1 starts at 20 ms 276 us and finishs at 20 ms 277 us
5:16: 2 sends to 8:16: 1 starts at 20 ms 277 us and finishs at 20 ms 278 us
5:16: 2 sends to 8:26:22 starts at 20 ms 278 us and finishs at 20 ms 279 us
5:16: 2 sends to 8:27: 1 starts at 20 ms 279 us and finishs at 20 ms 280 us
5:16: 2 sends to 8:28: 1 starts at 20 ms 280 us and finishs at 20 ms 281 us
5:16: 2 sends to 8:30:20 starts at 20 ms 281 us and finishs at 20 ms 302 us
5:16: 2 sends to 7:20: 4 starts at 20 ms 302 us and finishs at 20 ms 303 us
6:16: 2 sends to 5:19: 5 starts at 20 ms 303 us and finishs at 20 ms 304 us
6:16: 2 sends to 7:16: 1 starts at 20 ms 304 us and finishs at 20 ms 305 us
7:29: 1 sends to 7:16: 1 starts at 20 ms 305 us and finishs at 20 ms 374 us
5:16: 2 sends to 6:16: 1 starts at 20 ms 374 us and finishs at 20 ms 375 us
7:29: 1 sends to 6:16: 1 starts at 20 ms 375 us and finishs at 20 ms 444 us
5:16: 2 sends to 8:20: 4 starts at 20 ms 444 us and finishs at 20 ms 445 us
6:16: 2 sends to 8:20: 4 starts at 20 ms 445 us and finishs at 20 ms 446 us
5:16: 2 sends to 6:19: 5 starts at 20 ms 446 us and finishs at 20 ms 447 us
5:16: 2 sends to 1: 0: 1 starts at 20 ms 447 us and finishs at 20 ms 448 us
5:16: 2 sends to 8:16: 2 starts at 20 ms 448 us and finishs at 20 ms 461 us
6:16: 2 sends to 8:16: 2 starts at 20 ms 461 us and finishs at 20 ms 474 us
5:16: 2 sends to 7:29: 2 starts at 20 ms 474 us and finishs at 20 ms 475 us
6:16: 2 sends to 7:29: 2 starts at 20 ms 475 us and finishs at 20 ms 476 us
5:16: 2 sends to 7:16: 2 starts at 20 ms 476 us and finishs at 20 ms 489 us
6:16: 2 sends to 7:16: 2 starts at 20 ms 489 us and finishs at 20 ms 502 us

5:16: 2 sends to 5:21: 5 starts at 20 ms 502 us and finishs at 20 ms 503 us
6:16: 2 sends to 5:21: 5 starts at 20 ms 503 us and finishs at 20 ms 504 us
5:16: 2 sends to 5:22: 4 starts at 20 ms 504 us and finishs at 20 ms 511 us
5:19: 1 sends to 5:22: 4 starts at 20 ms 511 us and finishs at 20 ms 523 us
5:19: 4 sends to 5:22: 4 starts at 20 ms 523 us and finishs at 20 ms 537 us
8:17: 1 sends to 5:17: 1 starts at 20 ms 537 us and finishs at 20 ms 543 us
8:17: 1 sends to 6:17: 1 starts at 20 ms 543 us and finishs at 20 ms 549 us
8:17: 1 sends to 7:17: 1 starts at 20 ms 549 us and finishs at 20 ms 555 us
8:17: 1 sends to 8:30:20 starts at 20 ms 555 us and finishs at 20 ms 909 us
6:17: 1 sends to 11: 2: 3 starts at 20 ms 909 us and finishs at 21 ms 357 us
7:17: 1 sends to 11: 2: 3 starts at 21 ms 357 us and finishs at 21 ms 805 us
7:16: 1 sends to 7:29: 1 starts at 21 ms 805 us and finishs at 21 ms 805 us
8:17: 1 sends to 11: 2: 3 starts at 21 ms 805 us and finishs at 22 ms 253 us
19:16: 2 sends to 12: 0: 1 starts at 22 ms 253 us and finishs at 22 ms 254 us
19:16: 2 sends to 16:19: 5 starts at 22 ms 254 us and finishs at 22 ms 255 us
19:16: 2 sends to 16:21: 5 starts at 22 ms 255 us and finishs at 22 ms 256 us
19:16: 2 sends to 17:19: 5 starts at 22 ms 256 us and finishs at 22 ms 257 us
19:16: 2 sends to 18:33: 2 starts at 22 ms 257 us and finishs at 22 ms 258 us
19:16: 2 sends to 18:35: 1 starts at 22 ms 258 us and finishs at 22 ms 259 us
19:16: 2 sends to 19:26:22 starts at 22 ms 259 us and finishs at 22 ms 260 us
19:16: 2 sends to 19:27: 1 starts at 22 ms 260 us and finishs at 22 ms 261 us
19:16: 2 sends to 19:28: 1 starts at 22 ms 261 us and finishs at 22 ms 262 us
19:16: 2 sends to 14: 0: 1 starts at 22 ms 262 us and finishs at 22 ms 263 us
19:18: 1 sends to 14: 0: 1 starts at 22 ms 263 us and finishs at 22 ms 267 us
16:16: 1 sends to 18:29: 1 starts at 22 ms 267 us and finishs at 22 ms 272 us
19:16: 2 sends to 17:16: 2 starts at 22 ms 272 us and finishs at 22 ms 285 us
5:16: 2 sends to 6:21: 5 starts at 22 ms 285 us and finishs at 22 ms 286 us
5:21: 3 sends to 6:22: 2 starts at 22 ms 595 us and finishs at 22 ms 603 us
5:21: 3 sends to 8:28: 1 starts at 22 ms 603 us and finishs at 22 ms 610 us
6:22: 2 sends to 5:22: 2 starts at 22 ms 610 us and finishs at 22 ms 613 us
8:28: 1 sends to 5:22: 2 starts at 22 ms 613 us and finishs at 22 ms 616 us
5:21: 3 sends to 8:30:20 starts at 22 ms 616 us and finishs at 22 ms 653 us
5:21: 3 sends to 8:27: 1 starts at 22 ms 653 us and finishs at 22 ms 660 us
8:26:22 sends to 8:27: 1 starts at 22 ms 660 us and finishs at 22 ms 675 us
5:21: 3 sends to 6:21: 3 starts at 22 ms 675 us and finishs at 22 ms 680 us
8:26:22 sends to 6:21: 3 starts at 22 ms 680 us and finishs at 22 ms 683 us
8:28: 1 sends to 6:21: 3 starts at 22 ms 683 us and finishs at 22 ms 686 us
5:21: 3 sends to 7:29:10 starts at 22 ms 686 us and finishs at 22 ms 692 us
5:21: 3 sends to 6:22: 3 starts at 22 ms 692 us and finishs at 22 ms 698 us
5:21: 3 sends to 5:22: 3 starts at 22 ms 698 us and finishs at 22 ms 704 us
5:21: 3 sends to 7:32: 1 starts at 22 ms 704 us and finishs at 22 ms 712 us
6:19: 1 sends to 7:32: 1 starts at 22 ms 712 us and finishs at 22 ms 715 us
17:22: 3 sends to 17:21: 4 starts at 24 ms 580 us and finishs at 24 ms 580 us
17:22: 3 sends to 18:29:10 starts at 24 ms 580 us and finishs at 24 ms 614 us
5:22: 2 sends to 6:22: 2 starts at 24 ms 663 us and finishs at 24 ms 666 us
5:22: 2 sends to 8:28: 1 starts at 24 ms 666 us and finishs at 24 ms 672 us
5:22: 2 sends to 8:30:20 starts at 24 ms 672 us and finishs at 24 ms 677 us
6:21: 2 sends to 8:30:20 starts at 24 ms 677 us and finishs at 24 ms 679 us
6:22: 2 sends to 8:30:20 starts at 24 ms 679 us and finishs at 24 ms 684 us
7:29: 1 sends to 8:30:20 starts at 24 ms 684 us and finishs at 24 ms 699 us
8:26:22 sends to 8:30:20 starts at 24 ms 699 us and finishs at 24 ms 762 us

8:28: 1 sends to 8:30:20 starts at 24 ms 762 us and finishs at 24 ms 788 us
5:22: 2 sends to 7:33: 2 starts at 24 ms 788 us and finishs at 24 ms 792 us
6:22: 2 sends to 7:33: 2 starts at 24 ms 792 us and finishs at 24 ms 796 us
7:35: 1 sends to 7:33: 2 starts at 24 ms 796 us and finishs at 24 ms 802 us
7:35: 4 sends to 7:33: 2 starts at 24 ms 802 us and finishs at 24 ms 871 us
8:26:22 sends to 7:33: 2 starts at 24 ms 871 us and finishs at 24 ms 896 us
15: 0: 3 sends to 18:29: 1 starts at 24 ms 938 us and finishs at 24 ms 939 us
12: 0: 3 sends to 18:29: 1 starts at 24 ms 939 us and finishs at 24 ms 940 us
13: 0: 3 sends to 18:29: 1 starts at 24 ms 941 us and finishs at 24 ms 942 us
14: 0: 3 sends to 18:29: 1 starts at 24 ms 942 us and finishs at 24 ms 943 us
1: 0: 3 sends to 7:29: 1 starts at 26 ms 118 us and finishs at 26 ms 119 us
4: 0: 3 sends to 7:29: 1 starts at 26 ms 119 us and finishs at 26 ms 120 us
19:28: 1 sends to 16:22: 2 starts at 26 ms 346 us and finishs at 26 ms 349 us
19:28: 1 sends to 16:21: 3 starts at 26 ms 349 us and finishs at 26 ms 352 us
6:21: 3 sends to 5:21: 3 starts at 29 ms 146 us and finishs at 29 ms 151 us
6:21: 3 sends to 5:22: 2 starts at 29 ms 151 us and finishs at 29 ms 159 us
6:21: 3 sends to 8:26:22 starts at 29 ms 159 us and finishs at 29 ms 174 us
6:21: 3 sends to 8:27: 1 starts at 29 ms 174 us and finishs at 29 ms 181 us
6:21: 3 sends to 8:28: 1 starts at 29 ms 181 us and finishs at 29 ms 188 us
6:21: 3 sends to 8:30:20 starts at 29 ms 188 us and finishs at 29 ms 225 us
6:21: 3 sends to 7:29:10 starts at 29 ms 225 us and finishs at 29 ms 231 us
8:26:22 sends to 7:32: 1 starts at 29 ms 231 us and finishs at 29 ms 242 us
6:21: 4 sends to 6:22: 3 starts at 29 ms 646 us and finishs at 29 ms 646 us
5:21: 3 sends to 6:22: 3 starts at 29 ms 646 us and finishs at 29 ms 652 us
8:26:22 sends to 6:22: 3 starts at 29 ms 652 us and finishs at 29 ms 657 us
8:27: 1 sends to 6:22: 3 starts at 29 ms 657 us and finishs at 29 ms 662 us
8:28: 1 sends to 6:22: 3 starts at 29 ms 662 us and finishs at 29 ms 667 us
18:17: 1 sends to 19:17: 1 starts at 30 ms 127 us and finishs at 30 ms 133 us
18:17: 1 sends to 16:17: 1 starts at 30 ms 133 us and finishs at 30 ms 139 us
19:17: 1 sends to 16:17: 1 starts at 30 ms 139 us and finishs at 30 ms 145 us
8:27: 1 sends to 8:26:22 starts at 30 ms 944 us and finishs at 30 ms 949 us
8:27: 1 sends to 8:30:20 starts at 30 ms 949 us and finishs at 30 ms 958 us
8:27: 1 sends to 5:21: 4 starts at 30 ms 958 us and finishs at 30 ms 963 us
8:28: 1 sends to 5:21: 4 starts at 30 ms 963 us and finishs at 30 ms 968 us
5:19: 2 sends to 8:30:20 starts at 30 ms 972 us and finishs at 31 ms 16 us
8:27: 1 sends to 6:21: 4 starts at 31 ms 16 us and finishs at 31 ms 21 us
8:28: 1 sends to 6:21: 4 starts at 31 ms 21 us and finishs at 31 ms 26 us
8:27: 1 sends to 7:32: 1 starts at 31 ms 26 us and finishs at 31 ms 31 us
8:28: 1 sends to 7:32: 1 starts at 31 ms 31 us and finishs at 31 ms 40 us
16:17: 1 sends to 18:17: 1 starts at 31 ms 925 us and finishs at 31 ms 931 us
16:17: 1 sends to 19:17: 1 starts at 31 ms 931 us and finishs at 31 ms 937 us
16:17: 1 sends to 17:17: 1 starts at 31 ms 937 us and finishs at 31 ms 943 us
18:17: 1 sends to 17:17: 1 starts at 31 ms 943 us and finishs at 31 ms 949 us
19:17: 1 sends to 17:17: 1 starts at 31 ms 949 us and finishs at 31 ms 955 us
11: 2: 3 sends to 5:17: 1 starts at 32 ms 323 us and finishs at 32 ms 372 us
11: 2: 3 sends to 6:17: 1 starts at 32 ms 372 us and finishs at 32 ms 421 us
11: 2: 3 sends to 7:17: 1 starts at 32 ms 421 us and finishs at 32 ms 470 us
11: 2: 3 sends to 8:17: 1 starts at 32 ms 470 us and finishs at 32 ms 519 us
11: 2: 3 sends to 4: 0: 1 starts at 32 ms 519 us and finishs at 32 ms 531 us
6:16: 2 sends to 4: 0: 1 starts at 32 ms 531 us and finishs at 32 ms 532 us
8:18: 1 sends to 4: 0: 1 starts at 32 ms 532 us and finishs at 32 ms 536 us

