Template-Based Scheduling Algorithms
For Real-Time Tasks With Distance Constraints

by
Libin Dong
B.S., Tsinghua University of P.R.China, 1995
M.S., University of Pittsburgh, 1997

Submitted to the Graduate Faculty of
Arts and Sciences in partial fulfillment
of the requirements for the degree of

University of Pittsburgh
2001
This thesis was presented

by

Libin Dong

It was defended on

Feb 19th, 2001

and approved by

Prof. Rami Melhem

Prof. Daniel Mossé

Prof. Kirk Pruhs

Prof. Ragunathan Rajkumar

Committee Chairperson
A real-time system must generate computation results and transmit message packets in a timely manner. A large body of literature has been developed to guarantee the timely execution of periodic real-time tasks, where instance invocation rate is considered the most important timing constraint. However, some real-time tasks require relative timing constraints between the consecutive instances, which is called distance constraints. We propose a new task model, that combines the characteristics of both the periodic task model and the distance constraint task model. In the new model, a task has a rate requirement and a distance constraint.

A classical real-time system scheduling paradigm, template-based scheduling, is widely used in many time-critical systems. In this scheduling paradigm, an effective scheduling template is generated off-line, and the tasks (or message streams) are executed (or transmitted) cyclically according to the schedule. Template-based scheduling is also used in network systems, such as the media access control protocol TDMA (Time Division Multiplexing Access). We adopt template-based paradigm to solve scheduling problems for both computation tasks and communication tasks under the following three system configurations.

In this dissertation, we first study a system consisting of a static set of tasks, with all parameters of the tasks known in advance. We present a time slot allocation scheme to schedule a single resource, such as a processor or a broadcast bus. Moreover, we extend the scheme to schedule real-time traffic on crossbar switches and on WDMA (Wavelength Division Multiplexing Access) optical star couplers. In both the crossbar and the star coupler cases, input conflicts and output conflicts are taken into consideration in the scheduling algorithm, in addition to the rate requirements and the distance constraint specifications of the message streams. Simulation performance results of the algorithms are presented and analyzed. The results show that, by decoupling the rate requirements from the distance constraint specifications of the real-time message streams, high schedulability is achieved.

Our second scheduling problem is to satisfy rate requirements and distance constraints of dynamically arriving and departing tasks. Given a set of tasks already scheduled and a newly arriv-
ing task, three algorithms are presented to make greedy choices, heuristic but non-greedy choices, and optimal choices, respectively. The performances of the three algorithms are evaluated via simulation and compared in terms of time complexity, acceptance ratio and scheduling jitter.

The third scenario is a distributed system where isochronous message streams are transmitted from a source node to a destination node. In this scenario, end-to-end (ETE) delay is the main QoS (Quality of Service) concern. Since distance constraint is related to scheduling jitter, the algorithms derived to satisfy distance constraints are adapted to minimize the scheduling jitter. Once a scheduling template is generated at every node, the packets of a stream are delivered in an allocated time slot according to one of three delivery protocols proposed in this work. A worst-case ETE delay bound is derived for each protocol. The simulation studies show that by minimizing the scheduling jitter, we can achieve an end-to-end delay which is much better than the estimated one provided in the worst-case analysis.
Table of Contents

List of Tables ............................................................. vii
List of Figures ............................................................ viii

1 Overview ........................................................................... 1

2 Introduction and background .............................................. 4
  2.1 Task models ................................................................. 4
    2.1.1 Motivation of distance constraints ............................. 4
    2.1.2 Aperiodic task models and the related scheduling algorithms ...... 6
    2.1.3 Periodic task models and the related scheduling algorithms ....... 7
    2.1.4 Isochronous task models ........................................... 9
      2.1.4.1 Distance constraint task model and pinwheel algorithms ...... 10
      2.1.4.2 Distributed task models and scheduling algorithms ......... 13
  2.2 Scheduling paradigm ..................................................... 17
    2.2.1 Priority-driven ....................................................... 17
    2.2.2 Time-driven .......................................................... 18
  2.3 Research scope .......................................................... 20
    2.3.1 Scheduling paradigm ............................................... 20
    2.3.2 Task model .......................................................... 20
  2.4 Thesis contributions .................................................... 22

3 Time slot allocation schemes for static tasks ......................... 25
  3.1 Scheduling static tasks on a single resource ...................... 25
    3.1.1 Problem definition ................................................. 25
    3.1.2 Calculating the minimum size of the template .................. 26
    3.1.3 Scheduling algorithm Tpl_Sched .................................. 27
    3.1.4 Performance results of Tpl_Sched ................................ 31
  3.2 Scheduling static tasks on multiple resources ..................... 33
    3.2.1 Scheduling message streams in a crossbar switch .............. 34
      3.2.1.1 Problem definition ......................................... 34
      3.2.1.2 Calculating the smallest template size .................... 36
      3.2.1.3 Scheduling scheme for the TDMA switch .................. 39
      3.2.1.4 Performance results ....................................... 39
    3.2.2 Scheduling message streams in a WDMA star coupler .......... 41
  3.3 Adding minimum distance constraint in the task model .......... 44
    3.3.1 New task model with minimum distance constraint ............. 44
    3.3.2 Modification of Tpl_Sched to consider the minimum distance constraints 45
    3.3.3 Performance of the modified algorithm ....................... 47
List of Tables

3.1 Average $RDC$ v.s. Total Density ................................................. 33

4.1 Comparison of the three algorithms ............................................. 70
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Taxonomy of real-time tasks</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>The m-chain tree structure of the MUJSD problem [48]</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>The largest distance between two instances of a periodic task</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Illustration of a scheduling template</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>A Schedule for Example 2.1</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>The Tpl.Sched Algorithm</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>A Scheduling Example</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Success Ratio vs. RDC of the tasks</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>An example of a TDMA switching system.</td>
<td>35</td>
</tr>
<tr>
<td>3.5</td>
<td>Multi-equation fixed point scheme</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>The algorithm to schedule the input channels of a switch</td>
<td>40</td>
</tr>
<tr>
<td>3.7</td>
<td>Performance of $8 \times 8$ crossbar switch</td>
<td>41</td>
</tr>
<tr>
<td>3.8</td>
<td>Performance of $8 \times 8$ WDMA passive star coupler</td>
<td>43</td>
</tr>
<tr>
<td>3.9</td>
<td>The modified Tpl.Sched Algorithm</td>
<td>49</td>
</tr>
<tr>
<td>3.10</td>
<td>Performance comparison between the original and the modified algorithm</td>
<td>50</td>
</tr>
<tr>
<td>4.1</td>
<td>The $\text{min_instance_number}$ function</td>
<td>54</td>
</tr>
<tr>
<td>4.2</td>
<td>The Dynamic.greedy Algorithm</td>
<td>56</td>
</tr>
<tr>
<td>4.3</td>
<td>Relationship between $B$ and $B'$</td>
<td>57</td>
</tr>
<tr>
<td>4.4</td>
<td>An example of the allocation pattern generated by Dynamic.greedy.</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>The Dynamic.heuristic Algorithm</td>
<td>60</td>
</tr>
</tbody>
</table>
Chapter 1
Overview

In recent years, there has been an increased need for real-time computation and communication services in numerous application domains, such as automated control and manufacturing, robotics, telecommunication networks, multimedia and military systems. In these applications, predictable and guaranteed timeliness has become one of the critical components of the quality-of-service (QoS) requirements.

A hard real-time system should satisfy stringent timing constraints of individual tasks. In the periodic task model, a real-time task is invoked periodically, with its ready time and deadline defined relative to the period. A large body of literature has been developed to guarantee the timely execution of periodic real-time tasks. However, some real-time tasks require relative timing constraints between consecutive instances, which gives rise to the Maximum Distance Constraint (MXDC) task model and the Separation Constraint task model. The pinwheel scheduling problem is a specific instance of the MXDC task model. The motivation of the pinwheel problem originates from satellite transmission applications. Some satellites transmit information of one data stream for a fixed duration, then proceed with the information of another stream, and so on. The pinwheel problem formalizes how a ground station can be scheduled to receive from several satellites without loss of data. Much research work has been done for the pinwheel problem and its derivatives.

In practice, many real-time applications have both invocation rate requirements and distance constraints. We define a new task model that combines these two characteristics. The algorithms that are derived for periodic tasks can only satisfy the rate requirements. Though the pinwheel algorithms can meet the maximum distance constraints, they usually accomplish this by increasing the task’s rate, which will decrease the schedulability when the total system load is high. It is our goal to develop efficient algorithms that not only generate feasible schedules to satisfy the task requirements, but also highly utilize the system resources.

Template-based scheduling is a widely used scheduling paradigm in many time-critical systems, where time is divided into slots. An effective schedule is generated off-line, and the tasks are executed cyclically according to the schedule. In this scheduling paradigm, since the entire fu-
ture execution schedule is predetermined, the timing constraints are guaranteed. Moreover, since no on-line scheduling decision needs to be made, the run-time overhead can be very small. This is applicable to many real-time applications, such as mobile communication, where the scheduling overhead of priority-driven algorithms cannot be afforded. The motivation of this paradigm will be introduced in more detail in the next chapter. This research work focuses on applying template-based scheduling algorithms to our newly defined task model. The research scope includes scheduling algorithms for both static tasks and dynamic tasks, and protocols to guarantee end-to-end delay for distributed tasks. We say a set of tasks is a static task set if the parameters of all the tasks are submitted to the system before the system starts to execute the tasks. On the other hand, dynamic tasks refer to the tasks that dynamically arrive at and depart from the system. Distributed tasks are processed by several nodes in a distributed system, with data and results transmitted by a network system.

The rest of this thesis is organized as follows.

Chapter 2 introduces the background on real-time scheduling. We review different task models and their scheduling algorithms, and we introduce and compare two different scheduling paradigms, the priority-driven paradigm and the time-driven paradigm. A new task model is proposed to be used in this research work. Our methodology is template-based, which belongs to the time-driven paradigm.

This research work is focused on scheduling algorithms for three types of tasks which are related to distance constraints, namely, static tasks, dynamic tasks and distributed tasks with end-to-end concern. The following three chapters define the three problems, respectively, and present the solutions to each problem.

Chapter 3 addresses the problems of scheduling a static set of tasks. A template-based time slot allocation algorithm is presented to schedule static tasks with rate requirements and distance constraints on a single resource. It is extended to schedule a crossbar switch and a WDMA star coupler. Simulation results of the algorithms are presented and analyzed.

Chapter 4 considers the problem of allocating time slots to a new task in a system which allows tasks to dynamically arrive and depart. We present a greedy algorithm, an optimal algorithm and a non-greedy heuristic algorithm to solve the problem. The performance of the three algorithms are evaluated and compared.

Chapter 5 adapts the scheduling algorithm presented in Chapter 4 to minimize scheduling jitter at each intermediate node in a distributed system. Three delivery protocols NED, WED and Low-delay-NED are presented. The performance of the three protocols are compared in terms of end-to-end delay and implementation overhead. Simulation results show the performance improvement obtained by minimizing the scheduling jitter.
Chapter 6 concludes this research work and Chapter 7 proposes possible avenues to further extend and apply this research work in the future.
Chapter 2
Introduction and background

A lot of research work has been done on scheduling real-time tasks with various kinds of timing constraints. This chapter first introduces the previous research work based on different task models and the corresponding scheduling algorithms. Two scheduling paradigms are then described and compared. Based on the background introduced, we will define the new task model and the scheduling paradigm that we use in our research.

2.1 Task models

Tasks submitted to a real-time system are assumed to have known timing requirements and are called real-time tasks. Figure 2.1 gives a taxonomy of the most common real-time task models which are generally classified into two types. In the first type, an aperiodic task generates only one computation that must meet certain timing constraints. In the second type, tasks are invoked more than once, and each invocation is called an instance of the task. The repeatedly invoked tasks can be further classified into periodic tasks and isochronous tasks, based on whether the tasks are invoked in fixed period. Each class of tasks can be either preemptive or non-preemptive. A preemptive task can be interrupted during execution and later resumed. A non-preemptive task cannot be interrupted before it completes.

This research work concerns tasks with distance constraints, which are defined as the relative timing constraints between the finishing times of any two consecutive executions. We will discuss relationship between different task models and distance constraints during our review of the models and their scheduling schemes. The motivation of distance constraints is first explained.

2.1.1 Motivation of distance constraints

Scheduling problems with distance constraints were originally motivated by the performance requirements of a ground station that processes data from a number of satellites or mobile sensors [49]. Some satellites transmit information of one data stream for a fixed duration, then pro-
ceed with the information of another data stream. The following protocol is utilized to ensure that no data is lost. Each satellite may commence sending data at any time, but must repeat sending the same data for a specified interval. Assume that satellite $x$ specifies an interval of length $d_x$. Then the ground station can be assured not to lose any data from satellite $x$ by scheduling at least one instance to serve $x$ in any interval of length $d_x$, i.e., by making sure that the distance between any two consecutive slots assigned to serving satellite $x$ is smaller than or equal to $d_x$.

Similar to the satellite transmission situation, generation of telemetry formats also needs a certain distance constraints [60]. Telemetry is a technique that automatically measures and transmits the phenomenal parameters such as temperature, radiation, etc, to a remote recorder or observer. In order to prevent stale data, the transmission format of telemetry requires that every measurement parameter of the test article must be transmitted at a time that is no later than a certain threshold after the parameter is sampled. The threshold can be considered as a distance constraint between the sampling task and the transmission task.

Distance constraints are also typical characteristics of some real-time applications in embedded control systems. For example, the system that controls an automobile product line needs to schedule a certain number of painting jobs, each of which applies one layer of specific material on the body of a car. The time interval between applying two adjacent layers cannot be smaller than a minimum threshold and cannot be larger than a maximum threshold, according to the quality requirements. Thus, the minimum threshold and the maximum threshold specify the minimum distance constraint and the maximum distance constraint of the painting jobs, respectively.

Many real-time database transactions are tasks with minimum and maximum distance constraints. In a real-time database system, timing constraints are associated with transactions, and data are valid for specific time intervals [13, 92]. The transaction timing constraints can be defined as completion deadlines, start times, and periodic invocations as in the periodic task model. However, in addition to transaction timing requirements, data has time semantics as well. Data, such as sensor
data, stock market prices and locations of moving objects, all have semantics indicating that the recorded values are valid only for a certain time interval [99], which defines the absolute temporal constraints of a hard-real-time database transaction. Moreover, some transaction may have relative temporal constraints [6] when the transaction reads a set of data objects, the valid time interval of each data object is relative to the other data objects in the set.

2.1.2 Aperiodic task models and the related scheduling algorithms

A general aperiodic task model defines a fixed ready time and a deadline for each task. For preemptive aperiodic tasks, it is known that the *Earliest Deadline First* algorithm (EDF) is optimal for uniprocessor [51]. For non-preemptive tasks, the general problem of scheduling aperiodic tasks with different ready times and deadlines is known to be NP-complete [35]. However, if the ready times are all the same, the polynomial algorithm EDD (Earliest Due Date) is optimal [59]. There are many papers surveying the problems of deterministic scheduling with aperiodic jobs [40].

C.C. Han studied in his Ph.D. thesis [42] the *Job Scheduling with maximum Distance constraints* (JSD) problem where a job is a non-preemptive aperiodic task. A JSD problem consists of \( n \) jobs \( J_1, J_2, \ldots, J_n \) where each job \( J_i \) is characterized by a ready time, a deadline and an execution time. Moreover, it requires that the maximum distance between the finish times of two jobs \( J_i \) and \( J_j \) must be smaller than or equal to a certain constraint. The maximum distance constraint is equal to \( \infty \) if there is no distance constraint between \( J_i \) and \( J_j \).

It is shown in [48] that the general JSD problem is NP-complete, and so is the unit-time JSD (UJSD) problem where the job execution times are all one unit. The attention had then been restricted to a special class of the UJSD problem, namely the multi-level UJSD (MUJSD) problem. In a MUJSD problem, the job set is divided into multiple chains of jobs. In each chain, the header job has a deadline, and the other jobs have a constant distance constraint from their predecessors. Let us use the vertices of a graph to represent the jobs and its edges with weights to represent the maximum distance constraints between jobs. Figure 2.2 illustrates the definition of the MUJSD problem.

In Figure 2.2, the root node \( R \) is a dummy node, indicating the beginning of the schedule. Node \( R \) is connected with \( m \) header nodes of the \( m \) job chains, with the weight of each edge being equal to \( D_i \), the deadline of the \( i^{th} \) header node. In the \( i^{th} \) chain, there are \( n_i \) jobs and the maximum distance between every two consecutive jobs is \( C_i \). It is proved that even though the task structure is simplified a lot in this problem, the general MUJSD problem is still NP-complete. A polynomial optimal algorithm is given for a further simplification of the MUJSD problem where all \( C_i \)'s are the same, that is, all job chains have the same distance constraints.
Another type of relative timing constraints concerns the minimum distance between two consecutive instances, which is called separation constraint in [46]. In this problem, two instances cannot be closer than the separation constraint. Similar to the research results of the maximum distance constraint problem, C.C. Han has proved that most of the general forms of the separation constraint problem for aperiodic tasks are NP-hard, and he derived an optimal algorithm for a special form of the problem where the jobs have a forest structure, unit execution times and the same separation constraints [46].