11: 2: 3 sends to 2: 0: 1 starts at 32 ms 536 us and finishs at 32 ms 548 us
5:16: 2 sends to 2: 0: 1 starts at 32 ms 548 us and finishs at 32 ms 549 us
6:16: 2 sends to 2: 0: 1 starts at 32 ms 549 us and finishs at 32 ms 550 us
11: 2: 3 sends to 3: 0: 1 starts at 32 ms 550 us and finishs at 32 ms 562 us
5:16: 2 sends to 3: 0: 1 starts at 32 ms 562 us and finishs at 32 ms 563 us
6:16: 2 sends to 3: 0: 1 starts at 32 ms 563 us and finishs at 32 ms 564 us
16:21: 3 sends to 16:22: 2 starts at 32 ms 889 us and finishs at 32 ms 901 us
16:21: 3 sends to 19:26:22 starts at 32 ms 901 us and finishs at 32 ms 916 us
16:21: 3 sends to 19:28: 1 starts at 32 ms 916 us and finishs at 32 ms 923 us
16:21: 3 sends to 19:27: 1 starts at 32 ms 923 us and finishs at 32 ms 930 us
19:26:22 sends to 19:27: 1 starts at 32 ms 930 us and finishs at 32 ms 945 us
17:21: 2 sends to 17:22: 2 starts at 32 ms 945 us and finishs at 32 ms 945 us
16:21: 3 sends to 17:22: 2 starts at 32 ms 945 us and finishs at 32 ms 953 us
19:26:22 sends to 17:22: 2 starts at 32 ms 953 us and finishs at 33 ms 94 us
19:28: 1 sends to 17:22: 2 starts at 33 ms 94 us and finishs at 33 ms 97 us
16:21: 3 sends to 17:21: 3 starts at 33 ms 97 us and finishs at 33 ms 102 us
16:21: 3 sends to 16:22: 3 starts at 33 ms 102 us and finishs at 33 ms 108 us
16:21: 3 sends to 18:29:10 starts at 33 ms 108 us and finishs at 33 ms 114 us
16:21: 3 sends to 17:22: 3 starts at 33 ms 114 us and finishs at 33 ms 120 us
17:17: 1 sends to 16:17: 1 starts at 33 ms 120 us and finishs at 33 ms 126 us
17:17: 1 sends to 18:17: 1 starts at 33 ms 126 us and finishs at 33 ms 132 us
17:17: 1 sends to 19:17: 1 starts at 33 ms 132 us and finishs at 33 ms 138 us
4: 0: 1 sends to 11: 2: 3 starts at 33 ms 138 us and finishs at 33 ms 144 us
4: 0: 1 sends to 9: 2: 3 starts at 33 ms 144 us and finishs at 33 ms 150 us
6:17: 1 sends to 9: 2: 3 starts at 33 ms 150 us and finishs at 33 ms 598 us
8:17: 1 sends to 9: 2: 3 starts at 33 ms 598 us and finishs at 34 ms 46 us
16:21: 3 sends to 18:32: 1 starts at 34 ms 46 us and finishs at 34 ms 54 us
4: 0: 1 sends to 7:29: 9 starts at 34 ms 54 us and finishs at 34 ms 438 us
5:17: 1 sends to 7:29: 9 starts at 34 ms 438 us and finishs at 34 ms 533 us
6:17: 1 sends to 7:29: 9 starts at 34 ms 533 us and finishs at 34 ms 628 us
7:17: 1 sends to 7:29: 9 starts at 34 ms 628 us and finishs at 34 ms 723 us
17:16: 2 sends to 14: 0: 1 starts at 34 ms 723 us and finishs at 34 ms 724 us
17:16: 2 sends to 16:16: 1 starts at 34 ms 724 us and finishs at 34 ms 725 us
17:16: 2 sends to 16:19: 5 starts at 34 ms 725 us and finishs at 34 ms 726 us
17:16: 2 sends to 16:21: 5 starts at 34 ms 726 us and finishs at 34 ms 727 us
17:16: 2 sends to 16:22: 4 starts at 34 ms 727 us and finishs at 34 ms 734 us
17:16: 2 sends to 17:19: 5 starts at 34 ms 734 us and finishs at 34 ms 735 us
17:16: 2 sends to 18:20: 4 starts at 34 ms 735 us and finishs at 34 ms 736 us
17:16: 2 sends to 18:32: 1 starts at 34 ms 736 us and finishs at 34 ms 737 us
17:16: 2 sends to 18:33: 2 starts at 34 ms 737 us and finishs at 34 ms 738 us
17:16: 2 sends to 18:35: 1 starts at 34 ms 738 us and finishs at 34 ms 739 us
17:16: 2 sends to 19:16: 2 starts at 34 ms 739 us and finishs at 34 ms 752 us
17:16: 2 sends to 19:20: 4 starts at 34 ms 752 us and finishs at 34 ms 753 us
17:16: 2 sends to 19:26:22 starts at 34 ms 753 us and finishs at 34 ms 754 us
17:16: 2 sends to 19:27: 1 starts at 34 ms 754 us and finishs at 34 ms 755 us
17:16: 2 sends to 19:28: 1 starts at 34 ms 755 us and finishs at 34 ms 756 us
17:16: 2 sends to 18:29: 2 starts at 34 ms 756 us and finishs at 34 ms 757 us
19:16: 2 sends to 18:29: 2 starts at 34 ms 757 us and finishs at 34 ms 758 us
17:16: 2 sends to 16:16: 2 starts at 34 ms 758 us and finishs at 34 ms 771 us
19:16: 2 sends to 16:16: 2 starts at 34 ms 771 us and finishs at 34 ms 784 us
7:33: 2 sends to 5:22: 2 starts at 34 ms 832 us and finishs at 34 ms 900 us

7:33: 2 sends to 6:22: 2 starts at 34 ms 900 us and finishs at 34 ms 968 us
7:33: 2 sends to 8:30:20 starts at 34 ms 968 us and finishs at 34 ms 972 us
7:33: 2 sends to 7:35: 2 starts at 34 ms 972 us and finishs at 34 ms 989 us
7:35: 1 sends to 7:29:10 starts at 34 ms 989 us and finishs at 34 ms 998 us
7:35: 4 sends to 7:29:10 starts at 34 ms 998 us and finishs at 35 ms 15 us
8:26:22 sends to 7:29:10 starts at 35 ms 15 us and finishs at 35 ms 29 us
8:27: 1 sends to 7:29:10 starts at 35 ms 29 us and finishs at 35 ms 38 us
8:28: 1 sends to 7:29:10 starts at 35 ms 38 us and finishs at 35 ms 47 us
7:35: 2 sends to 7:33: 2 starts at 35 ms 800 us and finishs at 35 ms 820 us
7:35: 2 sends to 8:26:22 starts at 35 ms 820 us and finishs at 35 ms 826 us
19:27: 1 sends to 19:26:22 starts at 36 ms 480 us and finishs at 36 ms 485 us
19:28: 1 sends to 16:21: 4 starts at 36 ms 485 us and finishs at 36 ms 490 us
19:27: 1 sends to 17:21: 4 starts at 36 ms 490 us and finishs at 36 ms 495 us
19:28: 1 sends to 17:21: 4 starts at 36 ms 495 us and finishs at 36 ms 500 us
19:27: 1 sends to 12: 0: 1 starts at 36 ms 500 us and finishs at 36 ms 501 us
12: 0: 1 sends to 22: 2: 3 starts at 36 ms 501 us and finishs at 36 ms 507 us
16:17: 1 sends to 22: 2: 3 starts at 36 ms 507 us and finishs at 36 ms 955 us
17:17: 1 sends to 22: 2: 3 starts at 36 ms 955 us and finishs at 37 ms 403 us
19:27: 1 sends to 14: 0: 1 starts at 37 ms 403 us and finishs at 37 ms 404 us
12: 0: 1 sends to 20: 2: 3 starts at 37 ms 404 us and finishs at 37 ms 410 us
12: 0: 1 sends to 16:16: 5 starts at 37 ms 410 us and finishs at 37 ms 447 us
12: 0: 1 sends to 18:16: 5 starts at 37 ms 447 us and finishs at 37 ms 484 us
12: 0: 1 sends to 17:16: 3 starts at 37 ms 484 us and finishs at 37 ms 501 us
12: 0: 1 sends to 17:16: 5 starts at 37 ms 501 us and finishs at 37 ms 538 us
12: 0: 1 sends to 16:16: 3 starts at 37 ms 538 us and finishs at 37 ms 555 us
12: 0: 1 sends to 19:16: 5 starts at 37 ms 555 us and finishs at 37 ms 592 us
6:22: 3 sends to 6:21: 4 starts at 37 ms 778 us and finishs at 37 ms 778 us
6:22: 3 sends to 7:29:10 starts at 37 ms 778 us and finishs at 37 ms 812 us
6:22: 3 sends to 8:30:20 starts at 37 ms 812 us and finishs at 37 ms 819 us
18:17: 1 sends to 22: 2: 3 starts at 37 ms 819 us and finishs at 38 ms 267 us
9: 2: 3 sends to 4: 0: 1 starts at 38 ms 379 us and finishs at 38 ms 391 us
9: 2: 3 sends to 6:17: 1 starts at 38 ms 391 us and finishs at 38 ms 440 us
9: 2: 3 sends to 8:17: 1 starts at 38 ms 440 us and finishs at 38 ms 489 us
7:29: 9 sends to 4: 0: 1 starts at 38 ms 707 us and finishs at 38 ms 712 us
7:29: 9 sends to 5:17: 1 starts at 38 ms 712 us and finishs at 38 ms 755 us
7:29: 9 sends to 6:17: 1 starts at 38 ms 755 us and finishs at 38 ms 798 us
7:29: 9 sends to 7:17: 1 starts at 38 ms 798 us and finishs at 38 ms 841 us
6:16: 1 sends to 7:29: 1 starts at 38 ms 943 us and finishs at 38 ms 948 us
12: 0: 1 sends to 18:29: 9 starts at 38 ms 948 us and finishs at 39 ms 332 us
8:16: 2 sends to 1: 0: 1 starts at 40 ms 70 us and finishs at 40 ms 71 us
8:16: 2 sends to 4: 0: 1 starts at 40 ms 71 us and finishs at 40 ms 72 us
8:16: 2 sends to 5:16: 2 starts at 40 ms 72 us and finishs at 40 ms 85 us
8:16: 2 sends to 5:19: 5 starts at 40 ms 85 us and finishs at 40 ms 86 us
8:16: 2 sends to 6:16: 2 starts at 40 ms 86 us and finishs at 40 ms 99 us
8:16: 2 sends to 6:19: 5 starts at 40 ms 99 us and finishs at 40 ms 100 us
8:16: 2 sends to 7:16: 1 starts at 40 ms 100 us and finishs at 40 ms 101 us
8:16: 2 sends to 7:20: 4 starts at 40 ms 101 us and finishs at 40 ms 102 us
8:16: 2 sends to 7:31: 1 starts at 40 ms 102 us and finishs at 40 ms 103 us
8:16: 2 sends to 7:33: 2 starts at 40 ms 103 us and finishs at 40 ms 104 us
8:16: 2 sends to 7:35: 1 starts at 40 ms 104 us and finishs at 40 ms 105 us
8:16: 2 sends to 8:20: 4 starts at 40 ms 105 us and finishs at 40 ms 106 us

8:16: 2 sends to 8:26:22 starts at 40 ms 106 us and finishs at 40 ms 107 us
8:16: 2 sends to 8:27: 1 starts at 40 ms 107 us and finishs at 40 ms 108 us
8:16: 2 sends to 8:28: 1 starts at 40 ms 108 us and finishs at 40 ms 109 us
8:16: 2 sends to 8:30:20 starts at 40 ms 109 us and finishs at 40 ms 132 us
8:16: 2 sends to 7:29: 2 starts at 40 ms 132 us and finishs at 40 ms 133 us
17:22: 2 sends to 19:28: 1 starts at 40 ms 637 us and finishs at 40 ms 643 us
19:26:22 sends to 17:21: 3 starts at 40 ms 643 us and finishs at 40 ms 646 us
19:28: 1 sends to 17:21: 3 starts at 40 ms 646 us and finishs at 40 ms 649 us
7:16: 2 sends to 1: 0: 1 starts at 41 ms 197 us and finishs at 41 ms 198 us
7:16: 2 sends to 4: 0: 1 starts at 41 ms 198 us and finishs at 41 ms 199 us
7:16: 2 sends to 5:16: 2 starts at 41 ms 199 us and finishs at 41 ms 212 us
7:16: 2 sends to 5:19: 5 starts at 41 ms 212 us and finishs at 41 ms 213 us
7:16: 2 sends to 6:16: 2 starts at 41 ms 213 us and finishs at 41 ms 226 us
7:16: 2 sends to 6:19: 5 starts at 41 ms 226 us and finishs at 41 ms 227 us
7:16: 2 sends to 7:20: 4 starts at 41 ms 227 us and finishs at 41 ms 228 us
7:16: 2 sends to 7:29: 2 starts at 41 ms 228 us and finishs at 41 ms 229 us
7:16: 2 sends to 7:31: 1 starts at 41 ms 229 us and finishs at 41 ms 231 us
7:16: 2 sends to 7:33: 2 starts at 41 ms 231 us and finishs at 41 ms 233 us
7:16: 2 sends to 7:35: 1 starts at 41 ms 233 us and finishs at 41 ms 234 us
7:16: 2 sends to 8:16: 1 starts at 41 ms 234 us and finishs at 41 ms 235 us
7:16: 2 sends to 8:16: 3 starts at 41 ms 235 us and finishs at 41 ms 257 us
7:16: 2 sends to 8:20: 4 starts at 41 ms 257 us and finishs at 41 ms 258 us
7:16: 2 sends to 8:26:22 starts at 41 ms 258 us and finishs at 41 ms 259 us
7:16: 2 sends to 8:27: 1 starts at 41 ms 259 us and finishs at 41 ms 260 us
7:16: 2 sends to 8:28: 1 starts at 41 ms 260 us and finishs at 41 ms 261 us
7:16: 2 sends to 8:30:20 starts at 41 ms 261 us and finishs at 41 ms 282 us
7:16: 2 sends to 2: 0: 1 starts at 41 ms 282 us and finishs at 41 ms 283 us
7:16: 2 sends to 3: 0: 1 starts at 41 ms 283 us and finishs at 41 ms 284 us
7:16: 2 sends to 5:21: 5 starts at 41 ms 284 us and finishs at 41 ms 285 us
17:21: 3 sends to 16:21: 3 starts at 43 ms 446 us and finishs at 43 ms 451 us
17:21: 3 sends to 19:26:22 starts at 43 ms 451 us and finishs at 43 ms 466 us
17:21: 3 sends to 19:28: 1 starts at 43 ms 466 us and finishs at 43 ms 473 us
17:21: 3 sends to 17:22: 3 starts at 43 ms 473 us and finishs at 43 ms 479 us
17:21: 4 sends to 17:22: 3 starts at 47 ms 639 us and finishs at 47 ms 639 us
19:26:22 sends to 17:22: 3 starts at 47 ms 639 us and finishs at 47 ms 644 us
19:27: 1 sends to 17:22: 3 starts at 47 ms 644 us and finishs at 47 ms 649 us
19:28: 1 sends to 17:22: 3 starts at 47 ms 649 us and finishs at 47 ms 654 us
8:30:20 sends to 7:29:10 starts at 48 ms 922 us and finishs at 48 ms 931 us
16:21: 4 sends to 16:22: 3 starts at 49 ms 347 us and finishs at 49 ms 347 us
17:21: 3 sends to 16:22: 3 starts at 49 ms 347 us and finishs at 49 ms 353 us
19:26:22 sends to 16:22: 3 starts at 49 ms 353 us and finishs at 49 ms 358 us
19:27: 1 sends to 16:22: 3 starts at 49 ms 358 us and finishs at 49 ms 363 us
19:28: 1 sends to 16:22: 3 starts at 49 ms 363 us and finishs at 49 ms 368 us
15: 0: 3 sends to 18:29: 1 starts at 49 ms 876 us and finishs at 49 ms 877 us
12: 0: 3 sends to 18:29: 1 starts at 49 ms 877 us and finishs at 49 ms 878 us
13: 0: 3 sends to 18:29: 1 starts at 49 ms 880 us and finishs at 49 ms 881 us
14: 0: 3 sends to 18:29: 1 starts at 49 ms 881 us and finishs at 49 ms 882 us
1: 0: 3 sends to 7:29: 1 starts at 51 ms 57 us and finishs at 51 ms 58 us
4: 0: 3 sends to 7:29: 1 starts at 51 ms 58 us and finishs at 51 ms 59 us
18:35: 1 sends to 18:29: 1 starts at 51 ms 106 us and finishs at 51 ms 112 us
18:35: 1 sends to 18:33: 2 starts at 51 ms 112 us and finishs at 51 ms 118 us

18:29: 1 sends to 18:35: 4 starts at 52 ms 330 us and finishs at 52 ms 399 us
1: 0: 3 sends to 8:26:22 starts at 53 ms 507 us and finishs at 53 ms 509 us
4: 0: 3 sends to 8:26:22 starts at 53 ms 509 us and finishs at 53 ms 511 us
5:19: 3 sends to 8:30:20 starts at 54 ms 233 us and finishs at 54 ms 288 us
18:35: 4 sends to 18:29: 1 starts at 54 ms 430 us and finishs at 54 ms 448 us
6:21: 2 sends to 6:22: 2 starts at 55 ms 26 us and finishs at 55 ms 26 us
12: 0: 3 sends to 19:26:22 starts at 57 ms 76 us and finishs at 57 ms 78 us
13: 0: 3 sends to 19:26:22 starts at 57 ms 78 us and finishs at 57 ms 80 us
7:35: 1 sends to 7:29: 1 starts at 58 ms 319 us and finishs at 58 ms 325 us
7:35: 1 sends to 8:26:22 starts at 58 ms 325 us and finishs at 58 ms 331 us
16:22: 3 sends to 16:21: 4 starts at 61 ms 884 us and finishs at 61 ms 884 us
16:22: 3 sends to 18:29:10 starts at 61 ms 884 us and finishs at 61 ms 918 us
18:33: 2 sends to 18:29:10 starts at 61 ms 918 us and finishs at 61 ms 918 us
17:21: 3 sends to 18:29:10 starts at 61 ms 918 us and finishs at 61 ms 924 us
19:27: 1 sends to 18:29:10 starts at 61 ms 924 us and finishs at 61 ms 933 us
8:26:22 sends to 6:22: 2 starts at 62 ms 111 us and finishs at 62 ms 252 us
6:22: 2 sends to 8:28: 1 starts at 62 ms 252 us and finishs at 62 ms 258 us
8:26:22 sends to 8:28: 1 starts at 62 ms 258 us and finishs at 62 ms 269 us
7:20: 3 sends to 8:30:20 starts at 64 ms 675 us and finishs at 64 ms 699 us
7:29: 1 sends to 7:35: 4 starts at 64 ms 704 us and finishs at 64 ms 773 us
8:26:22 sends to 7:35: 4 starts at 64 ms 773 us and finishs at 64 ms 813 us
22: 2: 3 sends to 12: 0: 1 starts at 65 ms 276 us and finishs at 65 ms 288 us
22: 2: 3 sends to 16:17: 1 starts at 65 ms 288 us and finishs at 65 ms 337 us
22: 2: 3 sends to 17:17: 1 starts at 65 ms 337 us and finishs at 65 ms 386 us
22: 2: 3 sends to 18:17: 1 starts at 65 ms 386 us and finishs at 65 ms 435 us
22: 2: 3 sends to 14: 0: 1 starts at 65 ms 435 us and finishs at 65 ms 447 us
18:35: 4 sends to 18:33: 2 starts at 65 ms 677 us and finishs at 65 ms 746 us
19:26:22 sends to 18:33: 2 starts at 65 ms 746 us and finishs at 65 ms 771 us
7:35: 4 sends to 7:29: 1 starts at 66 ms 873 us and finishs at 66 ms 891 us
7:35: 4 sends to 8:26:22 starts at 66 ms 891 us and finishs at 66 ms 960 us
16:21: 2 sends to 16:22: 2 starts at 67 ms 620 us and finishs at 67 ms 620 us
19:26:22 sends to 16:22: 2 starts at 67 ms 620 us and finishs at 67 ms 761 us
18:33: 2 sends to 18:35: 2 starts at 67 ms 761 us and finishs at 67 ms 778 us
18:35: 2 sends to 18:33: 2 starts at 68 ms 589 us and finishs at 68 ms 609 us
18:35: 2 sends to 19:26:22 starts at 68 ms 609 us and finishs at 68 ms 615 us
8:28: 1 sends to 6:22: 2 starts at 68 ms 809 us and finishs at 68 ms 812 us
8:28: 1 sends to 8:26:22 starts at 68 ms 812 us and finishs at 68 ms 823 us
14: 0: 1 sends to 22: 2: 3 starts at 68 ms 823 us and finishs at 68 ms 829 us
14: 0: 1 sends to 20: 2: 3 starts at 68 ms 829 us and finishs at 68 ms 835 us
16:17: 1 sends to 20: 2: 3 starts at 68 ms 835 us and finishs at 69 ms 283 us
16:22: 2 sends to 19:28: 1 starts at 69 ms 670 us and finishs at 69 ms 676 us
8:28: 1 sends to 5:21: 3 starts at 69 ms 676 us and finishs at 69 ms 679 us
17:17: 1 sends to 20: 2: 3 starts at 69 ms 679 us and finishs at 70 ms 127 us
18:17: 1 sends to 20: 2: 3 starts at 70 ms 127 us and finishs at 70 ms 575 us
19:17: 1 sends to 20: 2: 3 starts at 70 ms 575 us and finishs at 71 ms 23 us
7:29:10 sends to 7:35: 1 starts at 71 ms 677 us and finishs at 71 ms 698 us
7:29:10 sends to 7:35: 4 starts at 71 ms 698 us and finishs at 71 ms 715 us
7:29:10 sends to 8:26:22 starts at 71 ms 715 us and finishs at 71 ms 736 us
7:29:10 sends to 8:27: 1 starts at 71 ms 736 us and finishs at 71 ms 757 us
7:29:10 sends to 8:28: 1 starts at 71 ms 757 us and finishs at 71 ms 778 us
7:29:10 sends to 8:30:20 starts at 71 ms 778 us and finishs at 71 ms 842 us

8:30:20 sends to 7:31: 1 starts at 71 ms 842 us and finishs at 72 ms 12 us
7:29:10 sends to 5:19: 5 starts at 72 ms 12 us and finishs at 72 ms 33 us
7:29: 9 sends to 1: 0: 1 starts at 72 ms 33 us and finishs at 72 ms 38 us
8:18: 1 sends to 1: 0: 1 starts at 72 ms 38 us and finishs at 72 ms 42 us
9: 2: 3 sends to 1: 0: 1 starts at 72 ms 42 us and finishs at 72 ms 54 us
7:29:10 sends to 2: 0: 1 starts at 72 ms 54 us and finishs at 72 ms 55 us
8:16: 2 sends to 2: 0: 1 starts at 72 ms 55 us and finishs at 72 ms 56 us
8:18: 1 sends to 2: 0: 1 starts at 72 ms 56 us and finishs at 72 ms 60 us
9: 2: 3 sends to 2: 0: 1 starts at 72 ms 60 us and finishs at 72 ms 72 us
2: 0: 1 sends to 11: 2: 3 starts at 72 ms 72 us and finishs at 72 ms 78 us
2: 0: 1 sends to 8:16: 3 starts at 72 ms 78 us and finishs at 72 ms 95 us
2: 0: 1 sends to 8:30:20 starts at 72 ms 95 us and finishs at 72 ms 147 us
2: 0: 1 sends to 9: 2: 3 starts at 72 ms 147 us and finishs at 72 ms 153 us
7:29:10 sends to 3: 0: 1 starts at 72 ms 153 us and finishs at 72 ms 154 us
8:16: 2 sends to 3: 0: 1 starts at 72 ms 154 us and finishs at 72 ms 155 us
8:18: 1 sends to 3: 0: 1 starts at 72 ms 155 us and finishs at 72 ms 159 us
9: 2: 3 sends to 3: 0: 1 starts at 72 ms 159 us and finishs at 72 ms 171 us
3: 0: 1 sends to 11: 2: 3 starts at 72 ms 171 us and finishs at 72 ms 177 us
3: 0: 1 sends to 8:16: 3 starts at 72 ms 177 us and finishs at 72 ms 194 us
3: 0: 1 sends to 8:30:20 starts at 72 ms 194 us and finishs at 72 ms 246 us
3: 0: 1 sends to 9: 2: 3 starts at 72 ms 246 us and finishs at 72 ms 252 us
7:29:10 sends to 5:21: 5 starts at 72 ms 252 us and finishs at 72 ms 273 us
8:16: 2 sends to 5:21: 5 starts at 72 ms 273 us and finishs at 72 ms 274 us
8:26:22 sends to 5:21: 5 starts at 72 ms 274 us and finishs at 72 ms 288 us
5:21: 2 sends to 5:22: 2 starts at 72 ms 488 us and finishs at 72 ms 488 us
6:22: 2 sends to 5:22: 2 starts at 72 ms 488 us and finishs at 72 ms 491 us
8:28: 1 sends to 5:22: 2 starts at 72 ms 491 us and finishs at 72 ms 494 us
5:21: 3 sends to 6:22: 2 starts at 72 ms 539 us and finishs at 72 ms 547 us
5:21: 3 sends to 8:28: 1 starts at 72 ms 547 us and finishs at 72 ms 554 us
6:21: 4 sends to 6:22: 3 starts at 72 ms 774 us and finishs at 72 ms 774 us
8:26:22 sends to 6:22: 3 starts at 72 ms 774 us and finishs at 72 ms 779 us
8:27: 1 sends to 6:22: 3 starts at 72 ms 779 us and finishs at 72 ms 784 us
8:28: 1 sends to 6:22: 3 starts at 72 ms 784 us and finishs at 72 ms 789 us
7:31: 1 sends to 8:30:20 starts at 73 ms 392 us and finishs at 73 ms 399 us
1: 0: 1 sends to 11: 2: 3 starts at 73 ms 399 us and finishs at 73 ms 405 us
1: 0: 1 sends to 7:29: 9 starts at 73 ms 405 us and finishs at 73 ms 789 us
1: 0: 1 sends to 8:30:20 starts at 73 ms 789 us and finishs at 73 ms 841 us
1: 0: 1 sends to 9: 2: 3 starts at 73 ms 841 us and finishs at 73 ms 847 us
1: 0: 1 sends to 8:16: 3 starts at 73 ms 847 us and finishs at 73 ms 864 us
4: 0: 1 sends to 8:16: 3 starts at 73 ms 864 us and finishs at 73 ms 881 us
6:16: 2 sends to 8:16: 3 starts at 73 ms 881 us and finishs at 73 ms 903 us
1: 0: 1 sends to 6:16: 3 starts at 73 ms 903 us and finishs at 73 ms 920 us
2: 0: 1 sends to 6:16: 3 starts at 73 ms 920 us and finishs at 73 ms 937 us
3: 0: 1 sends to 6:16: 3 starts at 73 ms 937 us and finishs at 73 ms 954 us
4: 0: 1 sends to 6:16: 3 starts at 73 ms 954 us and finishs at 73 ms 971 us
5:16: 2 sends to 6:16: 3 starts at 73 ms 971 us and finishs at 73 ms 993 us
7:16: 2 sends to 6:16: 3 starts at 73 ms 993 us and finishs at 74 ms 15 us
8:16: 2 sends to 6:16: 3 starts at 74 ms 15 us and finishs at 74 ms 37 us
1: 0: 1 sends to 5:16: 3 starts at 74 ms 37 us and finishs at 74 ms 54 us
7:20: 4 sends to 6:22: 4 starts at 74 ms 68 us and finishs at 74 ms 78 us
7:20: 4 sends to 8:30:20 starts at 74 ms 78 us and finishs at 74 ms 111 us

7:20: 4 sends to 8:20: 4 starts at 74 ms 111 us and finishs at 74 ms 114 us
7:29:10 sends to 8:20: 4 starts at 74 ms 114 us and finishs at 74 ms 135 us
2: 0: 1 sends to 5:16: 3 starts at 74 ms 135 us and finishs at 74 ms 152 us
3: 0: 1 sends to 5:16: 3 starts at 74 ms 152 us and finishs at 74 ms 169 us
4: 0: 1 sends to 5:16: 3 starts at 74 ms 169 us and finishs at 74 ms 186 us
6:16: 2 sends to 5:16: 3 starts at 74 ms 186 us and finishs at 74 ms 208 us
7:16: 2 sends to 5:16: 3 starts at 74 ms 208 us and finishs at 74 ms 230 us
8:16: 2 sends to 5:16: 3 starts at 74 ms 230 us and finishs at 74 ms 252 us
2: 0: 1 sends to 7:16: 5 starts at 74 ms 252 us and finishs at 74 ms 289 us
3: 0: 1 sends to 7:16: 5 starts at 74 ms 289 us and finishs at 74 ms 326 us
4: 0: 1 sends to 7:16: 5 starts at 74 ms 326 us and finishs at 74 ms 363 us
1: 0: 1 sends to 7:16: 3 starts at 74 ms 363 us and finishs at 74 ms 380 us
2: 0: 1 sends to 7:16: 3 starts at 74 ms 380 us and finishs at 74 ms 397 us
3: 0: 1 sends to 7:16: 3 starts at 74 ms 397 us and finishs at 74 ms 414 us
4: 0: 1 sends to 7:16: 3 starts at 74 ms 414 us and finishs at 74 ms 431 us
5:16: 2 sends to 7:16: 3 starts at 74 ms 431 us and finishs at 74 ms 453 us
6:16: 2 sends to 7:16: 3 starts at 74 ms 453 us and finishs at 74 ms 475 us
5:21: 4 sends to 8:30:20 starts at 74 ms 490 us and finishs at 74 ms 509 us
6:21: 3 sends to 5:22: 3 starts at 74 ms 509 us and finishs at 74 ms 515 us
8:26:22 sends to 5:22: 3 starts at 74 ms 515 us and finishs at 74 ms 520 us
8:27: 1 sends to 5:22: 3 starts at 74 ms 520 us and finishs at 74 ms 525 us
8:28: 1 sends to 5:22: 3 starts at 74 ms 525 us and finishs at 74 ms 530 us
5:21: 4 sends to 6:22: 4 starts at 74 ms 530 us and finishs at 74 ms 533 us
5:22: 2 sends to 6:22: 4 starts at 74 ms 533 us and finishs at 74 ms 569 us
6:19: 1 sends to 6:22: 4 starts at 74 ms 569 us and finishs at 74 ms 581 us
5:22: 2 sends to 6:22: 2 starts at 74 ms 589 us and finishs at 74 ms 592 us
5:22: 2 sends to 8:28: 1 starts at 74 ms 592 us and finishs at 74 ms 598 us
6:19: 4 sends to 6:22: 4 starts at 74 ms 598 us and finishs at 74 ms 612 us
6:21: 4 sends to 6:22: 4 starts at 74 ms 612 us and finishs at 74 ms 615 us
7:29:10 sends to 6:22: 4 starts at 74 ms 615 us and finishs at 74 ms 620 us
7:35: 1 sends to 6:22: 4 starts at 74 ms 620 us and finishs at 74 ms 626 us
7:35: 2 sends to 6:22: 4 starts at 74 ms 626 us and finishs at 74 ms 632 us
7:35: 4 sends to 6:22: 4 starts at 74 ms 632 us and finishs at 74 ms 650 us
8:26:22 sends to 6:22: 4 starts at 74 ms 650 us and finishs at 74 ms 664 us
5:19: 5 sends to 6:22: 4 starts at 74 ms 664 us and finishs at 74 ms 674 us
5:19: 5 sends to 7:29:10 starts at 74 ms 674 us and finishs at 74 ms 684 us
5:19: 5 sends to 8:30:20 starts at 74 ms 684 us and finishs at 74 ms 696 us
5:19: 5 sends to 6:19: 5 starts at 74 ms 696 us and finishs at 74 ms 699 us
6:16: 2 sends to 6:19: 5 starts at 74 ms 699 us and finishs at 74 ms 700 us
7:29:10 sends to 6:19: 5 starts at 74 ms 700 us and finishs at 74 ms 721 us
20: 2: 3 sends to 12: 0: 1 starts at 74 ms 721 us and finishs at 74 ms 733 us
20: 2: 3 sends to 14: 0: 1 starts at 74 ms 733 us and finishs at 74 ms 745 us
20: 2: 3 sends to 16:17: 1 starts at 74 ms 745 us and finishs at 74 ms 794 us
6:19: 5 sends to 5:19: 5 starts at 74 ms 801 us and finishs at 74 ms 804 us
6:19: 5 sends to 6:22: 4 starts at 74 ms 804 us and finishs at 74 ms 814 us
15: 0: 3 sends to 18:29: 1 starts at 74 ms 814 us and finishs at 74 ms 815 us
12: 0: 3 sends to 18:29: 1 starts at 74 ms 815 us and finishs at 74 ms 816 us
13: 0: 3 sends to 18:29: 1 starts at 74 ms 819 us and finishs at 74 ms 820 us
14: 0: 3 sends to 18:29: 1 starts at 74 ms 820 us and finishs at 74 ms 821 us
6:19: 5 sends to 7:29:10 starts at 74 ms 821 us and finishs at 74 ms 831 us
6:19: 5 sends to 8:30:20 starts at 74 ms 831 us and finishs at 74 ms 843 us