2.1.3 Periodic task models and the related scheduling algorithms

Most research work for the periodic task model concerns preemptive tasks. A periodic task, $\tau_i$, is invoked periodically with a fixed period $P_i$ and a worst case execution time $C_i$. In the original periodic task model, the ready time and deadline of each instance coincide with the task’s period boundaries [72]. That is, in the $k^{th}$ period, the ready time for the $k^{th}$ instance of $\tau_i$ is $(k - 1) \times P_i$ and the deadline of that instance is $k \times P_i$. Thus, at the beginning of each period, a new instance of the task is generated and is immediately available for processing, and the processing of each task instance must be completed by the end of the period. Many real-time applications generate tasks of this type. For example, in a radar tracking application, a new input is sensed every time
a sweep is completed, triggering a computation that must complete before the input from the next sweep is available.

One of the two widely used algorithms for scheduling periodic real-time tasks is the Rate Monotonic Scheduling (RMS) algorithm which follows a fixed priority scheduling scheme with the priority of each task defined as $1/P_i$. The other one is the Earliest Deadline First (EDF) algorithm which is a dynamic priority scheduling scheme that assigns the highest priority to the task with the earliest deadline. It is shown in [72] that a set of $n$ tasks, $T = \{\tau_1, \tau_2, ..., \tau_n\}$, is schedulable by EDF if and only if the total utilization $U_T = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$. For RMS, a sufficient condition for schedulability is $U_T = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$. Later, Lehoczky et al. provided a necessary and sufficient schedulability test, known as the exact characterization for RMS [66]. In his Ph.D thesis [76], Mok proved that LLF (Least Laxity First) is also capable of achieving 100% utilization and hence is also optimal among dynamic policies.

Research has been done on variations of the original periodic task model in which the ready time and deadline of each task instance may not coincide with the period boundaries. For example, a popular extension to the original periodic model defines the characteristics of a task $\tau_i$ as $\tau_i = (C_i, D_i, P_i)$ where $D_i$ is the relative deadline that is smaller than or equal to $P_i$ [68, 70]. So the original periodic task model is a special case where $D_i = P_i$. The fixed priority algorithm RMS was extended in [69] and [5] by introducing the Deadline Monotonic (DM) algorithm and its schedulability test. Audsley et al. analyzed the worst-case response time in the DM algorithm in [3, 4]. Other research work studied the schedulability of EDF, the optimal dynamic priority policy, on this periodic task model. Leung and Merrill derived a feasibility test to check the system schedulability on the interval $[0, LCM]$ where $LCM$ is the Least Common Multiple of all the task periods [68]. Chetto and Chetto studied the localization and duration of idle times [20]. Baruah et al. proposed a more accurate feasibility test, showing that for a large percentage of task set instances it is not necessary to check the schedulability throughout the whole $LCM$, but only in a small interval [11]. The work in [95] further improved the feasibility test for EDF on the extended periodic task model by proving a even smaller test interval.

Baruah et al. further extend the periodic task model by adding an arrival jitter parameter to the $(C_i, D_i, P_i)$ characteristics of periodic tasks. The goal is to explicitly incorporate the uncertainty in the arrival times of individual instances [9]. Thus, the ready times of the instances of the periodic tasks are not exactly at the beginning of the periods, but within a jitter relative to the period boundaries.

Some other researchers have considered relaxing the task characteristics in order to achieve high utilization. For example, the work in [16] assumes that task periods are elastic and can be changed to provide different QoS, the work in [96] determines the periods for the tasks to optimize
system-wide performance, and the work in [93] discards selected task instances when the system is overloaded, which results in decreasing the frequency of the selected task.

![Diagram of task instances](image)

**Figure 2.3:** The largest distance between two instances of a periodic task

In conclusion, the most important characteristic of all periodic task models are the fixed task period that specifies the frequency of the task invocations. As will be introduced in the next chapter, our task model also has a frequency requirement. However, periodic task models are not designed to solve problems with distance constraints, as illustrated in Figure 2.3. Given a periodic task with a worst-case computation time $C_i$ and period $P_i$, the earliest finishing time of the $k^{th}$ instance can be at the beginning of its period, which is $(k - 1) \times P_i + C_i$. The $(k + 1)^{th}$ instance could have the latest finishing time at the end of its period, which is $(k + 1) \times P_i$. Thus, the maximum distance between two consecutive instances can be as large as $2P_i - C_i$.

### 2.1.4 Isochronous task models

Similar to periodic task models, isochronous tasks require repeated invocations. However, isochronous tasks just use a rate requirement to specify the invocation frequency, instead of the fixed period parameter in the periodic task model. The deadline of each isochronous task is usually loosely related to the invocation rate parameter.

There are many different task models that can be considered as isochronous task models. For example, some real-time systems require that an instance of an isochronous task must complete within a distance constraint relative to the finishing time of the previous instance. That is, the distance between the finishing times of two consecutive instances of a task must satisfy a certain condition. The distance constraint task model and related algorithms are reviewed in Section 2.1.4.1

In distributed systems, real-time communication applications, such as remote audio or video display in multimedia conferencing, can be classified as isochronous tasks, because a real-time message stream needs to send packets/frames repeatedly with a specified bandwidth. If we consider a message stream as a task, then the message packets are equivalent to the task instances. Without loss of generality, we will use the term “task” and “instances” to describe both computation and communication applications. It should be noted that some scheduling results, like EDF and RMS, only apply to fully preemptive task sets [72]. For communication applications, transmitting
message packets cannot be preempted arbitrarily. We introduce the distributed task models and related scheduling algorithms in Section 2.1.4.2.

2.1.4.1 Distance constraint task model and pinwheel algorithms

To our knowledge, only the maximum distance constraints for isochronous tasks have been studied. A distance constraint (DC) task model is defined in [41] where the maximum distance between any two consecutive instances of each task is specified. We will refer to this as the maximum distance constraint (MXDC) model to distinguish it from our meaning of distance constraint which concerns both the maximum distance and the minimum distance. A task \( \tau_i \) in the MXDC task model is characterized by a maximum distance constraint, \( \max D_i \), and a worst case execution time, \( C_i \). Similar to the task utilization of the periodic task model, the density of \( \tau_i \) [49] is defined as \( \rho(\tau_i) = \frac{C_i}{\max D_i} \).

As shown in Figure 2.3 in the previous section, the scheduling algorithms for the periodic model, EDF or RMS, are not applicable to schedule tasks with strict distance constraints. Given a periodic task with a worst-case computation time \( C_i \) and period \( P_i \), the maximum distance between two consecutive instances can be as large as \( 2P_i - C_i \). In order to use EDF or RMS to schedule a set of tasks of the MXDC task model, we have to transform each task \( \tau_i = (\max D_i, C_i) \) to a periodic task \( \tau_i' = (P_i, C_i) \) where \( P_i = (\max D_i + C_i) / 2 \). Obviously, the total utilization of the transformed task set is almost twice the total density of the original task set. So the scheduling algorithms of the periodic task model can not solve MXDC problems efficiently. Moreover, in the schedule generated by EDF or RMS, two consecutive instances can be as close as \( C_i \) if the previous instance is scheduled at the end of the period and the succeeding instance is scheduled at the beginning of the period. So the algorithms of the periodic task model are not suitable to schedule tasks with strict minimum distance constraints, either.

The maximum distance constraint scheduling problem is closely related to the pinwheel problem which is formally defined as follows [49]: Given a multi-set of \( n \) positive integers \( A = \{a_1, ..., a_n\} \), where \( a_1 \leq a_2 \leq ... \leq a_n \), the problem is to find a sequence (schedule) of symbols from \( \{1,2,...,n\} \) such that there is at least one symbol \( i \) within any interval of \( a_i \) entries (“slots”). For example, if \( A = \{3,4,4\} \), one solution sequence is \( \{1,1,2,3,1,1,2,3...\} \) where \( \{1,1,2,3\} \) is a subsequence that repeats forever. The pinwheel problem is equivalent to a special case of the maximum distance constraint problem where the execution time of each task, \( C_i \), is one time unit and the maximum distance constraint is an integer number of time units. The total density of the set \( A \) is defined as \( \rho(A) = \sum_{i=0}^{n} \frac{1}{\text{avg}j_i} \).

Although it is proved that pinwheel problem is NP-complete [49], a lot of research results have been derived for the pinwheel problem. It is proved in [49] and [18] that all instances of the
pinwheel problem with total density no more than 0.5 can always be scheduled, which can be easily seen from the intuition of the relation between the MXDC task model and the periodic task model described in the previous paragraph.

Another straightforward result in [49, 50] is that an instance $A = \{a_1, a_2, ..., a_n\}$ of the pinwheel problem with a total density smaller than or equal to 1 is schedulable if for $i < j$, $a_i | a_j$ where $x | y$ means $x$ divides $y$. Other research results concerning the special instances of the pinwheel problem show that if there are only two distinct numbers in $A$, then $A$ is schedulable as long as $\rho(A) \leq 1$. The upper bound density threshold is $\frac{5}{6}$ if $A$ only consists of three distinct numbers [49].

For a general pinwheel problem, several algorithms are derived by using a specialization technique to find a corresponding multi-set $B = \{b_1, b_2, ..., b_n\}$ such that for all $i$, $b_i \leq a_i$. Since $B$ is more strict than $A$ in terms of distance constraints, $A$ is schedulable if $B$ is schedulable. The specialization operation is defined as follows:

**Definition:** Let $S$ be a set of real numbers with values larger than or equal to 2. $A$ is specialized to $B$ with respect to $S$ based on $g$, or $A \rightarrow_g^S B$, if every $a_i \in A$ is reduced to $b_i = x g^j$ for some $x \in S$ and integer $j \geq 0$ such that $b_i \leq a_i$. The numbers in $B$ are called the reduction numbers. The variable $g$ is called the reduction base.

The purpose of the specialization is to transform $A$ to $B$ with tighter distance constraints, such that the total density of $B$ is still smaller than or equal to 1 and it is easier to schedule $B$. For example, if $A = \{3, 4, 5, 7\}$, $g = 2$ and $S = \{2, 3\}$, the specialization result is $B = \{3, 4, 4, 6\}$.

Most of the pinwheel algorithms, except the one given in [54], perform a base 2 specialization, that is, $g = 2$. Two classes of the specialization techniques have been studied. The first class is called single-number reduction where there is only one number in $S$, that is, $|S| = 1$. In this case, the reduced integer set $B$ consists of only multiples of each other, and the generated schedule is jitter-less, that is, the distances between any two consecutive instances of a certain task are the same. This technique has been used in a variety of schedulers in [18, 19, 41, 47, 71]. The second class of specialization techniques are called double-number reduction where $|S| = 2$, which is used in several schedulers in [18, 19]. To guarantee a feasible schedule for a pinwheel problem, the schedulability density thresholds derived in all the algorithms are 1/2, 13/20, 2/3, 0.696, or 0.7, depending on the scheduling algorithm used.

In [54], a specialization technique that uses an optimal base $g$ is derived, instead of $g = 2$ as in the other research work. Although this technique does not improve the schedulability density threshold, it broadens the pinwheel problem research field. From the description of the specialization, the new set $B$ is more conservative than $A$ in the sense that it provides tighter distance
constraints, and thus result in higher total density. So the schedulability conditions are sufficient, but not necessary conditions.

In order to efficiently specialize the original set \( A \) to the new set \( B \), the algorithms introduced in the previous paragraph must know all the task parameters to find the appropriate reduction numbers and reduction base. That is, the above algorithms are only suitable to schedule static tasks. If the above pinwheel-related algorithms are applied to schedule a dynamically arriving task, the specialization operation may find a new reduction number or a new reduction base, and thus result in a new set of distance constraints for the old tasks that have already started execution. In [45], the pinwheel algorithm is adapted to schedule dynamic tasks with distance constraints. Since the parameters of the future tasks are unknown, the reduction number is fixed to be \( S = \{2\} \) and the reduction base is also set to be \( g = 2 \). Thus, the distance constraint of every task will be specialized to be a power of 2. For example, a task requesting a distance constraint of 30 will be scheduled 16 time units apart after the specialization. Obviously, the utilization of the system is not efficiently allocated.

The pinwheel algorithms have been extended and applied to different kinds of applications. For example, the original pinwheel model is generalized and extended such that each task requires more than one time slot for computation, and tasks may be preempted at time slot boundaries [10]. The work in [44], [43] and [52] study time slot allocation schemes and network management issues for isochronous message streams with distance constraints in a metropolitan area network (MAN) using the Distributed Queue Dual Bus (DQDB) medium access control (MAC) protocol. A generalized model for real-time fault tolerant broadcast disks [8] is closely related to the pinwheel scheduling problem, and the pinwheel algorithms are adapted and applied to that problem. The work in [71] extends the pinwheel algorithm by defining blocking factors to deal with tasks that share resources in critical sections, and by adding a polling server [67] to handle aperiodic tasks. The work in [103] applies pinwheel algorithms to schedule isochronous message streams on a WDMA star coupler where optical wavelengths can be considered as multiple resources on which the streams should be scheduled, and input/output conflicts are considered in addition to the distance constraints. The algorithm in [103] uses a specialization operation with respect to \( \{2\} \) based on 2, that is, \( S = \{2\} \) and \( g = 2 \), and the message distance constraints are only transformed to powers of 2. So in the worst case, the total density can be doubled by the specialization, which may not be efficient in terms of bandwidth utilization.

The MXDC task model and the pinwheel algorithms have the following disadvantages.

- Since the specialization technique in the pinwheel algorithms generates a new set \( B \) that is more conservative than \( A \), it provides tighter distance constraints, and thus increases the total density. In this way, the resource is not efficiently utilized.
Pinwheel algorithms are not suitable to schedule dynamic tasks. The solution presented in [45] and [52] for dynamic message streams is not efficient, since the distance constraints are reduced to powers of 2.

Although the distance constraint implies the minimum invocation frequency of a task, there is no explicit rate requirement described in the MXDC task model.

### 2.1.4.2 Distributed task models and scheduling algorithms

A distributed computation or communication task starts its processing at a source node, processes and transmits the data along a set of intermediate nodes, and finishes the processing until the destination node is reached. End-to-end timing constraints, such as end-to-end delay and delay jitter, are important factors of the measurement of QoS. End-to-end delay measures the length of the time interval between the finishing time at the destination node and the starting time at the source node. Delay jitter is used to specify the delay variability. Different definitions of delay jitter are given. For instance, absolute jitter is defined as the absolute difference between the largest end-to-end delay and the smallest end-to-end delay among all the instances of a certain task [102]. Other research work defines delay jitter as the average or variance of all the instances’ end-to-end delays [108, 107]. Note that delay jitter is closely related to distance constraint. If the distance between the transmission times of every two consecutive packets is constrained within a range, then the delay jitter is also constrained accordingly.

In this section, we will review the models defined for distributed computation tasks and isochronous message streams.

**Distributed transactions**

In some multiprocessor systems, a pipeline of tasks are running on distributed processors to handle a set of fast-changing and short-lived data via sensors and actuators. In [58, 57], such a sequence of tasks is called a *distributed transaction*. For example, in a military avionics system, data is collected from different sensor devices. These data are sent to one or more processors for processing until the data reaches the decision node, where a Multi-Sensor Situation Assessment tasks is invoked periodically to evaluate new threats and decide whether the response to a certain threat is necessary.

In [58, 57], the model definition of such a transaction includes a sequence of processors to run the pipeline of tasks, the worst case execution time of each task, and a distance constraint that applies to all the tasks in a transaction. Therefore, on a certain processor, tasks of different transactions need to be scheduled to meet the distance constraints. The single-number reduction pinwheel algorithm is adapted to satisfy the distance constraints of distributed transactions on all
the nodes. Moreover, the end-to-end delay and jitter is evaluated. The performance of applying
pinwheel algorithm is compared with that of applying RMS and EDF in [55], where it is shown
that the pinwheel single-number reduction scheduling algorithms obtains a delay jitter equal to 0,
and a shorter end-to-end delay than that of RMS or EDF. The pinwheel algorithm has better QoS
performance because the task set is transformed to a set of harmonic tasks (define as tasks with
distance constraints that are multiples of each other) by the specialization technique. However,
this research work has a limitation. It assumes that all the transactions start from the same source
node, pass the same set of intermediate node and reach the same destination node [53]. This strict
assumption is not applicable to a general distributed system. Moreover, since the specialization
scheme needs to find a common reduction number for all the tasks on different nodes, the reduction
number may not be optimal for a single node. The original distance constraints of the tasks can be
largely reduced. Thus, the resource utilization of the system cannot be highly utilized.

Isochronous message streams

Various message stream models have been proposed to specify the characteristics of traffic in a lot of research work. For example, [44] and [105] characterized an isochronous message stream by the minimum message inter-arrival time which implies the maximum transmission rate, the maximum message transmission time which is equivalent to the worst case execution time, and the delay bound which indicates the processing deadline of each packet. Some other research work on real-time communication apply the Linear Bounded Arrival Process (LBAP) to specify their message streams [31, 32, 30, 23]. A LBAP [1] is a mechanism which specifies the maximum number of packets an application can generate over any interval of time (which is defined by the maximum burst size), an interval and the number of packets generated in that interval. Rather than specifying one rate, the Deterministic Bounding Interval-Dependent (D-BIND) model [64] uses a family of rate-interval pairs where the rate is a bounding rate over the corresponding interval length [106]. It is shown in [98] that different message models can be transformed to each other. Traffic regulation algorithms are derived to enforce the parameters defined in the message model, such as the leaky bucket algorithm [101] for the LBAP model.