20: 2: 3 sends to 17:17: 1 starts at 74 ms 843 us and finishs at 74 ms 892 us
20: 2: 3 sends to 18:17: 1 starts at 74 ms 892 us and finishs at 74 ms 941 us
20: 2: 3 sends to 19:17: 1 starts at 74 ms 941 us and finishs at 74 ms 990 us
5:19: 5 sends to 5:22: 4 starts at 74 ms 990 us and finishs at 75 ms 0 us
5:21: 4 sends to 5:22: 4 starts at 75 ms 0 us and finishs at 75 ms 3 us
2: 0: 1 sends to 5:16: 5 starts at 75 ms 3 us and finishs at 75 ms 40 us
3: 0: 1 sends to 5:16: 5 starts at 75 ms 40 us and finishs at 75 ms 77 us
4: 0: 1 sends to 5:16: 5 starts at 75 ms 77 us and finishs at 75 ms 114 us
1: 0: 1 sends to 6:16: 5 starts at 75 ms 114 us and finishs at 75 ms 151 us
4: 0: 1 sends to 6:16: 5 starts at 75 ms 151 us and finishs at 75 ms 188 us
1: 0: 1 sends to 8:16: 5 starts at 75 ms 188 us and finishs at 75 ms 225 us
2: 0: 1 sends to 8:16: 5 starts at 75 ms 225 us and finishs at 75 ms 262 us
3: 0: 1 sends to 8:16: 5 starts at 75 ms 262 us and finishs at 75 ms 299 us
4: 0: 1 sends to 8:16: 5 starts at 75 ms 299 us and finishs at 75 ms 336 us
1: 0: 3 sends to 7:29: 1 starts at 75 ms 996 us and finishs at 75 ms 997 us
4: 0: 3 sends to 7:29: 1 starts at 75 ms 997 us and finishs at 75 ms 998 us
19:28: 1 sends to 16:22: 2 starts at 76 ms 221 us and finishs at 76 ms 224 us
5:21: 3 sends to 6:21: 3 starts at 76 ms 224 us and finishs at 76 ms 229 us
8:26:22 sends to 6:21: 3 starts at 76 ms 229 us and finishs at 76 ms 232 us
8:28: 1 sends to 6:21: 3 starts at 76 ms 232 us and finishs at 76 ms 235 us
5:21: 3 sends to 8:27: 1 starts at 77 ms 269 us and finishs at 77 ms 276 us
8:26:22 sends to 8:27: 1 starts at 77 ms 276 us and finishs at 77 ms 291 us
6:21: 3 sends to 5:21: 3 starts at 79 ms 32 us and finishs at 79 ms 37 us
6:21: 3 sends to 5:22: 2 starts at 79 ms 37 us and finishs at 79 ms 45 us
6:21: 3 sends to 8:26:22 starts at 79 ms 45 us and finishs at 79 ms 60 us
6:21: 3 sends to 8:27: 1 starts at 79 ms 60 us and finishs at 79 ms 67 us
6:21: 3 sends to 8:28: 1 starts at 79 ms 67 us and finishs at 79 ms 74 us
6:21: 3 sends to 8:30:20 starts at 79 ms 74 us and finishs at 79 ms 111 us
8:20: 4 sends to 6:22: 4 starts at 79 ms 125 us and finishs at 79 ms 135 us
8:20: 4 sends to 7:20: 4 starts at 79 ms 135 us and finishs at 79 ms 138 us
8:20: 4 sends to 7:29:10 starts at 79 ms 138 us and finishs at 79 ms 148 us
8:20: 4 sends to 8:30:20 starts at 79 ms 148 us and finishs at 79 ms 181 us
16:22: 2 sends to 16:21: 3 starts at 79 ms 964 us and finishs at 79 ms 964 us
19:26:22 sends to 16:21: 3 starts at 79 ms 964 us and finishs at 79 ms 967 us
19:28: 1 sends to 16:21: 3 starts at 79 ms 967 us and finishs at 79 ms 970 us
7:29: 2 sends to 7:29:10 starts at 80 ms 369 us and finishs at 80 ms 378 us
8:27: 1 sends to 8:26:22 starts at 80 ms 826 us and finishs at 80 ms 831 us
8:27: 1 sends to 8:30:20 starts at 80 ms 831 us and finishs at 80 ms 840 us
18:29: 2 sends to 18:29:10 starts at 81 ms 613 us and finishs at 81 ms 622 us
16:16: 2 sends to 12: 0: 1 starts at 82 ms 740 us and finishs at 82 ms 741 us
16:16: 2 sends to 14: 0: 1 starts at 82 ms 741 us and finishs at 82 ms 742 us
16:16: 2 sends to 16:19: 5 starts at 82 ms 742 us and finishs at 82 ms 743 us
16:16: 2 sends to 16:21: 5 starts at 82 ms 743 us and finishs at 82 ms 744 us
16:16: 2 sends to 16:22: 4 starts at 82 ms 744 us and finishs at 82 ms 751 us
16:16: 2 sends to 17:16: 1 starts at 82 ms 751 us and finishs at 82 ms 752 us
16:16: 2 sends to 17:16: 2 starts at 82 ms 752 us and finishs at 82 ms 765 us
16:16: 2 sends to 17:19: 5 starts at 82 ms 765 us and finishs at 82 ms 766 us
16:16: 2 sends to 18:20: 4 starts at 82 ms 766 us and finishs at 82 ms 767 us
16:21: 3 sends to 16:22: 2 starts at 82 ms 767 us and finishs at 82 ms 779 us
16:21: 3 sends to 19:26:22 starts at 82 ms 779 us and finishs at 82 ms 794 us
16:21: 3 sends to 19:28: 1 starts at 82 ms 794 us and finishs at 82 ms 801 us

5:22: 2 sends to 7:33: 2 starts at 82 ms 801 us and finishs at 82 ms 805 us
16:21: 3 sends to 19:27: 1 starts at 82 ms 805 us and finishs at 82 ms 812 us
19:26:22 sends to 19:27: 1 starts at 82 ms 812 us and finishs at 82 ms 827 us
6:22: 2 sends to 7:33: 2 starts at 82 ms 827 us and finishs at 82 ms 831 us
7:35: 1 sends to 7:33: 2 starts at 82 ms 831 us and finishs at 82 ms 837 us
7:35: 4 sends to 7:33: 2 starts at 82 ms 837 us and finishs at 82 ms 906 us
8:26:22 sends to 7:33: 2 starts at 82 ms 906 us and finishs at 82 ms 931 us
16:16: 2 sends to 18:32: 1 starts at 82 ms 931 us and finishs at 82 ms 932 us
16:16: 2 sends to 18:35: 1 starts at 82 ms 932 us and finishs at 82 ms 933 us
16:16: 2 sends to 19:16: 1 starts at 82 ms 933 us and finishs at 82 ms 934 us
16:16: 2 sends to 19:16: 2 starts at 82 ms 934 us and finishs at 82 ms 947 us
16:16: 2 sends to 19:26:22 starts at 82 ms 947 us and finishs at 82 ms 948 us
16:16: 2 sends to 19:28: 1 starts at 82 ms 948 us and finishs at 82 ms 949 us
16:16: 2 sends to 17:21: 5 starts at 82 ms 949 us and finishs at 82 ms 950 us
16:16: 2 sends to 13: 0: 1 starts at 82 ms 950 us and finishs at 82 ms 951 us
19:16: 2 sends to 13: 0: 1 starts at 82 ms 951 us and finishs at 82 ms 952 us
19:18: 1 sends to 13: 0: 1 starts at 82 ms 952 us and finishs at 82 ms 956 us
19:27: 1 sends to 13: 0: 1 starts at 82 ms 956 us and finishs at 82 ms 957 us
20: 2: 3 sends to 13: 0: 1 starts at 82 ms 957 us and finishs at 82 ms 969 us
22: 2: 3 sends to 13: 0: 1 starts at 82 ms 969 us and finishs at 82 ms 981 us
13: 0: 1 sends to 20: 2: 3 starts at 82 ms 981 us and finishs at 82 ms 987 us
13: 0: 1 sends to 22: 2: 3 starts at 82 ms 987 us and finishs at 82 ms 993 us
17:16: 2 sends to 18:16: 1 starts at 82 ms 993 us and finishs at 82 ms 994 us
18:29: 1 sends to 18:16: 1 starts at 82 ms 994 us and finishs at 83 ms 63 us
19:16: 2 sends to 18:16: 1 starts at 83 ms 63 us and finishs at 83 ms 64 us
16:16: 2 sends to 18:16: 2 starts at 83 ms 64 us and finishs at 83 ms 77 us
19:16: 2 sends to 18:16: 2 starts at 83 ms 77 us and finishs at 83 ms 89 us
16:16: 2 sends to 15: 0: 1 starts at 83 ms 89 us and finishs at 83 ms 90 us
17:16: 2 sends to 15: 0: 1 starts at 83 ms 90 us and finishs at 83 ms 91 us
13: 0: 1 sends to 16:16: 5 starts at 83 ms 91 us and finishs at 83 ms 128 us
14: 0: 1 sends to 16:16: 5 starts at 83 ms 128 us and finishs at 83 ms 165 us
14: 0: 1 sends to 18:16: 3 starts at 83 ms 165 us and finishs at 83 ms 182 us
13: 0: 1 sends to 18:16: 5 starts at 83 ms 182 us and finishs at 83 ms 219 us
13: 0: 1 sends to 17:16: 3 starts at 83 ms 219 us and finishs at 83 ms 236 us
14: 0: 1 sends to 17:16: 3 starts at 83 ms 236 us and finishs at 83 ms 253 us
14: 0: 1 sends to 19:16: 3 starts at 83 ms 253 us and finishs at 83 ms 270 us
13: 0: 1 sends to 18:29: 9 starts at 83 ms 270 us and finishs at 83 ms 654 us
14: 0: 1 sends to 18:29: 9 starts at 83 ms 654 us and finishs at 84 ms 38 us
16:16: 2 sends to 17:22: 4 starts at 84 ms 38 us and finishs at 84 ms 45 us
16:19: 1 sends to 17:22: 4 starts at 84 ms 45 us and finishs at 84 ms 57 us
13: 0: 1 sends to 17:16: 5 starts at 84 ms 57 us and finishs at 84 ms 94 us
13: 0: 1 sends to 16:16: 3 starts at 84 ms 94 us and finishs at 84 ms 111 us
14: 0: 1 sends to 16:16: 3 starts at 84 ms 111 us and finishs at 84 ms 128 us
13: 0: 1 sends to 19:16: 5 starts at 84 ms 128 us and finishs at 84 ms 165 us
14: 0: 1 sends to 19:16: 5 starts at 84 ms 165 us and finishs at 84 ms 202 us
18:16: 2 sends to 14: 0: 1 starts at 84 ms 204 us and finishs at 84 ms 205 us
18:16: 2 sends to 16:16: 1 starts at 84 ms 205 us and finishs at 84 ms 206 us
18:16: 2 sends to 16:16: 2 starts at 84 ms 206 us and finishs at 84 ms 219 us
18:16: 2 sends to 16:19: 5 starts at 84 ms 219 us and finishs at 84 ms 220 us
18:16: 2 sends to 16:21: 5 starts at 84 ms 220 us and finishs at 84 ms 221 us
18:16: 2 sends to 16:22: 4 starts at 84 ms 221 us and finishs at 84 ms 228 us

18:16: 2 sends to 17:16: 1 starts at 84 ms 228 us and finishs at 84 ms 229 us
18:16: 2 sends to 17:19: 5 starts at 84 ms 229 us and finishs at 84 ms 230 us
18:16: 2 sends to 17:21: 5 starts at 84 ms 230 us and finishs at 84 ms 231 us
18:16: 2 sends to 18:20: 4 starts at 84 ms 231 us and finishs at 84 ms 232 us
18:16: 2 sends to 18:29: 2 starts at 84 ms 232 us and finishs at 84 ms 233 us
18:16: 2 sends to 18:31: 1 starts at 84 ms 233 us and finishs at 84 ms 234 us
18:16: 2 sends to 18:32: 1 starts at 84 ms 234 us and finishs at 84 ms 235 us
18:16: 2 sends to 18:33: 2 starts at 84 ms 235 us and finishs at 84 ms 237 us
18:16: 2 sends to 18:35: 1 starts at 84 ms 237 us and finishs at 84 ms 238 us
18:16: 2 sends to 19:16: 2 starts at 84 ms 238 us and finishs at 84 ms 251 us
18:16: 2 sends to 19:20: 4 starts at 84 ms 251 us and finishs at 84 ms 252 us
18:16: 2 sends to 19:26:22 starts at 84 ms 252 us and finishs at 84 ms 253 us
18:16: 2 sends to 19:27: 1 starts at 84 ms 253 us and finishs at 84 ms 254 us
18:16: 2 sends to 19:28: 1 starts at 84 ms 254 us and finishs at 84 ms 255 us
18:16: 2 sends to 15: 0: 1 starts at 84 ms 255 us and finishs at 84 ms 256 us
19:16: 2 sends to 15: 0: 1 starts at 84 ms 256 us and finishs at 84 ms 257 us
22: 2: 3 sends to 15: 0: 1 starts at 84 ms 257 us and finishs at 84 ms 269 us
7:33: 2 sends to 5:22: 2 starts at 84 ms 849 us and finishs at 84 ms 917 us
7:33: 2 sends to 6:22: 2 starts at 84 ms 917 us and finishs at 84 ms 985 us
7:33: 2 sends to 8:30:20 starts at 84 ms 985 us and finishs at 84 ms 989 us
7:33: 2 sends to 7:35: 2 starts at 84 ms 989 us and finishs at 85 ms 6 us
15: 0: 1 sends to 22: 2: 3 starts at 85 ms 6 us and finishs at 85 ms 12 us
15: 0: 1 sends to 18:16: 3 starts at 85 ms 12 us and finishs at 85 ms 29 us
16:16: 2 sends to 18:16: 3 starts at 85 ms 29 us and finishs at 85 ms 51 us
19:16: 2 sends to 18:16: 3 starts at 85 ms 51 us and finishs at 85 ms 73 us
15: 0: 1 sends to 18:16: 5 starts at 85 ms 73 us and finishs at 85 ms 110 us
16:16: 2 sends to 17:16: 3 starts at 85 ms 110 us and finishs at 85 ms 132 us
18:16: 2 sends to 17:16: 3 starts at 85 ms 132 us and finishs at 85 ms 154 us
19:16: 2 sends to 17:16: 3 starts at 85 ms 154 us and finishs at 85 ms 176 us
15: 0: 1 sends to 19:16: 3 starts at 85 ms 176 us and finishs at 85 ms 193 us
16:16: 2 sends to 19:16: 3 starts at 85 ms 193 us and finishs at 85 ms 215 us
15: 0: 1 sends to 18:29: 9 starts at 85 ms 215 us and finishs at 85 ms 599 us
16:17: 1 sends to 18:29: 9 starts at 85 ms 599 us and finishs at 85 ms 694 us
17:17: 1 sends to 18:29: 9 starts at 85 ms 694 us and finishs at 85 ms 789 us
15: 0: 1 sends to 16:16: 3 starts at 85 ms 789 us and finishs at 85 ms 806 us
7:35: 2 sends to 7:33: 2 starts at 85 ms 817 us and finishs at 85 ms 837 us
7:35: 2 sends to 8:26:22 starts at 85 ms 837 us and finishs at 85 ms 843 us
19:17: 1 sends to 18:29: 9 starts at 85 ms 843 us and finishs at 85 ms 938 us
17:16: 2 sends to 16:16: 3 starts at 85 ms 938 us and finishs at 85 ms 960 us
7:32: 3 sends to 6:22: 4 starts at 85 ms 970 us and finishs at 85 ms 976 us
7:32: 3 sends to 8:30:20 starts at 85 ms 976 us and finishs at 85 ms 986 us
15: 0: 1 sends to 17:16: 5 starts at 85 ms 986 us and finishs at 86 ms 23 us
18:16: 2 sends to 16:16: 3 starts at 86 ms 23 us and finishs at 86 ms 45 us
15: 0: 1 sends to 19:16: 5 starts at 86 ms 45 us and finishs at 86 ms 82 us
19:27: 1 sends to 19:26:22 starts at 86 ms 362 us and finishs at 86 ms 367 us
18:16: 1 sends to 18:29: 1 starts at 86 ms 926 us and finishs at 86 ms 926 us
17:21: 2 sends to 17:22: 2 starts at 88 ms 462 us and finishs at 88 ms 462 us
16:21: 3 sends to 17:22: 2 starts at 88 ms 462 us and finishs at 88 ms 470 us
19:26:22 sends to 17:22: 2 starts at 88 ms 470 us and finishs at 88 ms 611 us
19:28: 1 sends to 17:22: 2 starts at 88 ms 611 us and finishs at 88 ms 614 us
7:16: 5 sends to 2: 0: 1 starts at 89 ms 11 us and finishs at 89 ms 17 us