Real-time message streams require end-to-end QoS guarantees, including end-to-end delay, jitter, throughput, etc. In order to guarantee the required performance, a connection is set up for a specific message stream at all the intermediate nodes along the path from the source to the destination. During connection establishment time, an admission control scheme is executed at each intermediate node to reserve network resources based on the specification of the message model and QoS requirement. If a stream is accepted, a scheduling algorithm at each intermediate node decides how to multiplex the packets of all the streams that are sent on the same output link. QoS requirements of the streams are enforced by the scheduling algorithm.
Scheduling policies can be classified into work-conserving algorithms and non-work-conserving algorithms [104]. A scheduling policy is work-conserving if the network resource is never idle when there are packets backlogged. The advantage of work-conserving policy is the efficient and high utilization of the bandwidth. On the other hand, a non-work-conserving policy allows the network resource to be idle even when there are packets backlogged. The advantage of non-work-conserving policy is that it can reshape the traffic inside the network to reduce traffic burstiness, or to maintain the traffic arrival pattern.

One example of work-conserving policy is the virtual clock scheduling algorithm [109, 110]. It eliminates the interference among message streams by introducing a virtual clock concept. Each message stream has two state variables. A virtual clock value is the time stamp in the packet header, indicating the progress of the stream. The other state variable is a tick value, which represents the average inter-packet spacing according to the average serving rate that is reserved for the message stream. That is, the larger the reserved rate, the smaller the tick value. Once a packet arrives at a certain node, the virtual clock value in the packet header is increased by the tick value of this message stream. The node transmits the backlogged packets in ascending order of the virtual clock stamp. Therefore, the message stream which progress the least and has the largest rate will be given the highest priority. It is shown in [33] that if the connection of a message stream is idle for a while, then the packets of this streams will have the highest priority and delay the packets of all the other streams, because the virtual clock of the idle message stream is much smaller than the other streams that continuously generate packets, no matter what the reserved rates are. Thus, the reserved service rates of the message streams can be violated indefinitely, and the end-to-end delay bound for the virtual clock algorithm can be infinity for some connections [39].

Weighted fair queuing (WFQ) [25] is another work-conserving scheduling algorithm. It is a discipline based on the fair queueing algorithm [77, 78] where separate queues are provided to protect each individual message stream from bursty connections. It is derived from the idealized Generalized Processor Sharing (GPS) policy which can split the bandwidth into infinitely small shares among multiple connections[83, 84, 85]. GPS achieves complete fairness, because at any time, GPS services all the non-empty queues simultaneously in proportion to their service rates. WFQ emulates the bit-by-bit GPS discipline in a packet-by-packet transmission. It computes the finish time of the packets as if the ideal GPS policy was applied, and then transmits the packets in increasing order of the finish times. Variations of WFQ are derived to improve the fairness in the worst case in [12, 100]. The end-to-end delay bound of WFQ algorithms is analyzed. It is shown that the delay bound of a packet at each intermediate node is equal to $\frac{\sigma + S_m}{R}$, where $\sigma$ is the burst size, $S_m$ is the maximum packet size, and $R$ is the service rate assigned at the node. Similar to the GPS policy, WFQ algorithm dynamically calculates the service rate of a certain message stream.
according to the proportion of weight shares. For example, if message streams $A$ and $B$ each has a weight share of 10, then each of the two streams is assigned half of the bandwidth if there is no other active message stream at the node. When a new message stream with a weight share of 10 requests transmission at the same node, then the proportion of the bandwidth is decreased to one third for stream $A$ and $B$. Therefore, in a system that allows message stream to dynamically arrive and depart, the service rate of a message stream at a given node may have a large variance, which leads to the variance of delay at the node. Since the jitter can be large when WFQ is applied to a dynamic system, it is not appropriate to solve our problem addressed in Chapter 5, where the applications require 0 jitter in a dynamic distributed system.

Although work-conserving scheduling policy efficiently utilizes network bandwidth, it cannot maintain traffic characterization, and traffic can become bursty inside the network. Some non-work-conserving algorithm, such as the stop-and-go policy [36, 37, 38], deliberately delay the transmission of a packet, in order to maintain the traffic arrival pattern. In the stop-and-go algorithm, time is divided into frames of constant size $T$. Each message stream is characterized by $(r, T)$, indicating that the number of bits of packets arriving during each frame of length $T$ is no more than $rT$ bits. The packets arrived in a certain frame will always be transmitted in the next frame on the outgoing link. In this way, as long as the admission control ensures that the number of bits to be transmitted is no more than the frame size, the $(r, T)$ characteristics can always be maintained throughout the network. It is shown that the end-to-end delay bound of stop-and-go is $2HT$, where $H$ is the number of links traversed by the connection from the source node to the destination node. The end-to-end jitter bound is $2T$. Since the delay bound is proportional to the frame size $T$, one way to reduce delay is to choose a small frame size. However, it is pointed out in [62] that frame size $T$ and the incremental step of bandwidth allocation cannot be simultaneously decreased. A reduction in one will lead to a proportional increase of the other. As we will show in the latter chapters, our algorithm is based on a concept of template, similar to the frame in the stop-and-go policy. However, the scheduling granularity is different between our algorithm and the stop-and-go policy. Since we schedule each packet in a template, and stop-and-go schedules all the packets of a stream in a frame, we can better reshape the traffic, and achieve smaller end-to-end delay as shown in Chapter 5.

A typical non-work-conserving policy is the Time Division Multiple Access (TDMA) scheme, where time is equally divided into successive frames/templates, each frame consists of a fixed number of slots. Packets are of the same length, and the transmission time of one packet is equal to the length of one time slot. The time slots in a frame are assigned to message streams based on the allocation scheme, and the frame is cyclically applied to control the transmission of the packets. In the basic TDMA scheme, each message stream has exactly one slot in the frame
Variations of TDMA scheme have been widely used in various network architectures, including wireless network and optical network. For example, [65] allocates time slots according to the service rate requirements of the message streams, and tries to space out the time slots of the same stream in the frame. Our template-based algorithms presented in this research work can be applied to schedule time slots in a TDMA frame. In addition to the rate requirement, we explicitly consider distance constraint and jitter in our scheduling policies.

2.2 Scheduling paradigm

In current real-time systems, there have been two basic approaches to the overall design of hard real-time systems: the priority-driven architecture and the time-driven architecture. Since this research work adopts template-based scheduling schemes, which belongs to the time-driven paradigm, we will focus on the related background in this section.

2.2.1 Priority-driven

In the priority-driven architecture, the scheduler and the task dispatcher choose the ready task with the highest priority to execute at any time. As introduced earlier, the priority-driven systems can be further classified by static priority based, such as RMS, and dynamic priority based, such as EDF. The advantages of the priority-driven algorithms are listed as follows.

- Many priority-driven algorithms have simple admission control conditions which are usually utilization thresholds. As long as the total task utilization does not exceed a threshold, the schedulability is guaranteed, even if a new task arrives or some individual task changes its parameters. So the admission control overhead is small, and the scheme is flexible. It should be noticed that this is not always true. Some priority-driven algorithms have complicated admission control method, such as the exact characterization for RMS or for the deadline monotonic algorithm.

- The priority-driven paradigm is flexible, in the sense that any change to the system does not influence the scheduling policy.

The disadvantages of the priority-driven paradigm are listed as follows.

- When a context switch occurs, the scheduler needs to extract the task with the highest priority for the next execution, which leads to a run time scheduling overhead that cannot be ignored in real-time systems.
• In a priority-driven system, especially the static priority based systems, an ill-behaved high priority task can starve the low priority tasks and destroy their timeliness [75].

2.2.2 Time-driven

Time-driven is a classical real-time system scheduling paradigm, and is widely used in many time-critical control systems and communication environment. In this paradigm, an effective schedule of finite length is generated off-line according to the timing requirements of the tasks. This schedule is also called a ‘cycle’ [73], a ‘frame’ [74] or a ‘template’ [26], because the tasks are dispatched and executed according to the schedule, and the schedule is applied repeatedly along the time-line.

Though a lot of research results have been derived for the priority-driven paradigm, there are many applications where the time-driven paradigm is more appropriate. For example, in [17], time-driven scheme is always used to schedule a switch that connects a number of input channels with a number of output channels. A scheduling algorithm has to avoid conflicts among the input channels and output channels, and to satisfy the timing constraints of the real-time traffic. In this case, it is very hard to define the priorities for the communication tasks. Another example is when tasks are executed on distributed nodes whereas the scheduling decision is made centrally. The run-time scheduling overhead includes sending the scheduling decision to the remote nodes, which may require an unacceptable delay. For instance, several aircrafts may share a radio frequency to communicate with the ground station. Since information of all the traffic parameters can be owned by the ground station only, the scheduling decision is made by the ground station centrally. Obviously, it is not practical for the ground station to use the priority-driven scheme and send the dispatching command every time a scheduling decision is made. In this case, time-driven is a good choice in which the scheduling template is broadcast to all the aircrafts, and each aircraft communicates according to the time slot allocated to it in the template. The advantage of the time-driven paradigm over the priority-driven paradigm are listed as follows.

• Since the entire future execution is predetermined by the scheduling template, the timing constraints of the tasks are guaranteed.

• Because the scheduling template is generated off-line, the run time schedule overhead is constant.

• Time-driven schemes do not have the difficulty of mapping timing constraints into a set of priorities. The major problem comes from the fact that most kernels have a limited number of priority levels, whereas task deadlines can vary in a much wider range [15].
Time-driven methods usually schedule a task to run until a functional unit is completed. Moreover, the context switches between tasks are deterministic and predictable. So context switch overhead may be smaller than the priority-driven architecture, where a task can be preempted whenever a higher priority task is ready [2].

The main disadvantage of time-driven paradigm is the potential under-utilization of resources. If a time interval is reserved for an instance of a certain task, and the instance is idle, then the time interval will be wasted in a time-driven system. However, this problem can be overcome by slight modification of the architecture. In real-time systems, not all the tasks are real-time tasks with hard timing constraints. The non-real-time tasks are called best-effort tasks in some research work. In order to improve the efficiency of a time-driven paradigm, the reserved time interval that is idle can be used to execute the best-effort tasks. This modified time driven scheduling scheme is implemented in hardware by Marconi (previously Fore Systems), where a time slot in a TDMA template is used to transmit an ATM cell of a certain CBR (Constant Bit Rate) stream, or a cell from the queue of the UBR (Unspecified Bit Rate for best-effort traffic) stream if the specific CBR queue is empty.

When time-driven paradigm is applied to a distributed system, time synchronization is required among the distributed nodes. Much research work has been done to provide guarantees for synchronization in a certain range. Some of the work proposed software solutions that achieve time synchronization by time stamp exchanging protocols [74, 7, 22]. Some other systems adopt hardware solutions, such as using a specific signaling line for synchronization purpose [94].

Because the time-driven scheduling paradigm can efficiently guarantee the timeliness of the real-time tasks, it is popular in many real-time embedded systems. For example, in [56], a cyclic executive scheme is used to schedule tasks in an industry automatic controller system.

As pointed out previously, a popular application of the time-driven paradigm in communication systems is the TDMA protocols. A real-time operating system, MARS, uses TDMA to implement its network system, in order to obtain predictable communication behavior [34]. Moreover, TDMA is widely used for optical and wireless networks or satellite communication to concentrate traffic from many low bandwidth sources into high speed channels. One example is the innovative Bluetooth wireless technology that allows users to make both voice and data connections between various wireless devices. The Bluetooth technology supports up to 7 ‘slave’ devices to be connected with a ‘master’ radio in one device. TDMA is applied to allocate time slots to the master and slaves according to their service properties [14].
2.3 Research scope

Investigating real-time scheduling is a multi-aspect problem because of the many different types of timing constraints of the tasks and the various solution approaches to follow. To clarify the scope of this research, this section outlines the scheduling paradigm, and defines the task model used in the research.

2.3.1 Scheduling paradigm

In this research work, we focus on studying scheduling algorithms in the time-driven real-time systems, such as systems that adopt the cyclic executive control policies or the TDMA network protocols to guarantee the timeliness of the tasks. We define our approach as the template-based scheduling paradigm, where the time is divided into equal length slots, and a scheduling template consists of a fixed integer number of time slots. The length of a slot is equal to a unit processing time. So the representations of timing constraints and the parameters of the real-time tasks are in terms of integer numbers of time slots.

Let $R_{k}\rightarrow L$ denote the \textit{SNHP} instance of task $R_k$. We use $\text{location}(i, j)$ to denote the location of the time slot that is allocated to $R_{k}\rightarrow L$. The real-time tasks are scheduled within a template of size $T$, that is, there are $T$ time slots in the template. Assume that there are $n_i$ instances of task $\tau_i$ in the template. Since the allocation pattern of the template is applied repeatedly to control the dispatching of the tasks, it is obvious that $\text{location}(i, j + n_i) = \text{location}(i, j) + T$, as illustrated in Figure 2.4. In this example, the template size $T = 6$. The number of instances of task $\tau_i$ is $n_i = 2$. When the scheduling template is applied cyclically, we can see that $\text{location}(i, 3) = \text{location}(i, 1) + 6$, and $\text{location}(i, 4) = \text{location}(i, 2) + 6$.

![Figure 2.4: Illustration of a scheduling template.](image)

2.3.2 Task model

In this research work, we will address the scheduling problems of the distance constraint task model. In this task model, an instance of a periodic task or an isochronous message stream must complete within a certain timing constraint relative to the finishing time of the previous instance.

We assume that the execution time of every task is equal to one time slot. This assumption is realistic in many practical applications. For example, in some communication architecture such
as ATM network, packets/cells of all message streams are of the same size, and thus the time to transmit every packet is the same. Each time slot is allocated to transmit a packet of a unique message stream, which is equivalent to an instance of a task. In order to keep the terminology uniform, we will also use $\tau_i$ to represent the communication task of the transmission of the $i^{th}$ message stream in a communication application, and $\tau_{i,j}$ to represent the $j^{th}$ packet of the $i^{th}$ message stream.

The original distance constraint task model does not have an explicit parameter that indicates the frequency of the task invocation. However, the average invocation rate is an important requirement. Suppose the size of a template is $T$ time slots, the average transmission rate decides the number of time slots that should be allocated to the message stream. For example, Link-16 is a radio communication system designed for tactical information distribution systems, and it applies a TDMA protocol [24]. Each TDMA template in Link-16 consists of $T$ time slots. A message stream that requires an average service rate of $\frac{1}{15}$ of the total bandwidth must occupy $\frac{T}{15}$ time slots in the template.

In our task model, we consider tasks that need both the rate and the distance characteristics. For instance, in an interactive TV network and video-on-demand applications, along a network link of bandwidth $B$, an isochronous video stream $\tau_i$ may require to transmit video frames at a rate of at least $R_i \times B$, where $R_i$ is specified as a proportion of the total link bandwidth. So, the average distance between the transmission of consecutive frames must be smaller than or equal to $\frac{1}{R_i}$ time slots to satisfy the rate requirement. But the distance between consecutive video frames can tolerate being larger than $\frac{1}{R_i}$, as long as it does not exceed some maximum value, say, $max D_i$ time slots, in order to maintain the human perception of the video and to avoid jagged display. If we cast this application into a pinwheel problem, the integer that indicates the maximum distance constraint, $a_i$, should be set to $\frac{1}{R_i}$ in order to maintain the rate requirement. The scheduler for the pinwheel problem will either succeed in finding a schedule with $a_i = \frac{1}{R_i}$ or it will fail to find a schedule, even though there may exist a feasible schedule with the more relaxed maximum distance constraint specification of $max D_i$ ($max D_i \geq \frac{1}{R_i}$). Moreover, the human perception condition also requires that two consecutive video frames cannot be transmitted too close to each other, which specifies a minimum distance constraint. The pinwheel algorithms cannot guarantee minimum distance constraint.

The advantage of decoupling the rate requirement and the distance constraint is further clarified in the following example.

**Example 2.1:** Assume that a message stream, $\tau_1$, requires a transmission rate of $1/2$ of the bandwidth, that is, on average one time slot is required to be allocated to $\tau_1$ in every two time slots. Another message stream, $\tau_2$, requires a transmission rate $1/3$; and a third stream, $\tau_3$, requires
a transmission rate of 1/6. If this example is translated to a pinwheel problem of three streams with requirement {2, 3, 6}, all pinwheel scheduling algorithms in [49, 18, 19, 54] fail to find a schedule and will reject the set of message streams. However, the scheduling template in Figure 2.5 shows that, by relaxing the maximum distance constraint of \( \tau_2 \) to 4, instead of 3, there exists a feasible schedule with a template of size 6 which satisfies the rate requirement of all three message streams. This can be seen by repeating the template and observing that the distance between \( \tau_{2,2} \) and \( \tau_{2,3} \) is 4. In the schedule of Figure 2.5, the maximum distance between any two instances of \( \tau_1 \) is 2. The maximum distance is 4 for \( \tau_2 \) and 6 for \( \tau_3 \).

![Figure 2.5: A Schedule for Example 2.1](image)

We propose a new task model that includes both rate requirement, as the period parameter in the periodic task model, and distance constraint, as in the MXDC task model. In our task model, a task \( \tau_i \) is characterized by \( \tau_i = (\text{avg}D_i, \text{max}D_i), (\text{avg}D_i \leq \text{max}D_i) \), where \( \text{max}D_i \) is the maximum distance constraint, and \( \text{avg}D_i \) indicates the average rate requirement of \( \tau_i \) in terms of the average distance between two consecutive instances of \( \tau_i \). That is, \( \text{avg}D_i = 1/R_i \). For example, if \( \tau_i \) requires 20% of the CPU utilization or network bandwidth, i.e. \( R_i = 0.2 \), then \( \text{avg}D_i = \lceil \frac{1}{0.2} \rceil = 5 \), which means that on average one of every five time slots should be assigned to \( \tau_i \). Since we use the template-based scheduling paradigm as our solution approach, the average rate requirement can be enforced in the interval of length \( T \), which is the size of the scheduling template. Thus, the number of instances of \( \tau_i \) in the template, \( n_i \), must be at least \( \lceil \frac{T}{\text{avg}D_i} \rceil \). Following the terminology of the pinwheel problem [49], we define the density of the task \( \tau_i \) as \( \rho(\tau_i) = 1/\text{avg}D_i \) and the total density of a set of \( k \) tasks, \( \mathcal{T} \), as \( \rho(\mathcal{T}) = \sum_{i=0}^{k} \frac{1}{\text{avg}D_i} \).

The task model defined in this section is the basic model that reflects the scope of this research work. In the later chapters, the task model will be adapted and extended according to the specific problems and assumptions.

### 2.4 Thesis contributions

The first contribution of this thesis is the new task model that decouples the rate requirement and distance constraint. In the new model, a task has a rate requirement which represents the frequency of the task invocations, similar to the period requirement in the periodic model, and distance constraints, similar to the MXDC (Maximum Distance Constraint) model. Moreover,
the distance constraint specification takes both maximum and minimum distance constraints into consideration.

The second contribution of this thesis is the algorithms that are presented in Chapter 3 to schedule a set of static tasks both on a single resource and in a system with multiple resource constraints, such as a crossbar switch and a WDMA optical star coupler. Since the requirements of the static tasks are known before hand, the minimum template size is calculated in order to generate an efficient schedule. We use the fixed point scheme to calculate the smallest size of a scheduling template for a single resource, according to the rate requirements of the tasks. The fixed point scheme is extended to satisfy the rate requirements of the tasks over multiple channels. In each time slot of the template, the algorithms schedule a task instance which is ready, and which has the highest priority. The readiness and priority of a task is defined according to the definition of a schedule window for each task instance. The schedule window of each instance is calculated relative to the previously allocated instances, taking into consideration the distance constraints in both the forward direction and the backward direction. In the research of the crossbar switch and WDMA star coupler cases, input conflicts, output conflicts and wavelength conflicts are taken into consideration in the scheduling algorithm, in addition to the rate requirements and distance constraint specifications of the message streams. Simulation performance results of the algorithms are presented and compared to the pinwheel algorithms and their derivatives which tie the maximum distance constraint to the average transmission rate of a task. The results show that higher schedulability is achieved when the rate requirements and distance constraint specifications of the real-time tasks are decoupled, especially when the system workload is high.

The third contribution of this thesis is the algorithms presented in Chapter 4 to solve the scheduling problem in a dynamic system where tasks dynamically arrive and depart. Three different scheduling algorithms are derived to allocate the vacant time slots in a template to the instances of a dynamic task by making greedy choices, non-greedy heuristic choices and optimization choices, respectively. The greedy algorithm can efficiently find a feasible schedule for the new task if there exists one. However, the greedy algorithm may generate large scheduling jitter, which can lead to rejecting future arriving tasks, because of the clustered distribution pattern of the remaining vacant time slots. The optimization algorithm transforms the original scheduling problem to an equivalent graph problem, and minimizes scheduling jitter via finding the shortest path in the graph. Thus, the optimization algorithm has the smallest scheduling jitter and the largest acceptance ratio among the three algorithms. However, the time complexity of the optimization algorithm is costly, especially when the system is lightly loaded. The heuristic algorithm has an efficient time complexity that is the same as that of the greedy algorithm. Moreover, similar to the optimization algorithm, the heuristic algorithm tries to distribute the instances of a certain task as uniformly as possible. Although the
heuristic algorithm is not locally optimal for the current task, it combines the advantages of both
the greedy algorithm and the optimization algorithm. Simulation results show that the performance
measurements of both scheduling jitter and acceptance ratio of the heuristic algorithm are very close
to those of the optimization algorithm.

The fourth contribution of this thesis is the scheduling algorithms and delivery proto-
cols presented in Chapter 5 for distributed communication applications with specific servicing rate
requirement. End-to-end delay is the main concern. In order to improve end-to-end delay, the pro-
posed scheduling algorithms minimize scheduling jitter and satisfy the average transmission rate of
a dynamically arriving stream at each intermediate node. Once the stream is successfully scheduled
and accepted, the delivery protocols decides how to transmit packets at each intermediate nodes
according to the allocated time slots. The destination node applies a certain start-up scheme to de-
cide when to start processing the received packets such that a continuously regular service rate can
be achieved. The first protocol presented is NED, standing for No Extra Delay at the intermediate
nodes. NED decides that each intermediate node sends every packet of a message stream immedi-
ately in the next available time slot that is allocated to this stream according to the schedule. This
protocol completely relies on the destination node to smooth the jitter before it starts to process
the packets. In the second presented protocol WED (With Extra Delay at the intermediate nodes),
the intermediate nodes may delay the arriving packets for a certain time, even if there are available
allocated time slots for the packet. The objective is to smooth the traffic on its route and reduce the
start-up delay at the destination node. In a third protocol, Low-Delay-NED, the intermediate nodes
deliver packets with no extra delay, as in the NED scheme, but the destination node has an efficient
start-up scheme, as in the Low-Delay-NED protocol. The simulation performance shows that WED
and Low-Delay-NED provide smaller end-to-end delay than NED. Moreover, our presented algo-
rum that minimizes scheduling jitter at the intermediate nodes largely improves the end-to-end
delay over the other algorithms that only consider average service rate.
Chapter 3

Time slot allocation schemes for static tasks

In this chapter, we focus on real-time systems consisting of static tasks, where the parameters of all the tasks are known before the system is activated. Some embedded control systems, such as the avionics system DEOS (Digital Engineer Operating System) [29], explicitly define the parameters of all the tasks according to the functionality of the applications. The main concern of the system is to reserve resources in advance and to guarantee the timeliness of the tasks. Therefore, such a system is designed to run static tasks.

A scheduling algorithm is first presented in Section 3.1 to schedule a set of static tasks on a single resource. The scheduling policy is then extended in two ways. In Section 3.2, the algorithm is applied to schedule isochronous message streams in a crossbar switch and a WDMA star coupler, where the input and output channels are considered as multiple resources with special constraints. In Section 3.3, the basic task model is extended to include a minimum distance constraint, and the original algorithm is adapted to satisfy the additional timing constraint.

Schedulability is the most important performance measurement for static task scheduling, which reflects the capability of an algorithm to find a feasible schedule. For a set of static tasks, a feasible schedule is found if the timing constraints of all the streams in the set are satisfied. We will use simulation results to show that the decoupling of distance constraint and rate requirement helps to increase the schedulability.

3.1 Scheduling static tasks on a single resource

3.1.1 Problem definition

In this section, we consider the problem of scheduling static tasks on a single resource, such as a single processor in a control system, or a single link in a point-to-point network. For example, this scheme can be applied to schedule the backbone bus on a broadcast network. Each node that is connected to the backbone bus will send and receive message streams based on a time
slot schedule. The schedule guarantees that there is no collision and all message streams on each node can meet their rate and maximum distance constraint specifications.

Since the pinwheel problem which decides whether a set of static tasks is schedulable subject to distance constraint is NP-complete, it is also NP-complete to satisfy both distance constraint and average transmission rate [79]. In this section, we propose a heuristic template-based algorithm to allocate the time slots to a set of static tasks that share a single resource.

We use \( \text{location}(i, j) \) to denote the location of the time slot that is allocated to the \( j^{th} \) instance of task \( \tau_i \), namely, \( \tau_{i,j} \). The problem is formally defined as follows.

**Problem statement:** Given a template of size \( T \) and a set of \( k \) tasks \( T = \{ \tau_1, \tau_2, ..., \tau_k \} \) where \( \forall i \in [1, k] \), each task is characterized as \( \tau_i = (\text{avg}D_i, \text{max}D_i) \), the problem is to schedule the tasks in the template such that both the rate requirements and distance constraints of all the tasks are satisfied, that is, the following conditions are true.

1. \( n_i \geq \left\lceil \frac{T}{\text{avg}D_i} \right\rceil \), where \( n_i \) denotes the number of instances of task \( \tau_i \) in the template.
2. \( \text{location}(i, j + n_i) = \text{location}(i, j) + T \).
3. \( \forall j \geq 1, \text{location}(i, j + 1) - \text{location}(i, j) \leq \text{max}D_i \).

The first condition guarantees that there are at least \( \left\lceil \frac{T}{\text{avg}D_i} \right\rceil \) instances in the template of size \( T \) for every task \( \tau_i \), such that the rate of \( \tau_i \) is at least \( \frac{1}{\text{avg}D_i} \). The second condition is the characteristic of the template based scheduling, illustrated in the previous chapter in Figure 2.4. The third condition makes sure that the distance between any two consecutive instances is constrained by \( \text{max}D_i \). Note that since the template is applied repeatedly, the distance between the last instance in the current template and the first instance in the next template should also be constrained, which is called *inter-template distance constraint*. The constraint concerning two instances within the same template is called *intra-template distance constraint*.

In the problem statement, we need to know the template size \( T \). Because the parameters of the static tasks are known beforehand, we can calculate the smallest possible template size that satisfies the requirements, as discussed in the next section.

### 3.1.2 Calculating the minimum size of the template

Because a small template size decreases the time complexity of the scheduling algorithm, simplifies the runtime processing control, minimizes the memory space used to store the schedule, and enforces the rate requirements over a short time interval, we find the minimum template size which satisfies the rate requirements of all the tasks.
For a set $\mathcal{T}$ of $k$ tasks, a necessary condition to find a feasible schedule is that the total density of $\mathcal{T}$ cannot exceed 100%.

$$\rho(\mathcal{T}) = \frac{1}{\sum_{i=1}^{k} \frac{1}{avgD_i}} \leq 1 \quad (3.1)$$

Assume that the template size is $T$. In order to fulfill the rate requirement $avgD_i$ of task $\tau_i$, at least $\lceil T / avgD_i \rceil$ time slots have to be allocated to $\tau_i$ within a template of $T$ slots. The summation of the time slots allocated to all the tasks cannot exceed the template size. So the following condition has to be satisfied if there exists a feasible schedule for all the tasks in the template.

$$\sum_{i=1}^{k} \lfloor \frac{T}{avgD_i} \rfloor \leq T \quad (3.2)$$

If $T$ is equal to the least common multiple of $avgD_1, avgD_2, ..., avgD_n$, which we will simply call LCM, then $\sum_{i=1}^{k} \lfloor \frac{T}{avgD_i} \rfloor = T \times \sum_{i=1}^{k} \frac{1}{avgD_i}$. Therefore, condition (3.1) guarantees that there exists at least one value of $T$ that satisfies condition (3.2).

A fixed point scheme [5, 61] is used to iteratively find the minimum $T$ that satisfies condition (3.2). Specifically, starting from an initial estimate $T^0$, we can find the successive estimates for $T$ from the formula

$$T^{j+1} = \sum_{i=1}^{k} \lfloor \frac{T^j}{avgD_i} \rfloor \quad (3.3)$$

The initial estimate of $T^0$ starts from $k$, since it is impossible for any smaller value to satisfy condition (3.2). The right side of equation (3.3) is monotonically non-decreasing in $T$, in the sense that $T^{j+1} \geq T^j$. The recurrence is guaranteed to converge at the first point that satisfies condition (3.2) if condition (3.1) is met [5, 61].

**Example 3.1:** A task set $\mathcal{T} = \{\tau_1, ..., \tau_5\}$ needs to be scheduled on a single resource, where $\tau_1 = (4, 4), \tau_2 = (5, 6), \tau_3 = (6, 6), \tau_4 = (7, 7), \tau_5 = (10, 10)$.

The total density of $\mathcal{T}$ is 0.86, which satisfies condition (3.1). Using the fixed point scheme described above, we begin at $T^0 = 5$ and $T^1 = \sum_{i=1}^{5} \lfloor \frac{5}{avgD_i} \rfloor = 6$. In the same way, we have $T^2 = 7, T^3 = 9, T^4 = 10$ and $T^5 = 10$. The recurrence stops at this point with the template size $T = 10$. Note that for this problem, the LCM is 420.

### 3.1.3 Scheduling algorithm Tpl.Sched

Once the template size $T$ is calculated as described in the previous section, the number of instances of task $\tau_i$, is $n_i = \lceil T / avgD_i \rceil$. In this section, we will derive a template scheduling algorithm, **Tpl.Sched** which allocates $n_i$ slots to each task $\tau_i$ in a set $\mathcal{T}$ of $k$ tasks, and satisfies the
distance constraints of all the tasks. We first introduce the scheduling policy, and then explain the pseudo code of the algorithm presented in Figure 3.1.

For each task \( \tau_i \), we define \( distance_i \) as the temporary distance constraint which is currently applied for scheduling, and which is in the range of \([avgD_i, maxD_i]\). In order to keep the maximum distance as close to the average distance \( avgD_i \) as possible, we first initialize \( distance_i = avgD_i \) (line 3 in Figure 3.1), then relax it by increasing \( distance_i \) only when the algorithm cannot continue the scheduling process and only until \( distance_i \) reaches \( maxD_i \). We use \( relaxability_i = \frac{(maxD_i - distance_i)}{maxD_i} \) to indicate the capability of \( \tau_i \) to further relax \( distance_i \).

To schedule the next instance of \( \tau_i \) subject to the current distance constraint condition \( distance_i \), we calculate the schedule window of the current instance, which is relative to the allocations of the previous instances. Let \( LEFT_{i,j} \) denote the left end of the schedule window of \( \tau_{i,j} \), which is the earliest possible schedule time for \( \tau_{i,j} \). Let \( RIGHT_{i,j} \) denote the right end, indicating the latest possible schedule time for \( \tau_{i,j} \). The schedule window of \( \tau_{i,j} \) is defined as \( window_{i,j} = [LEFT_{i,j}, RIGHT_{i,j}] \). We calculate \( RIGHT_{i,j} \) and \( LEFT_{i,j} \) based on the following policies.

- \( RIGHT_{i,j} \): the latest time slot of \( \tau_{i,j} \) such that the distance between the current instance and the previously scheduled instance, \( \tau_{i,j-1} \), does not exceed the distance constraint. In other words, \( RIGHT_{i,j} \) takes care of the distance constraint concern in the forward direction.

\[
RIGHT_{i,j} = location(i, j - 1) + distance_i
\]  

(3.4)

- \( LEFT_{i,j} \): the earliest time slot such that all the still unscheduled instances are able to satisfy the distance constraint within the current template. Moreover, the distance between the last instance in the current template and the first instance in the next template is also constrained, because the allocation pattern of the template repeats continuously. In other words, \( LEFT_{i,j} \) takes care of the distance constraint concern in the backward direction.

\[
LEFT_{i,j} = T + location(i, 1) - (n_i - j + 1) \times distance_i
\]  

(3.5)

Equation (3.5) calculates \( LEFT_{i,j} \) according to the distance constraint in the backward direction. Since the template is applied cyclically, the first instance in the second template, \( \tau_{i,n_i+1} \), occupies slot \( T + location(i, 1) \). To guarantee the distance constraints, the slot \( \tau_{i,n_i+1} \) and the slot \( \tau_{i,n_i} \) must be separated by no larger than \( distance_i \). Thus, \( \tau_{i,n_i} \) must be allocated no earlier than \( T + location(i, 1) - distance_i \). For the same reason, \( LEFT_{i,n_i-1} \) must be no earlier than \( T + location(i, 1) - 2 \times distance_i \), and so on. Later in this section, we will use \( LEFT_i \) and \( RIGHT_i \) to describe the schedule window of a certain instance of task \( \tau_i \), when the index of the instance is clear from the context.
**Tpl\_Sched(T)**

1. calculate the template size $T$ using the fixed point method;
2. for $i = 1$ to $k$ do { /* initialization for all the $k$ tasks*/
3. $distance_i = avgD_i$; /* the distance constraint begins from $avgD_i$ */
4. $LEFT_{i,1} = 0$; /* assume all $\tau_i$ are ready at the beginning */
5. $RIGHT_{i,1} = distance_i$; /* set $RIGHT$ for the first instance of $\tau_i$ */
6. $num\_instance_i = n_i$; /* number of instances not yet scheduled*/
7. }
8. for $s = 1$ to $T$ do { /* schedule every slot $s$ in the template */
9. if the ready task set is not empty then {
10. choose the task $\tau_u$ with highest priority from the ready set;
11. if $s > RIGHT_u$ then /* $\tau_u$ misses its $RIGHT$ */
12. $distance_u = distance_u + (s - RIGHT_u)$; /* relax $distance_u$ */
13. }
14. else { /* no task is in the ready set*/
15. choose a task $\tau_u$ with the largest relaxability;
16. $distance_u = \frac{T + location(u,1) - s}{num\_instance_u}$; /* relax $distance_u$ */
17. }
18. if $distance_u > maxD_u$ then /* $distance_u$ is relaxed too much */
19. fail to find a schedule and reject $T$.
20. allocate current slot $s$ to task $\tau_u$;
21. $num\_instance_u = num\_instance_u - 1$;
22. calculate the schedule window for the next instance of $\tau_u$ based on (3.4) and (3.5);
23. } /* for */

![Figure 3.1: The Tpl\_Sched Algorithm](image)

We define the *ready task set* at a time slot as the set that contains all tasks with $LEFT$ smaller than or equal to the current time slot, that is, the tasks that can be scheduled at the current slot. The task with the highest priority in the ready task set is chosen for the current slot. The priorities of the tasks are determined according to the their $RIGHT$ values and their relaxabilities. Similar to EDF, we assign the highest priority to the task with the smallest $RIGHT$ value. If several tasks have the same $RIGHT$ value, ties are broken by assigning the highest priority to the task with the least relaxability. Assume that $\tau_u$ is the chosen task, this policy tries to avoid further relaxation of $distance_u$, since $distance_u$ is already close to $maxD_u$. If two tasks have the same $RIGHT$
value and relaxability, the tie is broken arbitrarily. Whenever it is necessary to relax the distance constraint of a task, the task with the largest relaxability is chosen.