7:16: 5 sends to 3: 0: 1 starts at 89 ms 17 us and finishs at 89 ms 23 us
7:16: 5 sends to 4: 0: 1 starts at 89 ms 23 us and finishs at 89 ms 29 us
5:19: 4 sends to 8:30:20 starts at 89 ms 642 us and finishs at 89 ms 684 us
5:21: 2 sends to 8:30:20 starts at 89 ms 684 us and finishs at 89 ms 686 us
5:21: 3 sends to 8:30:20 starts at 89 ms 686 us and finishs at 89 ms 723 us
5:22: 2 sends to 8:30:20 starts at 89 ms 723 us and finishs at 89 ms 728 us
6:21: 2 sends to 8:30:20 starts at 89 ms 728 us and finishs at 89 ms 730 us
6:22: 2 sends to 8:30:20 starts at 89 ms 730 us and finishs at 89 ms 735 us
7:29: 1 sends to 8:30:20 starts at 89 ms 735 us and finishs at 89 ms 750 us
8:26:22 sends to 8:30:20 starts at 89 ms 750 us and finishs at 89 ms 813 us
8:28: 1 sends to 8:30:20 starts at 89 ms 813 us and finishs at 89 ms 839 us
16:21: 3 sends to 17:21: 3 starts at 90 ms 521 us and finishs at 90 ms 526 us
19:26:22 sends to 17:21: 3 starts at 90 ms 526 us and finishs at 90 ms 529 us
19:28: 1 sends to 17:21: 3 starts at 90 ms 529 us and finishs at 90 ms 532 us
17:22: 2 sends to 19:28: 1 starts at 90 ms 661 us and finishs at 90 ms 667 us
5:16: 5 sends to 2: 0: 1 starts at 92 ms 52 us and finishs at 92 ms 58 us
5:16: 5 sends to 3: 0: 1 starts at 92 ms 58 us and finishs at 92 ms 64 us
5:16: 5 sends to 4: 0: 1 starts at 92 ms 64 us and finishs at 92 ms 70 us
5:16: 5 sends to 6:16: 5 starts at 92 ms 70 us and finishs at 92 ms 110 us
7:16: 5 sends to 6:16: 5 starts at 92 ms 110 us and finishs at 92 ms 150 us
7:29:10 sends to 6:16: 5 starts at 92 ms 150 us and finishs at 92 ms 171 us
5:16: 5 sends to 8:16: 5 starts at 92 ms 171 us and finishs at 92 ms 211 us
17:21: 3 sends to 16:21: 3 starts at 93 ms 461 us and finishs at 93 ms 466 us
17:21: 3 sends to 19:26:22 starts at 93 ms 466 us and finishs at 93 ms 481 us
17:21: 3 sends to 19:28: 1 starts at 93 ms 481 us and finishs at 93 ms 488 us
5:21: 5 sends to 7:29:10 starts at 95 ms 427 us and finishs at 95 ms 436 us
5:21: 5 sends to 7:32: 1 starts at 95 ms 436 us and finishs at 95 ms 453 us
5:21: 5 sends to 7:33: 2 starts at 95 ms 453 us and finishs at 95 ms 470 us
5:21: 5 sends to 8:26:22 starts at 95 ms 470 us and finishs at 95 ms 474 us
5:21: 5 sends to 8:30:20 starts at 95 ms 474 us and finishs at 95 ms 476 us
6:16: 2 sends to 5:22: 4 starts at 95 ms 476 us and finishs at 95 ms 483 us
6:19: 1 sends to 5:22: 4 starts at 95 ms 483 us and finishs at 95 ms 495 us
6:19: 4 sends to 5:22: 4 starts at 95 ms 495 us and finishs at 95 ms 509 us
6:19: 5 sends to 5:22: 4 starts at 95 ms 509 us and finishs at 95 ms 519 us
6:21: 4 sends to 5:22: 4 starts at 95 ms 519 us and finishs at 95 ms 522 us
6:22: 2 sends to 5:22: 4 starts at 95 ms 522 us and finishs at 95 ms 558 us
5:21: 5 sends to 6:21: 5 starts at 95 ms 558 us and finishs at 95 ms 594 us
18:29:10 sends to 18:33: 2 starts at 95 ms 682 us and finishs at 95 ms 727 us
18:29:10 sends to 19:27: 1 starts at 95 ms 727 us and finishs at 95 ms 748 us
18:29:10 sends to 16:19: 5 starts at 95 ms 748 us and finishs at 95 ms 769 us
18:29:10 sends to 16:21: 5 starts at 95 ms 769 us and finishs at 95 ms 790 us
19:26:22 sends to 16:21: 5 starts at 95 ms 790 us and finishs at 95 ms 804 us
19:28: 1 sends to 16:21: 5 starts at 95 ms 804 us and finishs at 95 ms 809 us
18:29:10 sends to 18:31: 1 starts at 95 ms 809 us and finishs at 95 ms 822 us
18:29:10 sends to 19:20: 4 starts at 95 ms 822 us and finishs at 95 ms 843 us
19:16: 2 sends to 19:20: 4 starts at 95 ms 843 us and finishs at 95 ms 844 us
19:16: 2 sends to 18:20: 4 starts at 95 ms 844 us and finishs at 95 ms 845 us
16:19: 5 sends to 18:29:10 starts at 95 ms 849 us and finishs at 95 ms 859 us
16:19: 5 sends to 17:19: 5 starts at 95 ms 859 us and finishs at 95 ms 862 us
16:19: 5 sends to 16:22: 4 starts at 95 ms 862 us and finishs at 95 ms 872 us
16:21: 4 sends to 16:22: 4 starts at 95 ms 872 us and finishs at 95 ms 875 us

18:29:10 sends to 16:16: 5 starts at 95 ms 875 us and finishs at 95 ms 896 us
16:19: 5 sends to 17:22: 4 starts at 95 ms 896 us and finishs at 95 ms 906 us
16:21: 4 sends to 17:22: 4 starts at 95 ms 906 us and finishs at 95 ms 909 us
16:22: 2 sends to 17:22: 4 starts at 95 ms 909 us and finishs at 95 ms 945 us
17:19: 5 sends to 16:19: 5 starts at 95 ms 960 us and finishs at 95 ms 963 us
19:20: 4 sends to 18:29:10 starts at 96 ms 519 us and finishs at 96 ms 529 us
19:20: 4 sends to 18:20: 4 starts at 96 ms 529 us and finishs at 96 ms 532 us
18:20: 4 sends to 19:20: 4 starts at 97 ms 205 us and finishs at 97 ms 208 us
18:32: 3 sends to 16:22: 4 starts at 98 ms 326 us and finishs at 98 ms 332 us
18:32: 3 sends to 17:22: 4 starts at 98 ms 332 us and finishs at 98 ms 338 us
6:16: 5 sends to 1: 0: 1 starts at 98 ms 890 us and finishs at 98 ms 896 us
6:16: 5 sends to 4: 0: 1 starts at 98 ms 896 us and finishs at 98 ms 902 us
6:16: 5 sends to 7:29:10 starts at 98 ms 902 us and finishs at 98 ms 912 us
16:16: 5 sends to 12: 0: 1 starts at 98 ms 916 us and finishs at 98 ms 922 us
16:16: 5 sends to 13: 0: 1 starts at 98 ms 922 us and finishs at 98 ms 928 us
16:16: 5 sends to 14: 0: 1 starts at 98 ms 928 us and finishs at 98 ms 934 us
16:16: 5 sends to 18:29:10 starts at 98 ms 934 us and finishs at 98 ms 944 us
16:16: 5 sends to 18:16: 5 starts at 98 ms 944 us and finishs at 98 ms 984 us
18:29:10 sends to 18:16: 5 starts at 98 ms 984 us and finishs at 99 ms 5 us
16:16: 5 sends to 17:16: 5 starts at 99 ms 5 us and finishs at 99 ms 45 us
6:16: 5 sends to 5:16: 5 starts at 99 ms 45 us and finishs at 99 ms 85 us
6:16: 5 sends to 7:16: 5 starts at 99 ms 85 us and finishs at 99 ms 125 us
6:16: 5 sends to 8:16: 5 starts at 99 ms 125 us and finishs at 99 ms 165 us
7:16: 5 sends to 8:16: 5 starts at 99 ms 165 us and finishs at 99 ms 205 us
7:29:10 sends to 8:16: 5 starts at 99 ms 205 us and finishs at 99 ms 226 us
5:22: 3 sends to 7:29:10 starts at 99 ms 265 us and finishs at 99 ms 299 us
5:22: 3 sends to 8:30:20 starts at 99 ms 299 us and finishs at 99 ms 306 us
16:16: 5 sends to 19:16: 5 starts at 99 ms 306 us and finishs at 99 ms 346 us
1: 0: 2 sends to 7:29: 1 starts at 99 ms 750 us and finishs at 99 ms 752 us
15: 0: 3 sends to 18:29: 1 starts at 99 ms 752 us and finishs at 99 ms 753 us
12: 0: 3 sends to 18:29: 1 starts at 99 ms 753 us and finishs at 99 ms 754 us
3: 0: 2 sends to 7:29: 1 starts at 99 ms 754 us and finishs at 99 ms 756 us
4: 0: 2 sends to 7:29: 1 starts at 99 ms 756 us and finishs at 99 ms 758 us
13: 0: 3 sends to 18:29: 1 starts at 99 ms 758 us and finishs at 99 ms 759 us
14: 0: 3 sends to 18:29: 1 starts at 99 ms 759 us and finishs at 99 ms 760 us
14: 0: 2 sends to 18:29: 1 starts at 99 ms 760 us and finishs at 99 ms 763 us
15: 0: 2 sends to 18:29: 1 starts at 99 ms 763 us and finishs at 99 ms 766 us
2: 0: 2 sends to 7:29: 1 starts at 99 ms 766 us and finishs at 99 ms 769 us
12: 0: 2 sends to 18:29: 1 starts at 99 ms 769 us and finishs at 99 ms 770 us
13: 0: 2 sends to 18:29: 1 starts at 99 ms 770 us and finishs at 99 ms 772 us
1: 0: 3 sends to 7:29: 1 starts at 100 ms 935 us and finishs at 100 ms 936 us
4: 0: 3 sends to 7:29: 1 starts at 100 ms 936 us and finishs at 100 ms 937 us
18:35: 1 sends to 18:29: 1 starts at 100 ms 981 us and finishs at 100 ms 987 us
18:35: 1 sends to 18:33: 2 starts at 100 ms 987 us and finishs at 100 ms 993 us
18:29: 1 sends to 18:35: 4 starts at 102 ms 205 us and finishs at 102 ms 274 us
7:17: 1 sends to 8:30:20 starts at 102 ms 274 us and finishs at 102 ms 628 us
1: 0: 3 sends to 8:26:22 starts at 103 ms 385 us and finishs at 103 ms 387 us
4: 0: 3 sends to 8:26:22 starts at 103 ms 387 us and finishs at 103 ms 389 us
5:19: 3 sends to 8:30:20 starts at 104 ms 111 us and finishs at 104 ms 166 us
18:35: 4 sends to 18:29: 1 starts at 104 ms 305 us and finishs at 104 ms 323 us
6:21: 2 sends to 6:22: 2 starts at 104 ms 901 us and finishs at 104 ms 901 us

8:20: 2 sends to 8:30:20 starts at 104 ms 912 us and finishs at 104 ms 960 us
12: 0: 3 sends to 19:26:22 starts at 106 ms 953 us and finishs at 106 ms 955 us
13: 0: 3 sends to 19:26:22 starts at 106 ms 955 us and finishs at 106 ms 957 us
18:16: 5 sends to 12: 0: 1 starts at 107 ms 953 us and finishs at 107 ms 959 us
18:16: 5 sends to 13: 0: 1 starts at 107 ms 959 us and finishs at 107 ms 965 us
18:16: 5 sends to 15: 0: 1 starts at 107 ms 965 us and finishs at 107 ms 971 us
18:16: 5 sends to 16:16: 5 starts at 107 ms 971 us and finishs at 108 ms 11 us
18:16: 5 sends to 18:29:10 starts at 108 ms 11 us and finishs at 108 ms 21 us
18:29:10 sends to 17:16: 5 starts at 108 ms 21 us and finishs at 108 ms 42 us
7:35: 1 sends to 7:29: 1 starts at 108 ms 195 us and finishs at 108 ms 201 us
7:35: 1 sends to 8:26:22 starts at 108 ms 201 us and finishs at 108 ms 207 us
7:17: 1 sends to 6:17: 1 starts at 110 ms 93 us and finishs at 110 ms 99 us
17:16: 5 sends to 12: 0: 1 starts at 111 ms 62 us and finishs at 111 ms 68 us
17:16: 5 sends to 13: 0: 1 starts at 111 ms 68 us and finishs at 111 ms 74 us
6:17: 1 sends to 7:17: 1 starts at 111 ms 76 us and finishs at 111 ms 82 us
17:16: 5 sends to 15: 0: 1 starts at 111 ms 82 us and finishs at 111 ms 88 us
17:16: 5 sends to 16:16: 5 starts at 111 ms 88 us and finishs at 111 ms 128 us
17:16: 5 sends to 18:29:10 starts at 111 ms 128 us and finishs at 111 ms 138 us
17:16: 5 sends to 19:16: 5 starts at 111 ms 138 us and finishs at 111 ms 178 us
18:16: 5 sends to 19:16: 5 starts at 111 ms 178 us and finishs at 111 ms 218 us
18:29:10 sends to 19:16: 5 starts at 111 ms 218 us and finishs at 111 ms 239 us
8:26:22 sends to 6:22: 2 starts at 111 ms 989 us and finishs at 112 ms 130 us
6:22: 2 sends to 8:28: 1 starts at 112 ms 133 us and finishs at 112 ms 139 us
8:26:22 sends to 8:28: 1 starts at 112 ms 139 us and finishs at 112 ms 150 us
7:20: 2 sends to 8:30:20 starts at 112 ms 150 us and finishs at 112 ms 198 us
7:20: 3 sends to 8:30:20 starts at 114 ms 551 us and finishs at 114 ms 575 us
7:29: 1 sends to 7:35: 4 starts at 114 ms 648 us and finishs at 114 ms 717 us
8:26:22 sends to 7:35: 4 starts at 114 ms 717 us and finishs at 114 ms 757 us
18:35: 4 sends to 18:33: 2 starts at 115 ms 621 us and finishs at 115 ms 690 us
19:26:22 sends to 18:33: 2 starts at 115 ms 690 us and finishs at 115 ms 715 us
7:35: 4 sends to 7:29: 1 starts at 116 ms 817 us and finishs at 116 ms 835 us
7:35: 4 sends to 8:26:22 starts at 116 ms 835 us and finishs at 116 ms 904 us
16:21: 2 sends to 16:22: 2 starts at 117 ms 564 us and finishs at 117 ms 564 us
19:26:22 sends to 16:22: 2 starts at 117 ms 564 us and finishs at 117 ms 705 us
18:33: 2 sends to 18:35: 2 starts at 117 ms 705 us and finishs at 117 ms 722 us
6:19: 2 sends to 8:30:20 starts at 117 ms 957 us and finishs at 118 ms 1 us
6:17: 1 sends to 5:17: 1 starts at 118 ms 1 us and finishs at 118 ms 7 us
18:35: 2 sends to 18:33: 2 starts at 118 ms 533 us and finishs at 118 ms 553 us
18:35: 2 sends to 19:26:22 starts at 118 ms 553 us and finishs at 118 ms 559 us
8:28: 1 sends to 6:22: 2 starts at 118 ms 690 us and finishs at 118 ms 693 us
8:28: 1 sends to 8:26:22 starts at 118 ms 693 us and finishs at 118 ms 704 us
5:17: 1 sends to 6:17: 1 starts at 118 ms 982 us and finishs at 118 ms 988 us
5:17: 1 sends to 8:30:20 starts at 118 ms 988 us and finishs at 119 ms 342 us
5:17: 1 sends to 8:17: 1 starts at 119 ms 342 us and finishs at 119 ms 348 us
6:17: 1 sends to 8:17: 1 starts at 119 ms 348 us and finishs at 119 ms 354 us
7:17: 1 sends to 8:17: 1 starts at 119 ms 354 us and finishs at 119 ms 360 us
16:22: 2 sends to 19:28: 1 starts at 119 ms 545 us and finishs at 119 ms 551 us
8:28: 1 sends to 5:21: 3 starts at 119 ms 614 us and finishs at 119 ms 617 us
16:21: 5 sends to 18:29:10 starts at 120 ms 254 us and finishs at 120 ms 263 us
16:21: 5 sends to 18:33: 2 starts at 120 ms 263 us and finishs at 120 ms 280 us
16:21: 5 sends to 19:26:22 starts at 120 ms 280 us and finishs at 120 ms 284 us

16:21: 5 sends to 18:32: 1 starts at 120 ms 284 us and finishs at 120 ms 301 us
17:19: 1 sends to 18:32: 1 starts at 120 ms 301 us and finishs at 120 ms 304 us
17:21: 3 sends to 18:32: 1 starts at 120 ms 304 us and finishs at 120 ms 312 us
19:16: 2 sends to 18:32: 1 starts at 120 ms 312 us and finishs at 120 ms 313 us
19:27: 1 sends to 18:32: 1 starts at 120 ms 313 us and finishs at 120 ms 318 us
17:19: 1 sends to 16:22: 4 starts at 120 ms 318 us and finishs at 120 ms 330 us
8:17: 1 sends to 5:17: 1 starts at 120 ms 335 us and finishs at 120 ms 341 us
8:17: 1 sends to 6:17: 1 starts at 120 ms 341 us and finishs at 120 ms 347 us
8:17: 1 sends to 7:17: 1 starts at 120 ms 347 us and finishs at 120 ms 353 us
8:17: 1 sends to 8:30:20 starts at 120 ms 353 us and finishs at 120 ms 707 us
17:19: 4 sends to 16:22: 4 starts at 120 ms 707 us and finishs at 120 ms 721 us
17:19: 5 sends to 16:22: 4 starts at 120 ms 721 us and finishs at 120 ms 731 us
17:21: 4 sends to 16:22: 4 starts at 120 ms 731 us and finishs at 120 ms 734 us
17:22: 2 sends to 16:22: 4 starts at 120 ms 734 us and finishs at 120 ms 770 us
18:20: 4 sends to 16:22: 4 starts at 120 ms 770 us and finishs at 120 ms 780 us
18:29:10 sends to 16:22: 4 starts at 120 ms 780 us and finishs at 120 ms 785 us
18:35: 1 sends to 16:22: 4 starts at 120 ms 785 us and finishs at 120 ms 791 us
18:35: 2 sends to 16:22: 4 starts at 120 ms 791 us and finishs at 120 ms 797 us
18:35: 4 sends to 16:22: 4 starts at 120 ms 797 us and finishs at 120 ms 815 us
19:16: 2 sends to 16:22: 4 starts at 120 ms 815 us and finishs at 120 ms 822 us
19:20: 4 sends to 16:22: 4 starts at 120 ms 822 us and finishs at 120 ms 832 us
19:26:22 sends to 16:22: 4 starts at 120 ms 832 us and finishs at 120 ms 846 us
16:21: 5 sends to 17:21: 5 starts at 120 ms 846 us and finishs at 120 ms 882 us
17:16: 2 sends to 17:21: 5 starts at 120 ms 882 us and finishs at 120 ms 883 us
18:29:10 sends to 17:21: 5 starts at 120 ms 883 us and finishs at 120 ms 904 us
19:16: 2 sends to 17:21: 5 starts at 120 ms 904 us and finishs at 120 ms 905 us
19:26:22 sends to 17:21: 5 starts at 120 ms 905 us and finishs at 120 ms 919 us
19:28: 1 sends to 17:21: 5 starts at 120 ms 919 us and finishs at 120 ms 924 us
8:16: 5 sends to 1: 0: 1 starts at 121 ms 574 us and finishs at 121 ms 580 us
8:16: 5 sends to 2: 0: 1 starts at 121 ms 580 us and finishs at 121 ms 586 us
8:16: 5 sends to 3: 0: 1 starts at 121 ms 586 us and finishs at 121 ms 592 us
8:16: 5 sends to 4: 0: 1 starts at 121 ms 592 us and finishs at 121 ms 598 us
8:16: 5 sends to 5:16: 5 starts at 121 ms 598 us and finishs at 121 ms 638 us
8:16: 5 sends to 6:16: 5 starts at 121 ms 638 us and finishs at 121 ms 678 us
8:16: 5 sends to 7:16: 5 starts at 121 ms 678 us and finishs at 121 ms 718 us
8:16: 5 sends to 7:29:10 starts at 121 ms 718 us and finishs at 121 ms 728 us
5:21: 2 sends to 5:22: 2 starts at 122 ms 414 us and finishs at 122 ms 414 us
6:22: 2 sends to 5:22: 2 starts at 122 ms 414 us and finishs at 122 ms 417 us
8:28: 1 sends to 5:22: 2 starts at 122 ms 417 us and finishs at 122 ms 420 us
5:21: 3 sends to 6:22: 2 starts at 122 ms 483 us and finishs at 122 ms 491 us
5:21: 3 sends to 8:28: 1 starts at 122 ms 491 us and finishs at 122 ms 498 us
17:22: 3 sends to 17:21: 4 starts at 124 ms 468 us and finishs at 124 ms 468 us
17:22: 3 sends to 18:29:10 starts at 124 ms 468 us and finishs at 124 ms 502 us
5:22: 2 sends to 6:22: 2 starts at 124 ms 533 us and finishs at 124 ms 536 us
5:22: 2 sends to 8:28: 1 starts at 124 ms 536 us and finishs at 124 ms 542 us
19:16: 5 sends to 12: 0: 1 starts at 124 ms 615 us and finishs at 124 ms 621 us
19:16: 5 sends to 13: 0: 1 starts at 124 ms 621 us and finishs at 124 ms 627 us
19:16: 5 sends to 14: 0: 1 starts at 124 ms 627 us and finishs at 124 ms 633 us
19:16: 5 sends to 15: 0: 1 starts at 124 ms 633 us and finishs at 124 ms 639 us
19:16: 5 sends to 16:16: 5 starts at 124 ms 639 us and finishs at 124 ms 679 us
19:16: 5 sends to 18:29:10 starts at 124 ms 679 us and finishs at 124 ms 689 us

15: 0: 3 sends to 18:29: 1 starts at 124 ms 690 us and finishs at 124 ms 691 us
12: 0: 3 sends to 18:29: 1 starts at 124 ms 691 us and finishs at 124 ms 692 us
13: 0: 3 sends to 18:29: 1 starts at 124 ms 697 us and finishs at 124 ms 698 us
14: 0: 3 sends to 18:29: 1 starts at 124 ms 698 us and finishs at 124 ms 699 us
19:16: 5 sends to 17:16: 5 starts at 124 ms 699 us and finishs at 124 ms 739 us
19:16: 5 sends to 18:16: 5 starts at 124 ms 739 us and finishs at 124 ms 779 us
1: 0: 3 sends to 7:29: 1 starts at 125 ms 874 us and finishs at 125 ms 875 us
4: 0: 3 sends to 7:29: 1 starts at 125 ms 875 us and finishs at 125 ms 876 us
19:28: 1 sends to 16:22: 2 starts at 126 ms 96 us and finishs at 126 ms 99 us
5:21: 3 sends to 6:21: 3 starts at 126 ms 107 us and finishs at 126 ms 112 us
8:26:22 sends to 6:21: 3 starts at 126 ms 112 us and finishs at 126 ms 115 us
8:28: 1 sends to 6:21: 3 starts at 126 ms 115 us and finishs at 126 ms 118 us
5:21: 3 sends to 8:27: 1 starts at 127 ms 151 us and finishs at 127 ms 158 us
8:26:22 sends to 8:27: 1 starts at 127 ms 158 us and finishs at 127 ms 173 us
19:17: 1 sends to 18:17: 1 starts at 128 ms 896 us and finishs at 128 ms 902 us
6:21: 3 sends to 5:21: 3 starts at 128 ms 915 us and finishs at 128 ms 920 us
6:21: 3 sends to 5:22: 2 starts at 128 ms 920 us and finishs at 128 ms 928 us
6:21: 3 sends to 8:26:22 starts at 128 ms 928 us and finishs at 128 ms 943 us
6:21: 3 sends to 8:27: 1 starts at 128 ms 943 us and finishs at 128 ms 950 us
6:21: 3 sends to 8:28: 1 starts at 128 ms 950 us and finishs at 128 ms 957 us
6:21: 3 sends to 8:30:20 starts at 128 ms 957 us and finishs at 128 ms 994 us
16:22: 2 sends to 16:21: 3 starts at 129 ms 842 us and finishs at 129 ms 842 us
19:26:22 sends to 16:21: 3 starts at 129 ms 842 us and finishs at 129 ms 845 us
19:28: 1 sends to 16:21: 3 starts at 129 ms 845 us and finishs at 129 ms 848 us
18:17: 1 sends to 19:17: 1 starts at 129 ms 896 us and finishs at 129 ms 902 us
18:17: 1 sends to 16:17: 1 starts at 130 ms 694 us and finishs at 130 ms 700 us
19:17: 1 sends to 16:17: 1 starts at 130 ms 700 us and finishs at 130 ms 706 us
8:27: 1 sends to 8:26:22 starts at 130 ms 708 us and finishs at 130 ms 713 us
8:27: 1 sends to 8:30:20 starts at 130 ms 713 us and finishs at 130 ms 722 us
5:19: 2 sends to 8:30:20 starts at 130 ms 728 us and finishs at 130 ms 772 us
16:17: 1 sends to 18:17: 1 starts at 131 ms 689 us and finishs at 131 ms 695 us
16:17: 1 sends to 19:17: 1 starts at 131 ms 695 us and finishs at 131 ms 701 us
16:17: 1 sends to 17:17: 1 starts at 131 ms 701 us and finishs at 131 ms 707 us
18:17: 1 sends to 17:17: 1 starts at 131 ms 707 us and finishs at 131 ms 713 us
19:17: 1 sends to 17:17: 1 starts at 131 ms 713 us and finishs at 131 ms 719 us
16:21: 3 sends to 16:22: 2 starts at 132 ms 645 us and finishs at 132 ms 657 us
16:21: 3 sends to 19:26:22 starts at 132 ms 657 us and finishs at 132 ms 672 us
16:21: 3 sends to 19:28: 1 starts at 132 ms 672 us and finishs at 132 ms 679 us
16:21: 3 sends to 19:27: 1 starts at 132 ms 687 us and finishs at 132 ms 694 us
19:26:22 sends to 19:27: 1 starts at 132 ms 694 us and finishs at 132 ms 709 us
17:17: 1 sends to 16:17: 1 starts at 132 ms 709 us and finishs at 132 ms 715 us
17:17: 1 sends to 18:17: 1 starts at 132 ms 715 us and finishs at 132 ms 721 us
17:17: 1 sends to 19:17: 1 starts at 132 ms 721 us and finishs at 132 ms 727 us
5:22: 2 sends to 7:33: 2 starts at 132 ms 781 us and finishs at 132 ms 785 us
6:22: 2 sends to 7:33: 2 starts at 132 ms 785 us and finishs at 132 ms 789 us
7:35: 1 sends to 7:33: 2 starts at 132 ms 789 us and finishs at 132 ms 795 us
7:35: 4 sends to 7:33: 2 starts at 132 ms 795 us and finishs at 132 ms 864 us
8:26:22 sends to 7:33: 2 starts at 132 ms 864 us and finishs at 132 ms 889 us
7:33: 2 sends to 5:22: 2 starts at 134 ms 807 us and finishs at 134 ms 875 us
7:33: 2 sends to 6:22: 2 starts at 134 ms 875 us and finishs at 134 ms 943 us
7:33: 2 sends to 8:30:20 starts at 134 ms 943 us and finishs at 134 ms 947 us

7:33: 2 sends to 7:35: 2 starts at 134 ms 947 us and finishs at 134 ms 964 us
7:35: 2 sends to 7:33: 2 starts at 135 ms 775 us and finishs at 135 ms 795 us
7:35: 2 sends to 8:26:22 starts at 135 ms 795 us and finishs at 135 ms 801 us
19:27: 1 sends to 19:26:22 starts at 136 ms 244 us and finishs at 136 ms 249 us
6:22: 3 sends to 6:21: 4 starts at 137 ms 547 us and finishs at 137 ms 547 us
6:22: 3 sends to 7:29:10 starts at 137 ms 547 us and finishs at 137 ms 581 us
6:22: 3 sends to 8:30:20 starts at 137 ms 581 us and finishs at 137 ms 588 us
17:21: 2 sends to 17:22: 2 starts at 138 ms 486 us and finishs at 138 ms 486 us
16:21: 3 sends to 17:22: 2 starts at 138 ms 486 us and finishs at 138 ms 494 us
19:26:22 sends to 17:22: 2 starts at 138 ms 494 us and finishs at 138 ms 635 us
19:28: 1 sends to 17:22: 2 starts at 138 ms 635 us and finishs at 138 ms 638 us
5:19: 4 sends to 8:30:20 starts at 139 ms 688 us and finishs at 139 ms 730 us
5:21: 2 sends to 8:30:20 starts at 139 ms 730 us and finishs at 139 ms 732 us
5:21: 3 sends to 8:30:20 starts at 139 ms 732 us and finishs at 139 ms 769 us
5:22: 2 sends to 8:30:20 starts at 139 ms 769 us and finishs at 139 ms 774 us
6:21: 2 sends to 8:30:20 starts at 139 ms 774 us and finishs at 139 ms 776 us
6:22: 2 sends to 8:30:20 starts at 139 ms 776 us and finishs at 139 ms 781 us
7:29: 1 sends to 8:30:20 starts at 139 ms 781 us and finishs at 139 ms 796 us
8:26:22 sends to 8:30:20 starts at 139 ms 796 us and finishs at 139 ms 859 us
8:28: 1 sends to 8:30:20 starts at 139 ms 859 us and finishs at 139 ms 885 us
16:21: 3 sends to 17:21: 3 starts at 140 ms 536 us and finishs at 140 ms 541 us
19:26:22 sends to 17:21: 3 starts at 140 ms 541 us and finishs at 140 ms 544 us
19:28: 1 sends to 17:21: 3 starts at 140 ms 544 us and finishs at 140 ms 547 us
17:22: 2 sends to 19:28: 1 starts at 140 ms 685 us and finishs at 140 ms 691 us
19:26:22 sends to 17:21: 4 starts at 141 ms 488 us and finishs at 141 ms 493 us
19:27: 1 sends to 17:21: 4 starts at 141 ms 493 us and finishs at 141 ms 498 us
19:28: 1 sends to 17:21: 4 starts at 141 ms 498 us and finishs at 141 ms 503 us
19:26:22 sends to 16:21: 4 starts at 143 ms 196 us and finishs at 143 ms 201 us
19:28: 1 sends to 16:21: 4 starts at 143 ms 201 us and finishs at 143 ms 206 us
17:21: 3 sends to 16:21: 3 starts at 143 ms 485 us and finishs at 143 ms 490 us
17:21: 3 sends to 19:26:22 starts at 143 ms 490 us and finishs at 143 ms 505 us
17:21: 3 sends to 19:28: 1 starts at 143 ms 505 us and finishs at 143 ms 512 us
17:21: 5 sends to 16:21: 5 starts at 146 ms 290 us and finishs at 146 ms 326 us
17:21: 5 sends to 18:29:10 starts at 146 ms 326 us and finishs at 146 ms 335 us
17:21: 5 sends to 18:32: 1 starts at 146 ms 335 us and finishs at 146 ms 352 us
17:21: 5 sends to 19:26:22 starts at 146 ms 352 us and finishs at 146 ms 356 us
16:22: 4 sends to 17:21: 3 starts at 146 ms 807 us and finishs at 146 ms 809 us
16:22: 4 sends to 19:28: 1 starts at 146 ms 809 us and finishs at 146 ms 816 us
17:16: 2 sends to 17:22: 4 starts at 146 ms 816 us and finishs at 146 ms 823 us
17:19: 1 sends to 17:22: 4 starts at 146 ms 823 us and finishs at 146 ms 835 us
17:19: 4 sends to 17:22: 4 starts at 146 ms 835 us and finishs at 146 ms 849 us
17:19: 5 sends to 17:22: 4 starts at 146 ms 849 us and finishs at 146 ms 859 us
17:21: 5 sends to 17:22: 4 starts at 146 ms 859 us and finishs at 146 ms 859 us
17:21: 4 sends to 17:22: 4 starts at 146 ms 859 us and finishs at 146 ms 862 us
18:16: 2 sends to 17:22: 4 starts at 146 ms 862 us and finishs at 146 ms 869 us
18:20: 4 sends to 17:22: 4 starts at 146 ms 869 us and finishs at 146 ms 879 us
18:29:10 sends to 17:22: 4 starts at 146 ms 879 us and finishs at 146 ms 884 us
18:35: 1 sends to 17:22: 4 starts at 146 ms 884 us and finishs at 146 ms 890 us
18:35: 2 sends to 17:22: 4 starts at 146 ms 890 us and finishs at 146 ms 896 us
18:35: 4 sends to 17:22: 4 starts at 146 ms 896 us and finishs at 146 ms 914 us
19:16: 2 sends to 17:22: 4 starts at 146 ms 914 us and finishs at 146 ms 921 us