In the pseudo code of _Tpl_Sched_ in Figure 3.1, after calculating the minimum template size $T$ using the method described in Section 3.1.2 (line 1), _Tpl_Sched_ initializes variables for all the $k$ tasks (from line 2 to line 7). $distance_i$ is initialized as $avgD_i$. $LEFT_{i,1}$ is assigned 0, indicating that every task is ready at the beginning of the template (line 4). $RIGHT_{i,1}$ is set to be equal to $distance_i$ (line 5), so that there is at least one slot allocated to $\tau_i$ in the first $distance_i$ time slots. The variable $num\_instance_i$ records the number of instances that still need to be allocated to $\tau_i$ from the current time slot to the end of the template. So it is initialized to $n_i$, the total number of instances of $\tau_i$ in the template (line 6). Recall that $n_i = \lceil T/avgD_i \rceil$. From line 8 to line 23, _Tpl_Sched_ repeats the allocation scheme for every time slot within the template. First, from the set of ready tasks, it chooses a task, $\tau_u$, with the highest priority (line 10), allocates the current time slot $s$ to $\tau_u$ (line 20), and decreases $num\_instance_u$ by 1, indicating that one more instance of $\tau_u$ has been scheduled (line 21). Then _Tpl_Sched_ calculates the schedule window for the next instance of $\tau_u$ (line 22) according to Equations (3.4) and (3.5).

The algorithm may fail to schedule the current slot $s$, and the temporary distance constraint of a certain task needs to be relaxed under two conditions. In the first case, the task $\tau_u$ which is chosen to be scheduled at the current slot misses its $RIGHT$ limit (line 11 - line 13). To solve the problem, $distance_u$ is increased to extend the $RIGHT$ limit (line 12). In the second situation, the ready task set may be empty because every task has a $LEFT$ later than the current time slot (line 14 - line 17). In this latter case, the task with the largest relaxability, say $\tau_u$, is chosen to relax $distance_u$ (line 15) such that the $LEFT$ value of $\tau_u$ is decreased to be exactly equal to the current time slot $s$ (line 16). The new $distance_u$ is calculated according to Equation (3.5), where $LEFT$ of the current instance of $\tau_u$ is forced to be equal to $s$, and $distance_u$ becomes the unknown variable. After increasing $^1 distance_u$, we need to check whether it was relaxed too much such that it exceeds $maxD_u$ (line 18). If true, the algorithm fails to find a schedule and rejects the set of tasks (line 19).

The following example illustrates how the algorithm works.

**Example 3.2:** Apply _Tpl_Sched_ to Example 3.1 in Section 3.1.2, where template size is calculated to be $T = 10$ for a set of 5 tasks, $\tau_1 = (4, 4), \tau_2 = (5, 6), \tau_3 = (6, 6), \tau_4 = (7, 7), \tau_5 = (10, 10)$. In Example 3.1, $\rho(T)$ is 0.86, and every task $\tau_i$, except for $\tau_2$, has a $maxD_i$ value equal to $avgD_i$. If we cast this example into a pinwheel problem, the input should be $A = \{ 4, 5, 6, 7, 10 \}$. No pinwheel algorithm is able to schedule this task set. Figure 3.2 illustrates how _Tpl_Sched_ obtains a feasible schedule making use of the specification $maxD_2 > avgD_2$. Following the scheduling scheme in _Tpl_Sched_, Figure 3.2(a) shows the scheduling of the template up to the sixth time slot.

---

1 The newly computed $distance_u$ is applied to the subsequent instances of task $\tau_u$. 
At this point, $distance_1 = \text{avg}D_1 = 4$, $distance_2 = \text{avg}D_2 = 5$, and both $\tau_1$ and $\tau_2$ need to schedule one more instance. Since $\tau_{1,2}$ is allocated at slot 3, the $RIGHT$ value of the next instance of $\tau_1$ should be $3 + distance_1 = 7$. Because the first instance of $\tau_1$ in the second template is at slot 11, the $LEFT$ of $\tau_1$ is calculated from $11 - 1 \times distance_1 = 7$. Applying the same calculation to the next instance of $\tau_2$, we find that the $LEFT$ and the $RIGHT$ of $\tau_2$ are both 7, as well. So, at slot 7, $\tau_1$ and $\tau_2$ are ready and both should be scheduled by the end of the seventh slot, which causes a conflict. Note that, at this point, $distance_1 = \max D_1$ while $distance_2 = \max D_2 - 1$, which means that $\tau_1$ has a smaller relaxability than $\tau_2$. According to the algorithm, $\tau_1$ has higher priority than $\tau_2$. Slot 7 is allocated to $\tau_1$, which makes $\tau_2$ miss its $RIGHT$ at slot 8. Then $distance_2$ is increased by 1 to be equal to $\max D_2$. The problem is solved. Figure 3.2(b) displays the final template.

Since we need to search for the task with the highest priority in the list of tasks for each time slot, the time complexity of the algorithm is $\Theta(Tlgk)$ where $k$ is the number of tasks and $T$ is the template size.

### 3.1.4 Performance results of Tpl_Sched

In this section, we present a summary of the results of applying the algorithm Tpl_Sched to a synthetic workload. We also compare our results with the results of the best pinwheel scheduling algorithm applied to the same inputs.

The simulator applies algorithm Tpl_Sched to schedule a randomly generated set of tasks $\mathcal{T} = \{\tau_i = (\text{avg}D_i, \max D_i) | 1 \leq i \leq k\}$. Each $\text{avg}D_i$ is a random number with uniform dis-
tribution in the range $[2, 50]$. The maximum distance constraint of each task, $max D_i$, is calculated according to the relative distance constraint (RDC) which is defined as $RDC_i = (max D_i - avg D_i) / avg D_i$. $RDC$ is one of the input parameters of the simulator. It measures the tightness of the distance constraint relative to the average distance. Thus, from the randomly generated $avg D_i$ and the given $RDC$, the distance constraint of $\tau_i$ is calculated as $max D_i = (RDC + 1)avg D_i$.

$RDC$ is in the range of $[0, 1.5]$ as shown at the X-axis in Figure 3.3. The number of streams in the set $T$ is generated such that the total density $\rho(T)$ is in the range $[\rho_l, \rho_h]$ where $\rho_l$ and $\rho_h$ are the lowest and highest limit, respectively, and are input parameters to the simulator.

In Figure 3.3, the performance measurement is the success rate, which is the percentage of successfully scheduled sets of tasks from an input population of 10,000 random task sets. A set of tasks is successfully scheduled if the average rate requirements and the distance constraints of all the tasks in the set are satisfied. The Y-axis represents the success rate and the X-axis represents $RDC$. We measured the performance when $RDC$ is in the range of $[0, 1.5]$ as shown in Figure 3.3. When $RDC = 0$, $max D_i$ is equal to $avg D_i$, which means that this scheduling problem is equivalent to a pinwheel problem. The four points on the vertical line of $RDC = 0$ represent the success rate of the pinwheel algorithm (the pinwheel only finds solutions for $RDC = 0$). The figure shows the improvement obtained by Tpl_Sched when $RDC$ is increased, especially when the total density is high. When $\rho \leq 0.7$, the algorithm for the pinwheel problem is guaranteed to generate a feasible schedule [19]. However, when the total density increases, the pinwheel algorithm fails to have a high success rate, while our Tpl_Sched algorithm can achieve a high success rate by allowing
the maximum distance constraint to be more relaxed than the average distance requirement. For
example, when the total density is in the range of [0.8, 0.9] and $RDC = 0.2$, that is, $max D_i$ is no
more than $1.2 avg D_i$, the success rate is as high as 80%. In summary, Figure 3.3 illustrates that, by
decoupling the rate requirement and the distance constraint specification, the $TptSched$ algorithm
increases schedulability.

In the previous simulation, we observe that when a set of tasks is successfully scheduled,
not all the tasks in the set need to relax the maximum distance constraint to the limit given by $RDC$.
The average $RDC$ of the tasks in a successfully accepted set is measured and presented in Table
3.1, on the same workload randomly generated as described previously. From the table, we can see
that even when the total density range of the task set is $[0.9, 1.0]$ the average $RDC$ is less than 12%,
which means that the algorithm keeps the maximum distance between two consecutive instances
very close to the average distance for most of the tasks in the set.

<table>
<thead>
<tr>
<th>density</th>
<th>[0, 0.7]</th>
<th>[0.7, 0.8]</th>
<th>[0.8, 0.9]</th>
<th>[0.9, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg $RDC$</td>
<td>0.048%</td>
<td>0.624%</td>
<td>2.16%</td>
<td>11.6%</td>
</tr>
</tbody>
</table>

Table 3.1: Average $RDC$ v.s. Total Density

### 3.2 Scheduling static tasks on multiple resources

A general purpose real-time system may consist of several sets of resources, where each
set includes a pool of identical resources. Each resource has to be exclusively accessed, that is,
at any time, only one task can access a resource. In such a general purpose system, a real-time
task specifies the resources that need to be accessed during its execution, in addition to its timing
characteristics. Thus, the objective is to schedule the tasks such that at any time, each resource
is accessed by at most one task, and the timing constraints of the tasks are met. For example, a
computer system may have three sets of resources. The first set consists of a pool of four processors,
the second set consists of a shared memory object and the third set consists of an I/O device. A task
requires a processor from the first set and the shared memory object from the second set for its
execution purpose. Another task also requires a processor from the first set, and the I/O device
from the third set. These two tasks can be scheduled in parallel, because there are more than two
processors in the first set, and the two tasks do not share any other resource. However, a task that
requires a processor, the shared memory object and the I/O device cannot be scheduled in parallel
with either of the two tasks described previously, even though there are more than three processors
in the processor pool.
Some specific resource models have been extensively studied. For example, processing resources and I/O resources are allocated in parallel to real-time message streams based on the QoS specifications, where the problem of scheduling a single resource to satisfy multiple QoS dimensions is studied in [88], and the problem of apportioning multiple resources to satisfy a single QoS is studied in [89]. To schedule tasks on a single processor and an exclusively accessed resource, the priority ceiling protocol [97, 90] minimizes priority inversion. In [91, 86], priority synchronization protocols are developed to schedule real-time tasks in a shared memory multiprocessor system, where the memory object and a pool of identical processors are the resources to be shared by the tasks. Moreover, an architecture of resource kernel [87] is defined and implemented in Linux operating system [80, 81], which provides interface for tasks to specify resource requirement, and manages the reservation of multiple resources to meet the timeliness of the tasks.

In this section, we will introduce the problem of scheduling message streams on a crossbar switch and a WDMA star coupler which are two special instances of the multiple resource scheduling problem in the communication environment. Resource constraints are added in the task model, in addition to the distance constraints and average rate requirements. The slot allocation algorithm introduced in the previous section for single resource is extended to solve the scheduling problem for crossbar switch and WDMA star coupler.

3.2.1 Scheduling message streams in a crossbar switch

3.2.1.1 Problem definition

In a point-to-point network, switch scheduling is very important. Time Division Multiplex Access (TDMA) switching systems are widely used in terrestrial and satellite communication because of the need to concentrate traffic from many low bandwidth sources into high speed channels [17]. These systems typically consist of a number of input links connected to a number of output links by means of a crossbar switch. During each time slot in a TDMA template, the switch can receive one unit of traffic from each input link and transmit it to one distinct output link. A valid configuration of a switch should satisfy the following two rules:

1. **No input conflict**: At any time, no two message streams can be originated from the same input link.

2. **No output conflict**: At any time, no two message streams can be destined to the same output link.

Consider an \( a \times b \) switch which has \( a \) input links, \( iLink_1, \ldots, iLink_a \), and \( b \) output links \( oLink_1, \ldots, oLink_b \) (usually \( a = b \)). The crossbar switch can be considered as equivalent to a system
consisting of 2 sets of resources, the set of input links \( \{iLink_1, \ldots, iLink_a\} \) and the set of output links \( \{oLink_1, \ldots, oLink_b\} \). Each message stream indicates the input link and output link from the two sets, respectively. The resource must be exclusively allocated at each time slot, to prevent the input conflict and output conflict.

The task model needs to be extended in this switching system to add the requested resources for each task. In addition to the rate requirement and distance constraint, each message stream specifies an input link and an output link for transmission, based on the source and destination of the message stream. Assume that a set \( T \) of \( k \) isochronous message streams \( \tau_1, \ldots, \tau_k \) is to be scheduled on a switch. Each stream is represented as \( \tau_i = (I_i, O_i, avgD_i, maxD_i) \), where \( I_i \in [1, a] \) and \( O_i \in [1, b] \), indicating the input link number and the output link number for stream \( \tau_i \). The meanings of \( avgD_i \) and \( maxD_i \) are the same as defined earlier.

To illustrate how TDMA template controls the configuration of a switch, Figure 3.4 shows a simple example of TDMA switch with 4 input links and 4 output links. The template size in the example is 6 slots. At each time slot, 4 packets are simultaneously transmitted from the 4 input links to 4 output links, as long as there is no input conflict or output conflict. The switch configuration is controlled by the template, which is represented by 4 rows of schedule information shown at the input side. At the \( i^{th} \) input link, the number in each time slot indicates the number of the output link that will be connected to the \( i^{th} \) input link in that slot. For example, the column of 4 numbers in time slot 1 decides that at this time, input link 1 should be connected with output link 1, input link 2 with output link 3, input link 3 with output link 4, and input link 4 with output link 2. This switch
configuration for time slot 1 is shown with the dotted lines. Obviously, this is a valid configuration, because the 4 connections originate from 4 unique input links, and destined to 4 unique output links. This example shows that a valid configuration of the switch is represented by a permutation of the output channel numbers, which is (1, 3, 4, 2) for time slot 1.

The scheduling algorithm of the switch allocates each time slot within a template to a group of message streams which is allowed by a valid switch configuration. Without loss of generality, we assume $a \leq b$. We consider the $a$ input links as $a$ scheduling channels, thus the message streams in the set $IN_x$ can only be scheduled on the $x^{th}$ channel. At every time slot, if we select one message stream from each input set $IN_x$, then at most $a$ message streams can be scheduled at a time slot, and clearly, there is no input conflict in these message streams. Thus, the objective of the algorithm is to select one message stream from each input set $IN_x$ such that there is no output conflict and the maximum distance constraints are satisfied. The allocation information of each time slot records not only the group of selected streams, but also the switch configuration that specifies which input link will be switched to a certain output link.

We define $IN_x$ to be the set of streams originating at input link $x$, that is, $IN_x = \{ \tau_i | I_i = x \}$, and we define $OUT_y$ to be the set of streams that are destined to output link $y$, that is, $OUT_y = \{ \tau_i | O_i = y \}$. Note that $T = \bigcup_{x=1}^{a} IN_x = \bigcup_{y=1}^{b} OUT_y$.

Note that two necessary conditions of schedulability need to be taken into consideration. Specifically, it is impossible to get a feasible schedule for the switch if the total density of the set of streams originating at any input link exceeds one. The same condition should also hold for the set of message streams sent to the same output link. These two conditions are expressed as follows.

\[ \text{for } x = 1, ..., a \quad \rho(IN_x) = \sum_{\tau_i \in IN_x} \frac{1}{avgD_i} \leq 1 \quad (3.6) \]

\[ \text{for } y = 1, ..., b \quad \rho(OUT_y) = \sum_{\tau_i \in OUT_y} \frac{1}{avgD_i} \leq 1 \quad (3.7) \]

We define a set of streams $T$ as a valid set if it satisfies conditions (3.6) and (3.7).

In the following sections, we first calculate the smallest TDMA template size such that the rate requirement of every message stream is satisfied. Then we describe a time slot allocation scheme to schedule the message streams on each input channel for every time slot in a template.