19:20: 4 sends to 17:22: 4 starts at 146 ms 921 us and finishs at 146 ms 931 us
19:26:22 sends to 17:22: 4 starts at 146 ms 931 us and finishs at 146 ms 945 us
6:22: 4 sends to 6:21: 5 starts at 149 ms 547 us and finishs at 149 ms 547 us
6:22: 4 sends to 5:21: 3 starts at 149 ms 547 us and finishs at 149 ms 549 us
6:22: 4 sends to 8:28: 1 starts at 149 ms 549 us and finishs at 149 ms 553 us
6:22: 4 sends to 8:30:20 starts at 149 ms 553 us and finishs at 149 ms 555 us
6:22: 4 sends to 5:22: 4 starts at 149 ms 555 us and finishs at 149 ms 563 us
7:16: 2 sends to 5:22: 4 starts at 149 ms 563 us and finishs at 149 ms 570 us
7:20: 4 sends to 5:22: 4 starts at 149 ms 570 us and finishs at 149 ms 580 us
7:29:10 sends to 5:22: 4 starts at 149 ms 580 us and finishs at 149 ms 585 us
7:32: 3 sends to 5:22: 4 starts at 149 ms 585 us and finishs at 149 ms 591 us
7:35: 1 sends to 5:22: 4 starts at 149 ms 591 us and finishs at 149 ms 597 us
7:35: 2 sends to 5:22: 4 starts at 149 ms 597 us and finishs at 149 ms 603 us
7:35: 4 sends to 5:22: 4 starts at 149 ms 603 us and finishs at 149 ms 621 us
8:16: 2 sends to 5:22: 4 starts at 149 ms 621 us and finishs at 149 ms 628 us
15: 0: 3 sends to 18:29: 1 starts at 149 ms 628 us and finishs at 149 ms 629 us
12: 0: 3 sends to 18:29: 1 starts at 149 ms 629 us and finishs at 149 ms 630 us
7:16: 2 sends to 6:21: 5 starts at 149 ms 630 us and finishs at 149 ms 631 us
8:16: 2 sends to 6:21: 5 starts at 149 ms 631 us and finishs at 149 ms 632 us
13: 0: 3 sends to 18:29: 1 starts at 149 ms 636 us and finishs at 149 ms 637 us
14: 0: 3 sends to 18:29: 1 starts at 149 ms 637 us and finishs at 149 ms 638 us
8:20: 4 sends to 5:22: 4 starts at 149 ms 638 us and finishs at 149 ms 648 us
8:26:22 sends to 5:22: 4 starts at 149 ms 648 us and finishs at 149 ms 662 us
8:26:22 sends to 6:21: 5 starts at 149 ms 662 us and finishs at 149 ms 676 us
8:28: 1 sends to 6:21: 5 starts at 149 ms 676 us and finishs at 149 ms 681 us
1: 0: 3 sends to 7:29: 1 starts at 150 ms 813 us and finishs at 150 ms 814 us
4: 0: 3 sends to 7:29: 1 starts at 150 ms 814 us and finishs at 150 ms 815 us
18:35: 1 sends to 18:29: 1 starts at 150 ms 856 us and finishs at 150 ms 862 us
18:35: 1 sends to 18:33: 2 starts at 150 ms 862 us and finishs at 150 ms 868 us
18:29: 1 sends to 18:35: 4 starts at 152 ms 80 us and finishs at 152 ms 149 us
1: 0: 3 sends to 8:26:22 starts at 153 ms 263 us and finishs at 153 ms 265 us
4: 0: 3 sends to 8:26:22 starts at 153 ms 265 us and finishs at 153 ms 267 us
16:21: 4 sends to 16:22: 3 starts at 153 ms 983 us and finishs at 153 ms 983 us
16:21: 3 sends to 16:22: 3 starts at 153 ms 983 us and finishs at 153 ms 989 us
5:19: 3 sends to 8:30:20 starts at 153 ms 989 us and finishs at 154 ms 44 us
17:21: 3 sends to 16:22: 3 starts at 154 ms 44 us and finishs at 154 ms 50 us
19:26:22 sends to 16:22: 3 starts at 154 ms 50 us and finishs at 154 ms 55 us
19:27: 1 sends to 16:22: 3 starts at 154 ms 55 us and finishs at 154 ms 60 us
19:28: 1 sends to 16:22: 3 starts at 154 ms 60 us and finishs at 154 ms 65 us
18:35: 4 sends to 18:29: 1 starts at 154 ms 180 us and finishs at 154 ms 198 us
6:21: 2 sends to 6:22: 2 starts at 154 ms 776 us and finishs at 154 ms 776 us
5:21: 3 sends to 7:29:10 starts at 154 ms 777 us and finishs at 154 ms 783 us
6:21: 3 sends to 7:29:10 starts at 154 ms 783 us and finishs at 154 ms 789 us
7:35: 1 sends to 7:29:10 starts at 154 ms 789 us and finishs at 154 ms 798 us
7:35: 4 sends to 7:29:10 starts at 154 ms 798 us and finishs at 154 ms 815 us
8:26:22 sends to 7:29:10 starts at 154 ms 815 us and finishs at 154 ms 829 us
8:27: 1 sends to 7:29:10 starts at 154 ms 829 us and finishs at 154 ms 838 us
8:28: 1 sends to 7:29:10 starts at 154 ms 838 us and finishs at 154 ms 847 us
8:30:20 sends to 7:29:10 starts at 154 ms 847 us and finishs at 154 ms 856 us
12: 0: 3 sends to 19:26:22 starts at 156 ms 830 us and finishs at 156 ms 832 us
13: 0: 3 sends to 19:26:22 starts at 156 ms 832 us and finishs at 156 ms 834 us

7:35: 1 sends to 7:29: 1 starts at 158 ms 71 us and finishs at 158 ms 77 us
7:35: 1 sends to 8:26:22 starts at 158 ms 77 us and finishs at 158 ms 83 us
16:22: 3 sends to 16:21: 4 starts at 161 ms 711 us and finishs at 161 ms 711 us
8:26:22 sends to 6:22: 2 starts at 161 ms 867 us and finishs at 162 ms 8 us
6:22: 2 sends to 8:28: 1 starts at 162 ms 14 us and finishs at 162 ms 20 us
8:26:22 sends to 8:28: 1 starts at 162 ms 20 us and finishs at 162 ms 31 us
7:20: 3 sends to 8:30:20 starts at 164 ms 427 us and finishs at 164 ms 451 us
7:29: 1 sends to 7:35: 4 starts at 164 ms 592 us and finishs at 164 ms 661 us
8:26:22 sends to 7:35: 4 starts at 164 ms 661 us and finishs at 164 ms 701 us
18:35: 4 sends to 18:33: 2 starts at 165 ms 565 us and finishs at 165 ms 634 us
19:26:22 sends to 18:33: 2 starts at 165 ms 634 us and finishs at 165 ms 659 us
8:26:22 sends to 6:21: 4 starts at 166 ms 623 us and finishs at 166 ms 628 us
8:27: 1 sends to 6:21: 4 starts at 166 ms 628 us and finishs at 166 ms 633 us
8:28: 1 sends to 6:21: 4 starts at 166 ms 633 us and finishs at 166 ms 638 us
7:35: 4 sends to 7:29: 1 starts at 166 ms 761 us and finishs at 166 ms 779 us
7:35: 4 sends to 8:26:22 starts at 166 ms 779 us and finishs at 166 ms 848 us
16:21: 2 sends to 16:22: 2 starts at 167 ms 508 us and finishs at 167 ms 508 us
19:26:22 sends to 16:22: 2 starts at 167 ms 508 us and finishs at 167 ms 649 us
18:33: 2 sends to 18:35: 2 starts at 167 ms 649 us and finishs at 167 ms 666 us
8:26:22 sends to 5:21: 4 starts at 168 ms 339 us and finishs at 168 ms 344 us
8:27: 1 sends to 5:21: 4 starts at 168 ms 344 us and finishs at 168 ms 349 us
8:28: 1 sends to 5:21: 4 starts at 168 ms 349 us and finishs at 168 ms 354 us
18:35: 2 sends to 18:33: 2 starts at 168 ms 477 us and finishs at 168 ms 497 us
18:35: 2 sends to 19:26:22 starts at 168 ms 497 us and finishs at 168 ms 503 us
8:28: 1 sends to 6:22: 2 starts at 168 ms 571 us and finishs at 168 ms 574 us
8:28: 1 sends to 8:26:22 starts at 168 ms 574 us and finishs at 168 ms 585 us
16:22: 2 sends to 19:28: 1 starts at 169 ms 420 us and finishs at 169 ms 426 us
8:28: 1 sends to 5:21: 3 starts at 169 ms 558 us and finishs at 169 ms 561 us
7:29:10 sends to 7:35: 1 starts at 171 ms 497 us and finishs at 171 ms 518 us
7:29:10 sends to 7:35: 4 starts at 171 ms 518 us and finishs at 171 ms 535 us
7:29:10 sends to 8:26:22 starts at 171 ms 535 us and finishs at 171 ms 556 us
7:29:10 sends to 8:27: 1 starts at 171 ms 556 us and finishs at 171 ms 577 us
7:29:10 sends to 8:28: 1 starts at 171 ms 577 us and finishs at 171 ms 598 us
7:29:10 sends to 8:30:20 starts at 171 ms 598 us and finishs at 171 ms 662 us
8:30:20 sends to 7:31: 1 starts at 171 ms 662 us and finishs at 171 ms 832 us
5:21: 2 sends to 5:22: 2 starts at 172 ms 358 us and finishs at 172 ms 358 us
6:22: 2 sends to 5:22: 2 starts at 172 ms 358 us and finishs at 172 ms 361 us
8:28: 1 sends to 5:22: 2 starts at 172 ms 361 us and finishs at 172 ms 364 us
5:21: 3 sends to 6:22: 2 starts at 172 ms 427 us and finishs at 172 ms 435 us
5:21: 3 sends to 8:28: 1 starts at 172 ms 435 us and finishs at 172 ms 442 us
7:31: 1 sends to 8:30:20 starts at 173 ms 212 us and finishs at 173 ms 219 us
5:21: 4 sends to 8:30:20 starts at 174 ms 378 us and finishs at 174 ms 397 us
5:22: 2 sends to 6:22: 2 starts at 174 ms 477 us and finishs at 174 ms 480 us
5:22: 2 sends to 8:28: 1 starts at 174 ms 480 us and finishs at 174 ms 486 us
15: 0: 3 sends to 18:29: 1 starts at 174 ms 566 us and finishs at 174 ms 567 us
12: 0: 3 sends to 18:29: 1 starts at 174 ms 567 us and finishs at 174 ms 568 us
13: 0: 3 sends to 18:29: 1 starts at 174 ms 575 us and finishs at 174 ms 576 us
14: 0: 3 sends to 18:29: 1 starts at 174 ms 576 us and finishs at 174 ms 577 us
1: 0: 3 sends to 7:29: 1 starts at 175 ms 752 us and finishs at 175 ms 753 us
4: 0: 3 sends to 7:29: 1 starts at 175 ms 753 us and finishs at 175 ms 754 us
19:28: 1 sends to 16:22: 2 starts at 175 ms 971 us and finishs at 175 ms 974 us

5:21: 3 sends to 6:21: 3 starts at 175 ms 990 us and finishs at 175 ms 995 us
8:26:22 sends to 6:21: 3 starts at 175 ms 995 us and finishs at 175 ms 998 us
8:28: 1 sends to 6:21: 3 starts at 175 ms 998 us and finishs at 176 ms 1 us
5:21: 3 sends to 8:27: 1 starts at 177 ms 33 us and finishs at 177 ms 40 us
8:26:22 sends to 8:27: 1 starts at 177 ms 40 us and finishs at 177 ms 55 us
16:21: 3 sends to 18:29:10 starts at 178 ms 782 us and finishs at 178 ms 788 us
6:21: 3 sends to 5:21: 3 starts at 178 ms 798 us and finishs at 178 ms 803 us
6:21: 3 sends to 5:22: 2 starts at 178 ms 803 us and finishs at 178 ms 811 us
6:21: 3 sends to 8:26:22 starts at 178 ms 811 us and finishs at 178 ms 826 us
6:21: 3 sends to 8:27: 1 starts at 178 ms 826 us and finishs at 178 ms 833 us
6:21: 3 sends to 8:28: 1 starts at 178 ms 833 us and finishs at 178 ms 840 us
6:21: 3 sends to 8:30:20 starts at 178 ms 840 us and finishs at 178 ms 877 us
16:22: 3 sends to 18:29:10 starts at 178 ms 877 us and finishs at 178 ms 911 us
18:33: 2 sends to 18:29:10 starts at 178 ms 911 us and finishs at 178 ms 911 us
17:21: 3 sends to 18:29:10 starts at 178 ms 911 us and finishs at 178 ms 917 us
19:27: 1 sends to 18:29:10 starts at 178 ms 917 us and finishs at 178 ms 926 us
16:22: 2 sends to 16:21: 3 starts at 179 ms 720 us and finishs at 179 ms 720 us
19:26:22 sends to 16:21: 3 starts at 179 ms 720 us and finishs at 179 ms 723 us
19:28: 1 sends to 16:21: 3 starts at 179 ms 723 us and finishs at 179 ms 726 us
8:27: 1 sends to 8:26:22 starts at 180 ms 590 us and finishs at 180 ms 595 us
8:27: 1 sends to 8:30:20 starts at 180 ms 595 us and finishs at 180 ms 604 us
16:21: 3 sends to 16:22: 2 starts at 182 ms 523 us and finishs at 182 ms 535 us
16:21: 3 sends to 19:26:22 starts at 182 ms 535 us and finishs at 182 ms 550 us
16:21: 3 sends to 19:28: 1 starts at 182 ms 550 us and finishs at 182 ms 557 us
16:21: 3 sends to 19:27: 1 starts at 182 ms 569 us and finishs at 182 ms 576 us
19:26:22 sends to 19:27: 1 starts at 182 ms 576 us and finishs at 182 ms 591 us
5:22: 2 sends to 7:33: 2 starts at 182 ms 739 us and finishs at 182 ms 743 us
6:22: 2 sends to 7:33: 2 starts at 182 ms 743 us and finishs at 182 ms 747 us
7:35: 1 sends to 7:33: 2 starts at 182 ms 747 us and finishs at 182 ms 753 us
7:35: 4 sends to 7:33: 2 starts at 182 ms 753 us and finishs at 182 ms 822 us
8:26:22 sends to 7:33: 2 starts at 182 ms 822 us and finishs at 182 ms 847 us
7:33: 2 sends to 5:22: 2 starts at 184 ms 765 us and finishs at 184 ms 833 us
7:33: 2 sends to 6:22: 2 starts at 184 ms 833 us and finishs at 184 ms 901 us
7:33: 2 sends to 8:30:20 starts at 184 ms 901 us and finishs at 184 ms 905 us
7:33: 2 sends to 7:35: 2 starts at 184 ms 905 us and finishs at 184 ms 922 us
7:35: 2 sends to 7:33: 2 starts at 185 ms 733 us and finishs at 185 ms 753 us
7:35: 2 sends to 8:26:22 starts at 185 ms 753 us and finishs at 185 ms 759 us
19:27: 1 sends to 19:26:22 starts at 186 ms 126 us and finishs at 186 ms 131 us
17:21: 2 sends to 17:22: 2 starts at 188 ms 510 us and finishs at 188 ms 510 us
16:21: 3 sends to 17:22: 2 starts at 188 ms 510 us and finishs at 188 ms 518 us
19:26:22 sends to 17:22: 2 starts at 188 ms 518 us and finishs at 188 ms 659 us
19:28: 1 sends to 17:22: 2 starts at 188 ms 659 us and finishs at 188 ms 662 us
5:19: 4 sends to 8:30:20 starts at 189 ms 734 us and finishs at 189 ms 776 us
5:21: 2 sends to 8:30:20 starts at 189 ms 776 us and finishs at 189 ms 778 us
5:21: 3 sends to 8:30:20 starts at 189 ms 778 us and finishs at 189 ms 815 us
5:22: 2 sends to 8:30:20 starts at 189 ms 815 us and finishs at 189 ms 820 us
6:21: 2 sends to 8:30:20 starts at 189 ms 820 us and finishs at 189 ms 822 us
6:22: 2 sends to 8:30:20 starts at 189 ms 822 us and finishs at 189 ms 827 us
7:29: 1 sends to 8:30:20 starts at 189 ms 827 us and finishs at 189 ms 842 us
8:26:22 sends to 8:30:20 starts at 189 ms 842 us and finishs at 189 ms 905 us
8:28: 1 sends to 8:30:20 starts at 189 ms 905 us and finishs at 189 ms 931 us

16:21: 3 sends to 17:21: 3 starts at 190 ms 560 us and finishs at 190 ms 565 us
19:26:22 sends to 17:21: 3 starts at 190 ms 565 us and finishs at 190 ms 568 us
19:28: 1 sends to 17:21: 3 starts at 190 ms 568 us and finishs at 190 ms 571 us
17:22: 2 sends to 19:28: 1 starts at 190 ms 709 us and finishs at 190 ms 715 us
5:21: 3 sends to 5:22: 3 starts at 191 ms 364 us and finishs at 191 ms 370 us
6:21: 3 sends to 5:22: 3 starts at 191 ms 370 us and finishs at 191 ms 376 us
8:26:22 sends to 5:22: 3 starts at 191 ms 376 us and finishs at 191 ms 381 us
8:27: 1 sends to 5:22: 3 starts at 191 ms 381 us and finishs at 191 ms 386 us
8:28: 1 sends to 5:22: 3 starts at 191 ms 386 us and finishs at 191 ms 391 us
5:22: 4 sends to 5:21: 3 starts at 192 ms 590 us and finishs at 192 ms 592 us
5:22: 4 sends to 6:21: 3 starts at 192 ms 592 us and finishs at 192 ms 594 us
5:22: 4 sends to 6:22: 4 starts at 192 ms 594 us and finishs at 192 ms 602 us
5:22: 4 sends to 8:30:20 starts at 192 ms 602 us and finishs at 192 ms 604 us
17:21: 3 sends to 16:21: 3 starts at 193 ms 509 us and finishs at 193 ms 514 us
17:21: 3 sends to 19:26:22 starts at 193 ms 514 us and finishs at 193 ms 529 us
17:21: 3 sends to 19:28: 1 starts at 193 ms 529 us and finishs at 193 ms 536 us
18:29:10 sends to 18:33: 2 starts at 195 ms 567 us and finishs at 195 ms 612 us
18:29:10 sends to 19:27: 1 starts at 195 ms 612 us and finishs at 195 ms 633 us
5:22: 3 sends to 7:29:10 starts at 199 ms 37 us and finishs at 199 ms 71 us
5:22: 3 sends to 8:30:20 starts at 199 ms 71 us and finishs at 199 ms 78 us
6:21: 5 sends to 6:22: 4 starts at 199 ms 366 us and finishs at 199 ms 366 us
6:21: 5 sends to 5:21: 5 starts at 199 ms 366 us and finishs at 199 ms 402 us
6:21: 5 sends to 7:32: 1 starts at 199 ms 402 us and finishs at 199 ms 419 us
6:21: 5 sends to 8:26:22 starts at 199 ms 419 us and finishs at 199 ms 423 us
6:21: 5 sends to 8:30:20 starts at 199 ms 423 us and finishs at 199 ms 425 us
18:29: 9 sends to 12: 0: 1 starts at 199 ms 646 us and finishs at 199 ms 651 us
18:29: 9 sends to 13: 0: 1 starts at 199 ms 651 us and finishs at 199 ms 656 us
18:29: 9 sends to 14: 0: 1 starts at 199 ms 656 us and finishs at 199 ms 661 us
18:29: 9 sends to 15: 0: 1 starts at 199 ms 661 us and finishs at 199 ms 666 us
18:29: 9 sends to 16:17: 1 starts at 199 ms 666 us and finishs at 199 ms 709 us
18:29: 9 sends to 17:17: 1 starts at 199 ms 709 us and finishs at 199 ms 752 us
18:29: 9 sends to 19:17: 1 starts at 199 ms 752 us and finishs at 199 ms 795 us
16:21: 3 sends to 17:22: 3 starts at 216 ms 567 us and finishs at 216 ms 573 us
17:21: 4 sends to 17:22: 3 starts at 216 ms 573 us and finishs at 216 ms 573 us
17:21: 3 sends to 17:22: 3 starts at 216 ms 573 us and finishs at 216 ms 579 us
19:26:22 sends to 17:22: 3 starts at 216 ms 579 us and finishs at 216 ms 584 us
19:27: 1 sends to 17:22: 3 starts at 216 ms 584 us and finishs at 216 ms 589 us
19:28: 1 sends to 17:22: 3 starts at 216 ms 589 us and finishs at 216 ms 594 us
17:22: 4 sends to 17:21: 5 starts at 219 ms 28 us and finishs at 219 ms 28 us
17:22: 4 sends to 17:21: 3 starts at 219 ms 28 us and finishs at 219 ms 30 us
17:22: 4 sends to 19:28: 1 starts at 219 ms 30 us and finishs at 219 ms 34 us
17:22: 3 sends to 17:21: 4 starts at 224 ms 580 us and finishs at 224 ms 580 us
17:22: 3 sends to 18:29:10 starts at 224 ms 580 us and finishs at 224 ms 614 us
6:21: 4 sends to 6:22: 3 starts at 229 ms 646 us and finishs at 229 ms 646 us
5:21: 3 sends to 6:22: 3 starts at 229 ms 646 us and finishs at 229 ms 652 us
8:26:22 sends to 6:22: 3 starts at 229 ms 652 us and finishs at 229 ms 657 us
8:27: 1 sends to 6:22: 3 starts at 229 ms 657 us and finishs at 229 ms 662 us
8:28: 1 sends to 6:22: 3 starts at 229 ms 662 us and finishs at 229 ms 667 us
6:22: 3 sends to 6:21: 4 starts at 237 ms 778 us and finishs at 237 ms 778 us
6:22: 3 sends to 7:29:10 starts at 237 ms 778 us and finishs at 237 ms 812 us
6:22: 3 sends to 8:30:20 starts at 237 ms 812 us and finishs at 237 ms 819 us

Bibliography

[AR80a] R. K. Arora and S. P. Rana. Analysis of the module assignment problem in distributed computing systems with limited storage. *Information Processing Letters*, 10(3):111–115, 1980.

[AR80b] R. K. Arora and S. P. Rana. Heuristic algorithms for process assignment in distributed computing systems. *Information Processing Letters*, 11(4,5):199–203, 1980.

[BFB91] C. Barmon, M. N. Faruqui, and G. P. Battacharjee. Dynamic load balancing algorithm in a distributed system. *Microprocessing and microprogramming*, 29(5):273–285, March 1991.

[Bok81] Shahid H. Bokhari. A shortest tree algorithm for optimal assignment across space and time in distributed processor system. *IEEE Transactions on Software Engineering*, 7(11):583–589, Nov. 1981.

[Bok87] Shahid H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publishers, 1987.

[Bur91] A. Burns. Scheduling hard real-time systems: A review. *Software Engineering Journal*, 6(3):116–128, 1991.

[CA93] Sheng-Tzong Cheng and Ashok K. Agrawala. Optimal replication of series-parallel graphs for computation-intensive applications. Technical Report CS-TR-3020, UMIACS-TR-93-4, Department of Computer Science, University of Maryland, College Park, Jan. 1993. to appear in *Journal of Parallel and Distributed Computing, 1994.*

[CA94] Sheng-Tzong Cheng and Ashok K. Agrawala. Scheduling of periodic tasks with relative timing constraints. Technical Report CS-TR-3392, UMIACS-TR-94-135, Department of Computer Science, University of Maryland, College Park, December. 1994. Submitted to *the 10th Annual IEEE Conference on Computer Assurance, COMPASS '95.*

[CA95] Sheng-Tzong Cheng and Ashok K. Agrawala. Allocation and scheduling of real-time periodic tasks with relative timing constraints. Technical Report CS-TR-3402, UMIACS-TR-95-6, Department of Computer Science, University of Maryland, College Park, January 1995.

[CC90] Y. Chen and T. Chen. Implementing fault-tolerance via modular redundancy with comparison. *IEEE Transactions on Reliability*, 39:217–225, June 1990.

[CHA94] Sheng-Tzong Cheng, Shyhin Hwang, and Ashok K. Agrawala. Mission-oriented replication of periodic tasks in real-time distributed system. Technical Report CS-TR-3256, UMIACS-TR-94-46, Department of Computer Science, University of Maryland, College Park, 1994. to appear in *the 15th IEEE International Conference on Distributed Computing Systems, ICDCS '95.*

[CHLE80] Wesley W. Chu, Leslie J. Holloway, Min-Tsung Lan, and Kemal Efe. Task allocation in distributed data processing. *IEEE Computer*, 13:57–69, Nov. 1980.

[CL87] Wesley W. Chu and Lance M-T Lan. Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers*, 36(6):667–679, June 1987.

[CO91] H.W.D. Chang and W.J.B. Oldham. A dynamic task allocation model for large distributed computing systems. *Journal of Microcomputer Applications*, 10(3):92–107, 1991.

[GH93] R. Gerber and S. Hong. Semantics-based complier transformations for enhanced schedulability. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 232–242, Raleigh-Durham, NC, Dec. 1993.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to Theory of NP-Completeness.* W.H. Freeman & Company, Publishers, San Francisco, 1979.

[GMK+91] Ó. Gudmundsson, D. Mossé, K.T. Ko, A.K. Agrawala, and S.K. Tripathi. Maruti: A platform for hard real-time applications. In K. Gordon, A.K. Agrawala, and P. Hwang (eds.), editors*, Mission Critical Operating Systems.* IOS Press, 1991.

[GPS] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computers.* To appear.

[HL89] C. C. Han and K. J. Lin. Job scheduling with temporal distance constraints. Technical Report UIUCDCS-R-89-1560, Department of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[HL92a] C. C. Han and K. J. Lin. Scheduling distance-constrainted real-time tasks. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 300–308, Phoenix, AZ, Dec. 1992.

[HL92b] C. C. Han and K. J. Lin. Scheduling real-time computations with separation constraints. *Information Processing Letters*, 42(5):61–66, May 1992.

[HMR+89] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22nd Hawaii International Conference on System Science*, pages 693–702, Jan. 1989.

[HS78] E. Horowitz and S.Sahni. *Fundamentals of Computer Algorithms*. Computer Science Publishers, 1978.

[HS92] Chao-Ju Hou and Kang G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *Proceedings of the 1992 IEEE 13th Real-Time Systems Symposium*, pages 146–155, Phoenix, AZ, 1992.

[JLT] Rong-Hong Jan, Deron Liang, and Satish K. Tripathi. A linear-time algorithm for computing distributed task reliability in pseudo two-terminal series-parallel graphs.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[LAMS92] Deron Liang, Ashok K. Agrawala, Daniel Mosse, and Yiheng. Shi. Designing fault tolerant applications in *maruti*. In *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, pages 264–273, Research Triangle Park, NC, Oct. 1992.

[LH91] Feng-Tse Lin and Ching-Chi Hsu. Task assignment problems in distributed computing systems by simulated annealing. *Journal of the Chinese Institute of Engineers*, 14(5):537–550, Sept. 1991.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Cumputing Machinery*, 20(1):46–61, Jan. 1973.

[Lo88] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on Computers*, 37(11):1384–1397, Nov. 1988.

[MLT82] P.R. Ma, E.Y.S. Lee, and M. Tsuchiya. A task allocation model for distributed computing systems. *IEEE Transactions on Computers*, 31:41–47, Jan. 1982.

[MM89] V.F. Magirou and J.Z. Milis. An algorithm for the multiprocessor assignment problem. *Operations Research Letters*, 8:351–356, Dec. 1989.

[MSA92] Daniel Mossé, M.C. Saksena, and Ashok K. Agrawala. Maruti: An approach to real-time system design. Technical Report CS-TR-2845, UMIACS-TR-92-21, Department of Computer Science, University of Maryland, College Park, 1992.

[PK84] C.C. Price and S. Krishnaprasad. Software allocation models for distributed computing systems. In *Proceedings of the 4th International Conference on Distributed Computing Systems*, pages 40–48, May 1984.

[PP82] C.C. Price and U.W. Pooch. Search techniques for a nonlinear multiprocessor scheduling problem. *Naval Res. Logist. Quart.*, 29:213–233, June 1982.

[PS89] D. T. Peng and K. G. Shin. Static allocation of periodic tasks with precedence constraints in distributed real-time systems. In *The 9th International Conference on Distributed Computing Systems*, Newport Beach, CA, June 1989.

[Ram90] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 108–115, Paris, France, 1990.

[RS94] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, Jan. 1994.

[RSL89] R. Rajkumar, L. Sha, and J. P. Lehoczky. An experimental investigation of synchronization protocols. In *Proceedings of IEEE Workshop on Real-Time Operating Systems and Software*, pages 11–17, May 1989.

[SA93] M. Saksena and A. K. Agrawala. Temproal analysis for hard-real time scheduling. In *Proceedings 12th International Phoenix Conference on Computers and Communications*, pages 538–544, Phoenix, AZ, March 1993.

[SdSA94] M. Saksena, J. da Silva, and A. K. Agrawala. Design and implementation of *maruti-ii.* Technical Report CS-TR-2845, Department of Computer Science, University of Maryland, College Park, 1994.

[SK91] David Stewart and Pradeep Khosla. Real-time scheduling of dynamically reconfigurable systems. In *Proceedings of the IEEE International Conference on Systems Engineering*, pages 139–142, Dayton, Ohio, Aug. 1991.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sept. 1990.

[SS91] A. K. Sarje and G. Sagar. Heuristic models for task allocation in distributed computer systems. *IEE Proceedings, Part E, [Computers and Digital Techniques]*, 138(5):313–318, Sept. 1991.

[SSA89] G. Sagar, Anil K. Sarje, and Kamal U. Ahmed. Task allocation techniques for distributed computing systems: a review. *Journal of Microcomputer Applications*, 12(2):97–105, April 1989.

[ST85] Chien-Chung Shen and Wen-Hsiang Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Transactions on Computers*, 34(3):197–203, March 1985.

[Sto77] Harold S. Stone. Multiprocessor scheduling with the aid of network flow algorithm. *IEEE Transactions on Software Engineering*, 3:85–93, Jan. 1977.

[SW89] Sol M. Shatz and Jia-Ping Wang. Models & algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38(1):16–27, April 1989.

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.

[Tow86] Don Towsley. Allocating programs containing branches and loops within a multiple processor system. *IEEE Transactions on Software Engineering*, 12:1018–1024, Oct. 1986.

[TT89] P. Thambidurai and K. S. Trivedi. Transient overloads in fault-tolerant real-time systems. In *Proceedings of the 1989 IEEE 10th Real-Time Systems Symposium*, Santa Monica, CA, Dec. 1989.

[XP90] J. Xu and D.L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.

[XP91] J. Xu and D.L. Parnas. On satisfying timing constraints in hard-real-time systems. In *Proceedings of the ACM SIGSOFT'91 Conference on Software for Critical Systems*, pages 132–146, Dec. 1991.

[YSA94] X. Yuan, M. Saksena, and A. K. Agrawala. A decomposition approach to real-time scheduling. *Real-Time Systems*, 6(1):7–36, Jan. 1994.

[ZRS87] W. Zhao, K. Ramamritham, and J. A. Stankovic. Scheduling tasks with resource requirements in a hard real-time system. *IEEE Transactions on Software Engineering*, 13(5):564–577, May 1987.