### 3.2.1.2 Calculating the smallest template size

Assume that the template size is $T$. Since the number of instances of stream $\tau_i$ in the template is equal to $\lceil \frac{T}{avgD_i} \rceil$, conditions (3.6) and (3.7) are equivalent to the following two conditions,
which require that on all the input links and output links, the summation of the number of instances cannot be larger than the template size.

\[ \sum_{\tau_i \in IN_x} \left\lfloor \frac{T}{\text{avg}D_i} \right\rfloor \leq T \quad (3.8) \]

\[ \sum_{\tau_i \in OUT_y} \left\lfloor \frac{T}{\text{avg}D_i} \right\rfloor \leq T \quad (3.9) \]

In Section 3.1.2, the fixed point scheme was applied to calculate the smallest template size for a single resource. In order to find a \( T \) that simultaneously satisfies the multiple inequalities declared in conditions (3.8) and (3.9), we extend the fixed point scheme to iteratively find the minimum \( T \). Specifically, starting from an initial estimate \( T^0 \), we find successive estimates for \( T \) from the following equations.

\[ T^{j+1} = \text{MAX}_{z \in [1, a+b]} \left\{ \sum_{\tau_i \in T_z} \left\lfloor \frac{T^j}{\text{avg}D_i} \right\rfloor \right\} \quad (3.10) \]

In Equation (3.10), the operation \( \text{MAX} \) returns the maximum value among its arguments. The set \( T_z \) \((z = 1, \ldots, a+b)\) is a set of message streams that either originate from a specific input link or are destined to a specific output link. That is, \( T_z \in \{IN_1, \ldots, IN_a\} \cup \{OUT_1, \ldots, OUT_b\} \).

Starting with the current estimate \( T^j \), the method calculates the right side of equation (3.10) on each \( T_z \) of the \( a+b \) sets. The maximum value among the \( a+b \) results is used as the next estimate, \( T^{j+1} \). The recurrence continues until \( T^{j+1} = T^j \).

Figure 3.5 helps to visually interpret the fixed point scheme used for solving these inequalities. If we define \( F_z(T) = \sum_{\tau_i \in T_z} \left\lfloor \frac{T}{\text{avg}D_i} \right\rfloor \), then the \( X \)-axis in the figure represents the value of \( T \) and the \( Y \)-axis is \( F_z(T) \); there is one curve for each different value of \( z \). Each function \( F_z(T) \) \((z = 1, 2, 3)\) is a monotonically non-decreasing step function because of the ceiling operation. The straight \( 45^\circ \) line, which represents the function \( Y = X \) is helpful to see how the next estimate, \( T^{j+1} \), is obtained from \( T^j \) by projecting the \( Y \) value to the \( X \)-axis. For example, assume that the current estimate is \( T^j \). The next estimate, \( T^{j+1} \), should be equal to the \( Y \) coordinate value of point A, which is the highest point on the three curves whose \( X \) coordinate is \( T^j \). By projecting point A onto the \( X \)-axis through the \( 45^\circ \) line, we obtain the next estimate \( T^{j+1} \) and continue the recurrence.

With the multiple-equation fixed-point scheme, the goal is to find the minimum \( T \) at which point all the curves are on or below the \( Y = X \) line, i.e., all the inequalities in (3.8) and (3.9) are satisfied.

**Proposition:** The fixed point scheme is guaranteed to converge at the minimum \( T \) that satisfies condition (3.8) and (3.9) as long as the input set is a valid set, that is, the conditions in (3.6) and (3.7) are true.
To prove the proposition, note that, due to conditions (3.6) and (3.7), conditions (3.8) and (3.9) are always satisfied when \( T = LCM\{avgD_1, ..., avgD_n\} \). So there exist at least one valid \( T \). Assume that the minimum \( T \) which satisfies conditions (3.8) and (3.9) is \( T_{min} \), we need to show that the fixed point recurrence will find this value. By the definition of \( T_{min} \), \( F_z(T_{min}) \leq T_{min} \), for \( z = 1, ..., a + b \) (all the curves in Figure 3.5 should be below the 45° line at the point \( X = T_{min} \)). Without loss of generality, assume that \( F_1(T_{min}) \geq F_1(T_{min}) \forall z \neq 1 \) (curve \( C_1 \) in Figure 3.5), and let \( L \) be the largest value smaller than \( T_{min} \) that satisfies \( F_1(L) < F_1(T_{min}) \) (see Figure 3.5). Now, consider an iterate \( T^j < T_{min} \). If \( L < T^j \leq T_{min} \), then clearly \( T^{j+1} = T_{min} \). If, however, \( T^j \leq L \), then we can prove that \( T^{j+1} \leq T_{min} \). Assume \( T^{j+1} > T_{min} \), which means that, for some \( z \neq 1 \), \( T^{j+1} = F_z(T^j) \). Because \( T^j \leq T_{min} \), and we know that the function \( F_z \) is non-decreasing, then \( F_z(T^j) \leq F_z(T_{min}) \). Note that the relation \( T^{j+1} > T_{min} \) is equivalent to \( F_z(T^j) > F_1(T_{min}) \), and thus, implies that \( F_1(T_{min}) < F_z(T_{min}) \). This contradicts the assumption that \( F_1(T_{min}) \geq F_z(T_{min}) \). Hence, either \( T^{j+1} \leq L \), in which case we can repeat the above argument, or \( L < T^{j+1} \leq T_{min} \), which leads to \( T^{j+2} = T_{min} \). This proves that it is impossible for the fixed point recurrence in (3.10) to skip \( T_{min} \).
3.2.1.3 Scheduling scheme for the TDMA switch

After calculating the minimum template size that satisfies conditions (3.8) and (3.9), the scheduler allocates each time slot in the template to a group of message streams allowed by a valid switch configuration. Since every message stream \( \tau_i \) belongs to only one set \( IN_x \), there is no input conflict if we schedule one stream from each set streams \( IN_x \) on the \( x^{th} \) input link. However, the output conflict has to be taken into consideration when scheduling all the output links. In other words, we cannot use \textbf{Tpl.Sched} to schedule each \( IN_x \) independently. Instead, the algorithm should select a message stream from each set \( IN_x \) to schedule in a slot in such a way that no output conflict results.

The scheduling algorithm for a crossbar switch is presented in Figure 3.6. In order to solve the output conflict efficiently, we define the task priority in the same way as in \textbf{Tpl.Sched} for a single resource, that is, the task with the smallest \( \text{RIGHT} \) value and the least relaxability has the highest priority. On an input link \( u \), we first choose a stream \( \tau_i \) from \( IN_u \) which has the highest priority. It is possible, however, that the chosen stream, \( \tau_i \), has an output conflict with another stream, \( \tau_j \), which is chosen to be scheduled on the input link \( v \) in the same time slot. In order to solve this conflict, we compare the priorities of \( \tau_i \) and \( \tau_j \). Suppose that \( \tau_i \) has a priority higher than \( \tau_j \), then we keep \( \tau_i \) at the current slot and delay \( \tau_j \) on the input link \( v \) by picking another stream from \( IN_v \) that has the highest priority in the subset \( OUT_v \) – \{ \tau_j \}. If some stream misses its \( \text{RIGHT} \) limit, the same relaxation and negotiation scheme is applied as in \textbf{Tpl.Sched}. This scheduling scheme is repeated until all the slots in the template are scheduled successfully.

3.2.1.4 Performance results

Figure 3.7 shows the performance of the algorithm explained in the previous section applied to the scheduling of an \( 8 \times 8 \) crossbar switch. The density range specified for each curve in Figure 3.7 is a parameter to the simulation program, indicating the total density of the message set scheduled on each input link. The set of random streams are generated to comply to the density specification. For each stream \( \tau_i \) in \( T \), the \( \text{avg}D_i \) is randomly generated in the same way as the one described in Section 3.1.4, and the parameter \( \text{max}D_i \) is also calculated in the same way according to a given \( \text{RDC} \) (Relative Distance Constraint) value. The input link, \( I_i \), and the output link, \( O_i \), of a message stream \( \tau_i \) are randomly chosen in the range \([1, 8]\) such that \( T \) is a valid set and the density of the set of message streams on each output link is within the specified range. Again, the \( Y \)-axis represents the success rate and the \( X \)-axis represents \( \text{RDC} \). A set of message streams is successfully accepted if a feasible schedule template is found such that the rate requirements of the message streams are met within the flexible maximum distance constraint specifications, and there is no input conflict or output conflict in any time slot.
1. For each message stream, initialize the variables in the same way as shown from Line 2 to Line 7 in Figure 3.1.

2. For each time slot in the template, execute the following steps to select one message stream on each input channel.

   (a) On each input channel \( u \), choose a stream with the highest priority from the ready set in the same way as shown from Line 9 to Line 10 in Figure 3.1. Thus, there are \( a \) (in an \( a \times b \) switch) candidate streams selected.

   (b) If the candidate streams do not have output conflict, then return them as the set of message streams that are scheduled in the current time slot.

   (c) Otherwise, at least two message streams have output conflict among the \( a \) streams selected in Step 2(a). Assume that stream \( \tau_i \) from input \( IN_u \) and \( \tau_j \) from input \( IN_v \) have output conflict, and the priority of \( \tau_i \) is higher than the priority of \( \tau_j \). Then \( \tau_j \) is removed from the set of candidate streams for the current time slot. The next stream with the highest priority is selected from \( IN_v \). Repeat this step until there is no output conflict in the candidate set of streams.

   (d) When necessary, the temporary distance constraints of the message streams are relaxed following the same policy as described from Line 11 to Line 17 in Figure 3.1.

Figure 3.6: The algorithm to schedule the input channels of a switch

From Figure 3.7, we can see that the success rate is low when the maximum distance constraint is equal to the average distance and the density is high for each output link. This is due to the fact that high density of each input link also leads to high possibility of output conflict. Moreover, when the maximum distance constraint specification is tight compared to the rate requirement, the high conflict probability will result in a low success ratio. However, by relaxing the maximum distance constraint, the success rate is greatly improved even when the density is high. For example, when the density of each link is in the range of [0.75, 1.0], the success rate is less than 10% when the RDC is equal to 0, because the tight maximum distance constraint specification prevents the algorithm from solving the output conflicts effectively. If we relax the value of RDC to be 0.5, the success rate increases sharply to more than 50%.
3.2.2 Scheduling message streams in a WDMA star coupler

In order to further evaluate the performance of our algorithm, we adapt the algorithm presented for a crossbar switch to scheduling a passive star coupler in WDMA (Wavelength Division Multiplexing) optical networks and compare its results with those of an algorithm given in [103] for solving the same problem.

Wavelength Division Multiple Access is a promising approach to efficiently utilize the large bandwidth of optical networks [82]. In a WDMA-based network, an optical wavelength represents a transmission channel, and multiple channels can be multiplexed onto a single fiber. Stations may transmit/receive packets on different channels using a tunable laser transmitter/receiver.

A widely used optical network topology is the passive star topology which uses a broadcast star coupler to transmit message streams. In this configuration, every source station uses a tunable transmitter to send message streams on a specific wavelength. The star coupler combines the message streams from various source stations and broadcasts the mixed optical information to all the destination stations. In order to receive a packet from a certain source, a destination station needs a tunable receiver to pick up its message streams on the expected wavelength from the wavelength division multiplexed broadcast stream.

In some sense, scheduling the passive star coupler is similar to scheduling the crossbar switch. In the passive star coupler, one source is connected to one destination by a specific wavelength channel, in a way similar to the crossbar switch configuration where an input link is connected to an output link by the switch. Similar to the crossbar switch, the passive star coupler configuration
also requires the two validation rules concerning both input conflict and output conflict. That is, no multiple source stations can be connected to the same destination station at the same time, and vice versa. However, the number of wavelengths supported by the system implies the maximum number of message streams that can be transmitted at any time slot. Therefore, a third validation rule specifies that there is no wavelength conflict, i.e., a specific wavelength can only be used to transmit one message stream at any time slot. Assume that an \( a \times b \) WDMA star coupler supports \( w \) wavelengths, and has \( a \) source stations \( \text{source}_1, ..., \text{source}_a \), and \( b \) destination stations \( \text{dest}_1, ..., \text{dest}_b \). The star coupler is equivalent to a system consisting of 3 sets of resources, namely, the pool of \( w \) wavelengths \( \{WL_1, ..., WL_w\} \), the set of source stations \( \{\text{source}_1, ..., \text{source}_a\} \), and the set of destination stations \( \{\text{dest}_1, ..., \text{dest}_b\} \). Again, the resources must be exclusively allocated at each time slot because of the concerns of the input conflict, output conflict, and wavelength conflict.

Each message stream needs to specify a source station and a destination station, in addition to its timing constraints. The characteristics of a message stream is the same as defined in Section 3.2.1. That is, \( \tau_i = (I_i, O_i, \text{avg}D_i, \text{max}D_i) \), where \( I_i \in [1, a] \) and \( O_i \in [1, b] \), indicating the source station ID number and the destination station ID number for stream \( \tau_i \).

The WDMA star coupler scheduling problem is to schedule at most \( w \) distinct message streams at every time slot in a template such that there is no conflict, and the timing constraints of the message streams are satisfied.

In an \( a \times b \) crossbar switch \( (a \leq b) \), \( a \) channels can be utilized to transmit message streams, with each channel corresponding to an input link. This implies that at most \( a \) message streams can be scheduled in each time slot. In an \( a \times b \) star coupler network with \( a \) source stations and \( b \) destination stations, the number of channels is equal to the number of tunable wavelengths, \( w \), which is usually smaller than \( a \) and \( b \). This implies that only \( w \) message streams can be scheduled in each slot. Specifically, if \( w \geq a \), then each wavelength can be used to schedule one of the sets \( \text{IN}_x \), \( x = 1, ..., a \). Thus, the algorithm for the crossbar switch can be applied. If \( w < a \), however, the schedule select one stream in the set \( T = \bigcup_{x=1}^{a} \text{IN}_x \) to be assigned to each of the \( w \) wavelengths, in which case the wavelength conflict can be avoided, but both output and input conflicts must be solved by the scheduling algorithm.

We adapted the algorithm for the crossbar switch to the star coupler situation by following the same definition of task priority and the same policies of relaxing and the temporary distance constraints. The algorithm of crossbar switch starts by selecting \( a \) streams, each of which has the highest priority in every set \( \text{IN}_x \), \( x = 1, ..., a \). However, only the the first \( w \) message streams with the highest priorities among those \( a \) message streams are chosen for the current time slot. Input conflict and output conflict are solved in the same way as shown in Figure 3.6, which is based on the comparison of the priorities among the conflicting streams.
A heuristic algorithm named *Binary Splitting* was presented in [103] for scheduling passive star couplers. *Binary Splitting* is a time slot allocation scheme for time-constraint communications based on the specialization result of the pinwheel problem. The algorithm, however, only applies to sets of streams whose specialization is successful and whose density satisfies certain validation conditions [103]. Furthermore, a valid set of streams may still be rejected by *Binary Splitting* after passing the validation conditions in [103].

![Figure 3.8: Performance of 8 x 8 WDMA passive star coupler](attachment:image)

Figure 3.8 presents the performance of our algorithm on an 8 x 8 star coupler configuration. The four curves correspond to the success rate of four groups of random inputs, each generated based on the specified density range for the message stream set on every source station. Since the number of channels cannot be smaller than the total density, we choose the value of $w$ according to the possible highest density of each station. For example, if the density of each set of message streams that originates from each source station is in the range of $[0.25, 0.5]$, then with 8 destination stations, the maximum total density can be as large as $8 \times 0.5 = 4$. So we set $w$ to 4 for this group of inputs. The four points on the vertical line of $RDC = 0$ represent the success rate of the validation test of the *Binary Splitting* algorithm on the same input sets. Note that although the performance of the validation test of *Binary Splitting* are better than the performance of our algorithm when $RDC = 0$, it is not necessarily true that *Binary Splitting* performs better. The results shown in Figure 3.8 are the optimistic estimates of the performance of *Binary Splitting*. In reality, it is possible that *Binary Splitting* will not be able to successfully schedule all the sets that pass the validation test given the restrictions imposed on the number of wavelengths. In other words, the success rate
of *Binary Splitting* cannot be higher than the four points shown in Figure 3.8. When the density is high, most input sets are rejected even before applying *Binary Splitting* because they violate the validation conditions needed to apply the algorithm. Again, Figure 3.8 illustrates that the policy of relaxing maximum distance constraint improves the success rate, especially when the network workload is high.

### 3.3 Adding minimum distance constraint in the task model

#### 3.3.1 New task model with minimum distance constraint

As introduced in Chapter 2, a general distance constraint includes both a maximum and a minimum distance constraint. The minimum distance constraint specifies that two consecutive instances of a certain task cannot be too close to each other. The separation between two adjacent instances must be larger than the minimum distance constraint. The research work presented in the previous sections in this chapter is based on the basic task model where only maximum distance constraint is considered. In this section, we will extend the task model by adding the minimum distance constraint, and describe how to modify the algorithm Tpl.Sched presented in Section 3.1 to take the minimum distance constraint into consideration.

A task defined by the extended task model is represented as \( \tau_i = (\text{avg}D_i, \min D_i, \max D_i) \) where \( \min D_i \) denotes the minimum distance constraint, and \( \min D_i \leq \text{avg}D_i \leq \max D_i \). Note that when the template-based scheduling paradigm is adopted, the minimum distance constraint is related to the maximum distance constraint, because of the fact that the template is applied repeatedly, and the average rate requirement must be satisfied. For example, if a task \( \tau_i \) requests \( \max D_i \) to be equal to \( \text{avg}D_i \), then \( \min D_i \) must also be equal to \( \text{avg}D_i \). A request of \( \min D_i \) smaller than \( \text{avg}D_i \) is not accurate, since the distance between any adjacent instances has to be equal to \( \text{avg}D_i \) in order to satisfy the \( \text{avg}D_i \) and \( \max D_i \) requirements. On the other hand, if \( \max D_i \) is much larger than \( \text{avg}D_i \), then \( \min D_i \) should also be much smaller than \( \text{avg}D_i \). Otherwise, \( \max D_i \) and \( \min D_i \) can never be satisfied together with \( \text{avg}D_i \), which can be considered as an invalid requirement. In general, \( \min D_i \) and \( \max D_i \) have to be both tight or both loose relative to the average distance requirement.

In order to simplify the relationship between the three parameters, we assume that \( \min D_i \) and \( \max D_i \) are both determined by a *relative distance constraint* factor \( RDC_i \). Specifically, \( \max D_i = \text{avg}D_i(1 + RDC_i) \) and \( \min D_i = \text{avg}D_i(1 - RDC_i) \), where \( RDC_i \geq 0 \). If \( RDC_i \geq 1 \), then \( \min D_i = 1 \). Recall that \( RDC \) is used to generate task parameters in the simulations. This is a measurement that reflects the tightness of the distance constraints relative to the average distance requirement. Therefore, another equivalent form of the task model is \( \tau_i = (\text{avg}D_i, RDC_i) \).
3.3.2 Modification of Tpl_Sched to consider the minimum distance constraints

Given a set $k$ of static tasks with $\tau_i = (avg D_i, RDC_i)$, the problem is to generate a scheduling template that satisfies the $avg D_i$, $\min D_i$ and $\max D_i$ requirement of each task $\tau_i$.

As described in Section 3.1.2, the objective of calculating the minimum template size is to satisfy the average rate requirement, which is represented by the parameter $avg D_i$ of each task $\tau_i$. Distance constraints are not considered in this problem. Therefore, adding the minimum distance constraint $\min D_i$ does not influence the result. The original fixed point scheme presented in Section 3.1.2 can be used to find the smallest template size.

Once the template size $T$ is calculated, the number of instances of task $\tau_i$ is $n_i = \lceil T / avg D_i \rceil$. Similar to the usage of the temporary distance constraint $dist_i$ in the original algorithm Tpl_Sched presented in Section 3.1.3, a variable $\alpha_i$ is used as the currently applied relative distance constraint for each task $\tau_i$, which is in the range of $[0, RDC_i]$. In order to keep the distances between adjacent instances as close to the average distance $avg D_i$ as possible, we first initialize $\alpha_i = 0$, then relax $\alpha_i$ by increasing it only when the algorithm cannot continue the scheduling process and only until $\alpha_i$ reaches $RDC_i$. Based on the temporary relative distance constraint $\alpha_i$, we derive the temporary maximum distance constraint $\max D'_i$ and the temporary minimum distance constraints $\min D'_i$. That is, $\max D'_i = avg D_i(1 + \alpha_i)$. $\min D'_i = avg D_i(1 - \alpha_i)$ if $\alpha_i < 1$, otherwise, $\min D'_i = 1$. Whenever $\alpha_i$ is relaxed, $\max D'_i$ and $\min D'_i$ are also relaxed accordingly, until they reach the limit $\max D_i$ and $\min D_i$ which are set by $RDC_i$.

Same as the algorithm Tpl_Sched, for each instance $\tau_{i,j}$, a schedule window, $\text{window}_{i,j} = [LEFT_{i,j}, RIGHT_{i,j}]$, is calculated based on the allocations of the previously scheduled instances and distance constraints. In Equation (3.4) and (3.5), the calculation of $LEFT_{i,j}$ and $RIGHT_{i,j}$ only considers the maximum distance constraint. The following formula define the left and right boundaries of a valid schedule window for $\tau_{i,j}$, considering both the minimum and maximum distance constraints.

- $LEFT_{i,j}$ takes care of the minimum distance constraint concern in the forward direction, and the maximum distance constraint concern in the backward direction.
  
  $$LEFT_{i,j} = \text{MAX}(location(i, j - 1) + \min D'_i, T + location(i, 1) - (n_i - j + 1) \times \max D'_i)$$  

- $RIGHT_{i,j}$ takes care of the maximum distance constraint concern in the forward direction, and the minimum distance constraint concern in the backward direction.
  
  $$RIGHT_{i,j} = \text{MIN}(location(i, j - 1) + \max D'_i, T + location(i, 1) - (n_i - j + 1) \times \min D'_i)$$
In Equation (3.11), the function $\text{MAX()}$ returns the maximum value among its arguments. In Equation (3.12), the function $\text{MIN()}$ returns the minimum value among its arguments. The consideration of the distance constraints in the forward direction makes sure that the distance between the current instance and the next instance is within the range $[\text{min}D'_i, \text{max}D'_i]$. The consideration of the distance constraints in the backward direction makes sure that the inter-template distance constraints can be satisfied, and it is possible for the remaining unscheduled instances to meet the distance constraints.

The definition of $\text{relaxability}_i$ is now changed to $\text{relaxability} = RDC_i - \alpha_i$ to indicate the capability of $\tau_i$ to further relax $\alpha_i$. The meaning of the ready task set at a time slot is the same as the original definition, which is a set that contains all tasks with $\text{LEFT}$ smaller than or equal to the current time slot. We apply the same scheduling policy to allocate the time slots in the template. At each slot, the task with the highest priority in the ready task set is chosen. The priorities of the tasks are determined according to their $\text{RIGHT}$ values and their relaxabilities. Moreover, whenever it is necessary to relax the relative distance constraint of a task, the task with the largest relaxability is chosen.

The modified $\text{Tpl.Sched}$ algorithm is presented in Figure 3.9. Similar to the original algorithm in Figure 3.1, it calculates the smallest template size, initializes the variables for all tasks, and allocates each time slot in the template to a specific task instance according to the scheduling policy explained earlier. The difference lies in the fact that the original algorithm only needs one variable for the temporary maximum distance constraint, $\text{distance}_i$, whereas the modified version has three temporary variables $\alpha_i$, $\text{min}D'_i$ and $\text{max}D'_i$. For example, at the initialization stage, the original algorithm initializes $\text{distance}_i$ be equal to $\text{avg}D_i$ (line 3 in Figure 3.1). Following the same policy, the modified algorithm initializes $\alpha_i$ to be 0, and both $\text{min}D'_i$ and $\text{max}D'_i$ to be equal to $\text{avg}D_i$ (line 3.a, line 3.b and line 3.c in Figure 3.9).

Recall that the algorithm may fail to schedule the current slot $s$ and the distance constraint of a certain task needs to be relaxed under two conditions. In the first case, the task $\tau_u$ which is chosen to be scheduled at the current slot misses its $\text{RIGHT}$ limit. The original algorithm simply relax $\text{distance}_u$ to extend $\text{RIGHT}_u$ (line 12 in Figure 3.1). However, Equation (3.12) shows that $\text{RIGHT}_u$ is determined by both the minimum distance constraint and the maximum distance constraint. In order to extend the limit $\text{RIGHT}_u$ to current time $s$, we calculate the relaxed value of $\text{max}D'_u$ and $\text{min}D'_u$ according to the first and the second argument of the $\text{MAX()}$ function in Equation (3.12), respectively (line 12.a and line 12.b in Figure 3.9). Then a value is derived for $\alpha_u$ based on the definition of relative distance constraint, such that it is large enough to relax both $\text{min}D'_u$ and $\text{max}D'_u$ (line 12.c in Figure 3.9). In the second situation where the algorithm fails to proceed, the ready task set may be empty because every task has a $\text{LEFT}$ later than the current
time slot $s$. In order to extend $LEFT_u$, we relax $\text{min} D'_u$, $\text{max} D'_u$ and $\alpha_u$ in a similar way (line 16.a, line 16.b and line 16.c in Figure 3.9).

### 3.3.3 Performance of the modified algorithm

We apply the modified $\text{Tpl.Sched}$ algorithm to schedule the same 10,000 sets of tasks that are randomly generated for the simulator described in Section 3.1.4. The parameter $\text{avg} D_i$ of each task $\tau_i$ is a random number within a certain range. The number of tasks, $k$, is generated such that the total workload is in a certain range. The simulator supplies a system parameter $RDC$ as the relative distance constraint for all the tasks. Originally, the simulation program only considered $\text{max} D_i$ which is calculated based on $\text{avg} D_i$ and $RDC$. In this section, we apply the modified scheduling algorithm with the additional constraint of $\text{min} D_i$ which is also calculated from $\text{avg} D_i$ and $RDC$.

The original simulation performance is shown in Figure 3.3, where the success rate is measured and plotted with $RDC$ in the range of $[0, 1.5]$. Applying the same measurement, the comparison of the original algorithm and the modified algorithm is shown in Figure 3.10(a), 3.10(b), 3.10(c), and 3.10(d), where the total density of each task set is in the range of $[0, 0.7]$, $[0.7, 0.8]$, $[0.8, 0.9]$, $[0.9, 1.0]$, respectively. In each figure, the curve on the top is the result of applying the original algorithm to the basic task model. The curve at the bottom is the result of applying the modified algorithm to the extended task model. Since there is no minimum distance constraint in the basic task model, it is equivalent to having a $\text{min} D_i = 1$ for each task. The extended task model has a more strict scheduling requirement, because of the explicit specification of a $\text{min} D_i$ that can be larger than 1. Therefore, the performance of the original algorithm is better than the modified algorithm. However, when $RDC = 0$ and when $RDC \geq 1$, both task models requests the same distance constraints according to the definition of relative distance constraint. That is why the performances of the two algorithms are the same at $RDC = 0$ and $RDC \geq 1$. The difference between the performances of the two algorithms is smaller under a low system workload than under a high system workload, because the effect of the $\text{min} D_i$ constraint is more significant in the latter case.

### 3.4 Summary

This chapter derived scheduling algorithms for systems that consist of static tasks. The first problem is to schedule a set of static tasks with average rate requirements and maximum distance constraints over a single resource. In order to decrease scheduling overhead, the minimum template size is calculated via a fixed point scheme such that the rate requirements of all the static
tasks can be satisfied. Once the template size is determined, algorithm $\text{Tpl\_Sched}$ allocates each time slot in the template to a ready task with the highest priority.

$\text{Tpl\_Sched}$ is extended to solve the problem of scheduling static tasks on multiple channels of a crossbar switch and a WDMA star coupler. Resource conflicts, such as output conflicts and wavelength conflicts, are taken into consideration in addition to rate requirements and distance constraints. The minimum template size is calculated according to a new technique which extends the original fixed point scheme to satisfy rate requirements over multiple channels. $\text{Tpl\_Sched}$ is also extended to schedule static tasks with both minimum distance constraints and maximum distance constraints.

The performance of my algorithms is compared with the performance pinwheel related algorithms which tie the maximum distance constraints to the average transmission rate of the tasks. The simulation results show that higher schedulability is achieved when the rate requirements and the distance constraints are decoupled, especially when the load is high. It is also shown that my algorithms are particularly useful when the distance constraints are relaxed.
1. calculate the template size $T$ using the fixed point method;

2. for $i = 1$ to $k$ do { /* initialization for all the $k$ tasks*/
   3.a $\alpha_i = 0$; /* initially, $\alpha_i$ begins from 0 */
   3.b $max D'_i = avg D_i$; /* initially, $max D'_i$ is set to be $avg D_i$ */
   3.c $min D'_i = avg D_i$; /* initially, $min D'_i$ is set to be $avg D_i$ */

4. $LEFT_{i,1} = 0$; /* assume all $\tau_i$ are ready at the beginning */

5. $RIGHT_{i,1} = max D'_i$; /* set $RIGHT$ for the first instance of $\tau_i$ */

6. $num\_instance_i = n_i$; /* number of instances not yet scheduled */

7. }

8. for $s = 1$ to $T$ do { /* schedule every slot $s$ in the template */
   9. if the ready task set is not empty then {
      10. choose the task $\tau_u$ with highest priority from the ready set;
      11. if $s > RIGHT_{u}$ then /* $\tau_u$ misses its $RIGHT$ */
         12.a $max D'_u = s - location(u, n_u - num\_instance_u)$;
            /* relax $max D'_u$ to extend $RIGHT_{u}$ */
         12.b $min D'_u = \frac{T + location(u,1) - s}{num\_instance_u}$; /* relax $min D'_u$ to extend $RIGHT_{u}$ */
         12.c $\alpha_u = \text{MAX}\left(\frac{max D'_u - avg D_u}{avg D_u}, \frac{avg D_u - min D'_u}{avg D_u}\right)$; /* find a large enough $\alpha_u$ */
      13. }
   14. else { /* no task is in the ready set */
      15. choose a task $\tau_u$ with the largest relaxability;
         16.a $max D'_u = \frac{T + location(u,1) - s}{num\_instance_u}$; /* relax $max D'_u$ to extend $LEFT_{u}$ */
         16.b $min D'_u = s - location(u, n_u - num\_instance_u)$;
            /* relax $min D'_u$ to extend $LEFT_{u}$ */
         16.c $\alpha_u = \text{MAX}\left(\frac{max D'_u - avg D_u}{avg D_u}, \frac{avg D_u - min D'_u}{avg D_u}\right)$; /* find a large enough $\alpha_u$ */
      17. }
   18. if $\alpha_u > RDC_{u}$ then /* $\alpha_u$ is relaxed too much */
      19. fail to find a schedule and reject $T$.
   20. allocate current slot $s$ to task $\tau_u$;
   21. $num\_instance_u = num\_instance_u - 1$;
   22. calculate the schedule window for the next instance of $\tau_u$ based on (3.11) and (3.12);
   23. }

/* for */

Figure 3.9: The modified Tpl_Sched Algorithm
Figure 3.10: Performance comparison between the original and the modified algorithm
Chapter 4

Time slot allocation schemes for dynamic tasks

The previous chapter studies real-time systems consisting of static tasks, where the parameters of all the tasks are known before the schedule is activated. Many general purpose real-time systems, however, allow the tasks to dynamically arrive and depart. For example, a video-on-demand application in a network system must give users the freedom to start and stop to use the service at any time.

In this chapter, we consider scheduling a dynamically arriving task on a single resource. Three different scheduling algorithms will be presented to allocate the time slots in a template to dynamic tasks by making greedy choices, non-greedy heuristic choices, and optimization choices, respectively.

The performances of the three algorithms are evaluated and compared. Since dynamic tasks usually need to be responded to in a short time, the time complexity of the scheduling algorithm is an important performance measurement. Another critical measurement is the system-wide acceptance ratio, which is defined as the percentage of the successfully scheduled tasks over the total number of tasks arriving in the system. The system-wide acceptance ratio indicates the capability of the scheduling algorithm for utilizing the system resource. Moreover, jitter control is an important performance issue for the QoS control in multimedia systems, such as audio/video presentations. In these applications, scheduling jitter, defined as the variance of the distances between all neighboring instances [28], must be low for a smooth service quality. We will evaluate and analyze the performance of the three algorithms according to the time complexity, the system-wide acceptance ratio and the scheduling jitter.

After deriving and evaluating the three algorithms for the basic task model, we discuss the issue of extending the task model by adding minimum distance constraints at Section 4.4.
4.1 Problem definition

In a dynamic system, when a new task arrives at the system, some time slots in the template are already allocated to the existing tasks. It is desirable that while the new task is scheduled in the same template, the allocation patterns of the already scheduled tasks are not changed, especially when the scheduling decisions need to be distributed to remote terminals, such as mobile objects. Moreover, if the slot allocations of the already scheduled tasks are changed in the template, some concerns need to be taken into consideration, including when to switch from the old allocation pattern to the new pattern, and how to guarantee the distance constraints of all the tasks when the switch occurs. Thus, we assume that the template size is fixed, and the slot allocations of the existing tasks will not be changed when a new task is accepted. In the systems that adopt this assumption, the template size is provided as a system parameter. For example, the radio communication system Link-16 has a fixed template size of 512 time slots. Once a task is accepted and allocated in the template, its schedule will not be changed until the task terminates. The problem is formally stated as follows.

Problem statement: Given a template of size $T$ and a set $\mathcal{S}$ of $m$ vacant time slots in the template, $\mathcal{S} = \{S_1, S_2, ..., S_m\}$, the problem is to allocate the vacant time slots to the instances of the new task $\tau_i = (avgD_i, maxD_i)$ such that the following conditions are satisfied.

1. There are $n_i$ instances of $\tau_i$ in the template where $n_i = \left\lceil \frac{T}{avgD_i} \right\rceil$.
2. $location(i, j) \in \mathcal{S}, \forall j, 1 \leq j \leq n_i$.
3. $location(i, j + 1) - location(i, j) \leq maxD_i, \forall j \geq 1$.

The first condition specifies that in order to satisfy the rate requirement, there should be $n_i$ instances in a template of size $T$, where $n_i = \left\lceil \frac{T}{avgD_i} \right\rceil$. Since it is expected that some other new task will arrive in the future, the current task should not occupy more slots than it needs so as to increase the schedulability of future tasks. Without this constraint, a simple way to solve the problem is to allocate every vacant time slot to the new task. Although this method can easily find a feasible schedule to satisfy both the rate requirement and the distance constraint of the new task, there is no resource left for future arriving tasks. The second condition indicates that only vacant time slots are available to be allocated to the new arriving task. The third condition specifies the distance constraint of the new task.

Since in this chapter, the template size $T$ is fixed, then the original task model $\tau_i = (avgD_i, maxD_i)$ is equivalent to the representation $\tau_i = (n_i, maxD_i)$. The average rate requirement is expressed in terms of the number of time slots the task needs in the template. In some systems, the application requirement is directly expressed in this way, such as in the Link-16 system. Later in this chapter, we will use the two equivalent forms of task model interchangeably.
Recall that, because the template is applied repeatedly to control the transmission of the tasks, if there are \( n_i \) instances of \( \tau_i \) in the template, then \( \text{location}(i, j + n_i) = \text{location}(i, j) + T \).

4.2 Allocation algorithms for dynamic tasks

In this section, we present three algorithms to solve the scheduling problem in a system that allows the tasks to dynamically arrive at and depart from. A new task with average rate requirement and maximum distance constraint is allocated in the available time slots of a template, where some other tasks have already been scheduled.

4.2.1 A greedy algorithm Dynamic_greedy

We first present a simple greedy algorithm, Dynamic_greedy, to schedule a dynamic task. An algorithm is \textit{locally optimal} if it can find a feasible scheme to schedule all the instances of the new task in the vacant time slots of the template if and only if there exists one. Dynamic_greedy is proven to be locally optimal in Section 4.2.1.2.

4.2.1.1 Description of the greedy algorithm

As defined previously, given a list of \( m \) vacant time slots \( S = \{S_1, S_2, ..., S_m\} \) in a template of size \( T \), the problem is to allocate \( n_i \) time slots to a new task \( \tau_i = (n_i, \text{max} D_i) \), and satisfy the maximum distance constraint. Assume that the list of vacant slots is sorted in increasing order, that is, \( S_i < S_j, \forall 1 \leq i < j < m \). In order to satisfy the distance constraint, there should be at least one instance of \( \tau_i \) in the first \( \text{max} D_i \) time slots at the beginning of the template. Otherwise, the inter-template distance constraint will be violated when the template is applied repeatedly. Therefore, \( \tau_{i,1} \), the first instance of \( \tau_i \), cannot be scheduled later than \( \text{max} D_i \). We define a set \( \text{legal}_1 \text{first}_i \) which consists of the valid choices for \( \tau_{i,1} \). That is, \( \text{legal}_1 \text{first}_i \) contains all the vacant time slots that are smaller than or equal to \( \text{max} D_i \).

\[
\text{legal}_1 \text{first}_i = \{S_1, ..., S_v\}, s.t. \ S_v \leq \text{max} D_i.
\]

If the set \( \text{legal}_1 \text{first}_i \) is empty, then clearly it is impossible to find a feasible solution for the new task. For the same reason, it is impossible to find a feasible solution if the distance between any two vacant time slots is larger than \( \text{max} D_i \).

As defined in the problem statement, \( n_i = \left\lceil \frac{T}{\text{avg} D_i} \right\rceil \) vacant time slots will be assigned to \( \tau_i \) to satisfy its rate requirement. Assume that the first instance of \( \tau_i \) is scheduled at time slot \( S_f \) (\( S_f \in \text{legal}_1 \text{first}_i \)), then the \((n_i + 1)^{th}\) instance is fixed at slot \( S_f + T \), due to the repetition of the template. The problem is reduced to scheduling the remaining \( n_i - 1 \) instances within the interval
The function $\text{min\_instance\_number}$ presented in Figure 4.1 will apply a greedy policy to select a set of the minimum number of instances that are needed in this interval to generate a feasible schedule. Note that if slot $S_i$ is vacant, then so is the slot $S_i + T$. If the function $\text{min\_instance\_number}$ selects a slot $S_j$ in the interval $(S_f, S_f + T)$ and $S_j > T$, it is equivalent to selecting $S_j - T$ within the template.

$$\text{min\_instance\_number}(S_f)$$

1. $num = 1$
2. $selected = S_f$
3. $schedule = \{S_f\}$
4. while $(T + S_f - selected > \max D_i)$, repeat
5. {
6. \hspace{1em} $limit = selected + \max D_i$
7. \hspace{1em} get the farthest un-selected vacant slot $S_x$, which is smaller than or equal to $limit$, that is, $selected < S_x \leq limit$ and $S_{x+1} > limit$
8. \hspace{1em} if no $S_x$ exists, return IMPOSSIBLE.
9. \hspace{1em} selected = $S_x$
10. \hspace{1em} schedule = schedule $\cup \{S_x\}$
11. \hspace{1em} num = num + 1
12. }
13. return $num$ and the set of selected slots in the schedule.

Figure 4.1: The $\text{min\_instance\_number}$ function

In function $\text{min\_instance\_number}$, after deciding the position of the first instance, a vacant slot is selected that is as far as possible from the first instance, but within the distance constraint for the next instance. The scheme is repeated until it reaches the end point, $\text{location}(i, n_i + 1)$. Specifically, in Figure 4.1, the variable $num$ records the number of instances that have been scheduled. The variable $selected$ represents the selected time slot for the previously scheduled instance. The variable $schedule$ is the set of selected vacant slots allocated to the instances of $\tau_i$. During the loop that starts from line 4, the farthest legal position, $limit$, is calculated for the next instance based on the location of the previously selected slot and the distance constraint (line 6). Then the position of the legal vacant time slot closest to the limit is located (line 7), and assigned to the next instance (line 9 and line 11). If no such time slot exists, then the function returns IMPOSSIBLE (line 8), indicating that it is impossible to find a feasible schedule for the current task with $\max D_i$.
constraint, because there is a time interval of length $max D_i$ that does not contain any vacant slot. The loop continues until the previously selected slot is close enough to the end point (line 4). Then the number of instances needed and the set of selected slots are returned (line 13).

The algorithm **Dynamic greedy** is presented in Figure 4.2. It first makes sure that the number of vacant slots, $m$, is sufficient for the number of instances of task $\tau_i$ in order to satisfy the rate requirement of the task (line 1). It selects a vacant slot $S_f$ from $legal_{first_i}$ as $location(i, 1)$ (line 2), and calls the function $min_{instance, number}$ with $S_f$ as a parameter (line 4). The algorithm rejects the new task if the function returns $IMPOSSIBLE$ (line 5), which indicates that the allocation pattern of the available slots cannot satisfy the maximum distance constraint of the new task. If the minimum needed number of instances to satisfy the distance constraint is smaller than or equal to the required instance number, $n_i$, then there exists a feasible schedule, which is constructed by the time slots selected in function $min_{instance, number}$ and $(n_i - \text{instance, needed})$ randomly selected ones from the remaining vacant slots (line 6-10). Since the vacant slots returned by function $min_{instance, number}$ already satisfy the distance constraint of $\tau_i$, and allocating more slots will not increase the distance between the selected slots, the algorithm randomly chooses $(n_i - \text{instance, needed})$ slots to make a total of $n_i$ slots. In this way, both the maximum distance constraint and the rate requirement are satisfied. Otherwise, if the minimum needed number of instances is already larger than $n_i$, then there is no feasible schedule that starts with $S_f$ (line 11). The next candidate in the set $legal_{first_i}$ is picked for the same procedure. If all the time slots in $legal_{first_i}$ are tried without success, then there is no feasible schedule, and the algorithm rejects the new task (line 13).

4.2.1.2 Proof of local optimality

In order to prove that algorithm **Dynamic greedy** is locally optimal for the new task, we first prove the correctness of function $min_{instance, number}$ in **Lemma 1**. That is, we prove that the function returns the minimum number of available time slots in the template to satisfy the maximum distance constraint of the new task.

**Lemma 1**: Function $min_{instance, number}(S_f)$ produces a set of minimum number of vacant time slots that satisfies the distance constraint $max D_i$ in the time interval $(S_f, S_f + T)$.

**Proof** Suppose that the set $B = \{S_f, b_1, b_2, \ldots\}$ with $S_f < b_1 < b_2 < ...$ is a correct set which begins with $S_f$ and contains the minimum number of time slots constrained by $max D_i$. Assume that the set of time slots generated by function $min_{instance, number}$ is $C = \{S_f, c_1, c_2, \ldots\}$ where $S_f < c_1 < c_2 < ...$. We can show that there is a correct solution, say $B'$, which also selects $c_1$ as the first slot after $S_f$. 
Dynamic_greedy(τᵢ, S)

1. if \( m < n_i \), no feasible schedule, reject \( τ_i \);
2. for every \( S_f \in legal_{first_i} = \{S_1, \ldots, S_c\} \), do
3.  
4. \( instance_needed = \min instance_number(S_f) \);
5. if it returns IMPOSSIBLE then no feasible schedule, reject \( τ_i \);
6. if \( \text{instance_needed} \leq n_i \)
7.  
8. randomly select \((n_i - \text{instance_needed})\) vacant slots for the remaining instances;
9. exit and output the schedule.
10. }
11. else, there is no feasible schedule starting with \( S_f \).
12. }
13. no feasible schedule, reject \( τ_i \);

Figure 4.2: The Dynamic_greedy Algorithm

Specifically, if \( B \) starts with \( c_1 \), that is, \( b_1 = c_1 \), then \( B' = B \). Otherwise, if \( b_1 \neq c_1 \), define \( B' = B - \{b_1\} \cup \{c_1\} \). We will prove that \( B' \) is a correct set.

Note that if \( b_1 > c_1 \), then the distance between \( S_f \) and \( b_1 \) is larger than \( \text{max} D_i \) which is not valid, because \( c_1 \) is the largest time slot within the \( \text{max} D_i \) constraint. So \( b_1 < c_1 \), since we are discussing the case \( b_1 \neq c_1 \). For the next selection \( b_2 \), if \( b_2 < c_1 \), then \( b_2 - S_f < \text{max} D_i \), which contradicts the fact that \( B \) is a correct set, because the set \( \{S_f, b_2, \ldots\} \) is a set of time slots that satisfies the \( \text{max} D_i \) constraint and it has one less time slot than \( B \). Thus, \( b_1 < c_1 < b_2 \). Figure 4.3 illustrates the relationship between \( B \) and \( B' \). Obviously, replacing \( b_1 \) with \( c_1 \) does not influence the validity of the schedule since \( e_1 - S_f \leq \text{max} D_i \) and \( b_2 - c_1 < b_2 - b_1 < \text{max} D_i \). So there is a correct solution, \( B' \), that selects \( c_1 \) as the first slot after \( S_f \).

Once the greedy choice of the first slot is made, the problem reduces to finding a correct solution in the time interval \((c_1, S_f + T)\). Therefore, after each greedy choice, we are left with a problem of the same form, but within a smaller interval. By induction on the number of selected slots, we can see that the greedy choice at every step produces a correct solution.

Now that we have proved the correctness of function \( \min instance_number(S_f) \), the proof of the optimality of algorithm Dynamic_greedy is trivial. Randomly selecting some vacant time slots in addition to the greedy choices will not violate the maximum distance constraint, and
the rate requirement is satisfied by enforcing the total number of instances in the template. So **Dynamic_greedy** will find a feasible schedule for $\tau_i$ if one exists.

The basic step of **Dynamic_greedy** is to make a greedy choice for each of the $n_i$ instances of $\tau_i$. If we use binary search, each choice needs $O(lg(max D_i))$ time to find the vacant slot that is closest to the limit derived from the maximum distance constraint in line 7 in Figure 4.1. In the worst case, there are $max D_i$ vacant slots in the set $legal_first_i$, and the procedure repeats until all the time slots in $legal_first_i$ are examined. So the worst case time complexity of the greedy algorithm is $O(n_i \times max D_i \times lg(max D_i))$.

### 4.2.1.3 An example of applying the greedy algorithm

Although we have presented a locally optimal algorithm with polynomial time complexity to generate a feasible schedule for a new task, it is an interesting issue to study how the instance allocation pattern of the current task will influence the schedulability of the tasks arriving in the future. The following example illustrates the characteristics of the allocation pattern and scheduling jitter of this algorithm.

**Example 4.1:** Assume in a template of 12 time slots, there are 9 vacant slots. The set of available slots is $S = \{2, 3, 5, 6, 7, 8, 10, 11, 12\}$, as shown in Figure 4.4. In Figure 4.4, the first 12 slots in the time line form the scheduling template which is repeated continuously. The shaded time slots indicate the already occupied slots. A task $\tau_i$ arrives with $avg D_i = 3$ and $max D_i = 6$. So 4 instances need to be scheduled in the template. In this case, the set $legal_first_i$ is $\{2, 3, 5, 6\}$, and the algorithm begins with choosing slot 2 for the first instance of $\tau_i$. In the function $min\_instance\_number()$ that starts with slot 2, the limit for the next greedy choice is $2 + max D_i = 8$. Slot 8 is vacant. Since the distance between slot 8 and the first instance in the second template, slot $2 + T = 14$, is smaller than or equal to $max D_i$, there is no need for further
search. Thus, the function $\text{min\_instance\_number()}$ returns the set of greedy choices $\{2, 8\}$. In order to satisfy the average rate requirement, 2 vacant slots are randomly selected from the remaining ones. In the worst case, the remaining 2 instances can be scheduled at slot 3 and 5, such that instances are clustered together, and 5 consecutive time slots are occupied, as shown in Figure 4.4.

The example shows that Dynamic greedy may generate irregular allocation patterns where some of the instances are scattered, while some other instances are clustered. The irregularity is especially significant when $\max D_i$ is large compared with $\text{avg} D_i$. This can lead to large scheduling jitters, as shown in the example where the largest distance between two instances is 6 and the smallest distance is 1. Moreover, the schedulability of future tasks can be affected. In the above example, assume that the next arriving task is $\tau_j = (4, 5)$. Its distance constraint of 5 cannot be satisfied and it will be rejected, because the cluster generated by the algorithm results in an interval of length 5 without a single vacant slot.

![template of size 12](image)

Figure 4.4: An example of the allocation pattern generated by Dynamic greedy.

### 4.2.2 A non-greedy heuristic algorithm Dynamic heuristic

Although the greedy algorithm presented in the previous section can efficiently find a feasible schedule for a dynamically arriving task, it may have large scheduling jitter, and small system-wide acceptance ratio because of the greedy choices. In this section, we present a non-greedy heuristic algorithm which is not locally optimal, but improves the scheduling jitter and the allocation pattern of the accepted task.

#### 4.2.2.1 Description of the heuristic algorithm

Algorithm Dynamic heuristic is described in Figure 4.5. Given a template that is partially occupied and a new task $\tau_i = (n_i, \max D_i)$, the algorithm selects $n_i$ vacant time slots that satisfy $\max D_i$ constraint, and returns location$(i, j), j = 1, \ldots, n_i$.

Same as in algorithm Dynamic greedy, a set of vacant time slots legal first $i$ consists of the valid possible locations for the first instance of task $\tau_i$. Dynamic heuristic starts by allocating one vacant time slot in legal first $i$ to the first instance of task $\tau_i$, $\tau_{i,1}$. Then the later instances are scheduled one by one, each within its schedule window, which is calculated according to the locations of the previous instances and the timing constraints. The window defines the time interval
within which the current instance $\tau_{i,j}$ can be scheduled in order to satisfy the timing constraints. Similar to the definition given in Section 3.1.3, we use $\text{LEFT}_{i,j}$ and $\text{RIGHT}_{i,j}$ to denote the left end and the right end of the scheduling window for $\tau_{i,j}$, and the schedule window is represented as $\text{window}_{i,j} = [\text{LEFT}_{i,j}, \text{RIGHT}_{i,j}]$.

After allocating a time slot to the $(j - 1)^{th}$ instance of $\tau_i$, $\text{location}(i, j - 1)$ is determined. The left end and the right end of $\text{window}_{i,j}$ are calculated in a way similar to what is used in the static algorithm $\text{Tpl.Sched}$ in Section 3.1.3, taking care of the maximum distance constraint in both the forward and the backward direction.

\[
\text{LEFT}_{i,j} = \text{MAX}(T + \text{location}(i, 1) - (n_i - j + 1) \times \text{max} \text{D}_i, \text{location}(i, j - 1) + 1) \quad (4.1)
\]

\[
\text{RIGHT}_{i,j} = \text{MIN}(\text{location}(i, j - 1) + \text{max} \text{D}_i, T + \text{location}(i, 1)) \quad (4.2)
\]

In Equation (4.1), the function $\text{MAX}()$ returns the maximum value among its arguments. The first argument in function $\text{MAX}()$ takes care of the distance constraint in the backward direction in the same way as Equation (3.5) in Section 3.1.3, which guarantees that the inter-template distance constraint between the last instance in the first template and the first instance in the second template is not violated. Equation (4.1) also makes sure that it is possible for all the unscheduled instances to satisfy the distance constraint within the template. In addition, since $\tau_{i,j}$ cannot be scheduled earlier than $\tau_{i,j-1}$, $\text{LEFT}_{i,j}$ cannot be smaller than $\text{location}(i, j - 1) + 1$, which is the second argument in the function $\text{MAX}()$.

In Equation (4.2), the function $\text{MIN}()$ returns the minimum value among its arguments. The first argument in function $\text{MIN}()$ makes sure that the distance between the current instance and the previously scheduled instance, $\tau_{i,j-1}$, does not exceed the maximum distance constraint. It is the same as the consideration of distance constraint in the forward direction described in Equation (3.4) in Section 3.1.3. Moreover, the current instance must be scheduled before the first instance in the second template, that is, $\text{RIGHT}_{i,j}$ cannot exceed $\text{location}(i, 1) + T$. This is the second argument in the function $\text{MIN}()$.

After the calculation of the scheduling window, one vacant slot in the window will be chosen for the next instance according to the algorithm shown in Figure 4.5. The choice is made according to the policy that the instances of task $\tau_i$ should be distributed as uniformly as possible. Once the $(j - 1)^{th}$ instance of $\tau_i$ is allocated, we are left with $n_i - j + 1$ instances to schedule in the time interval $(\text{location}(i, j - 1), T + \text{location}(i, 1))$, i.e., the interval between the location of the $(j - 1)^{th}$ instance and the first instance in the second template. Since the $n_i - j + 1$ instances will divide the interval into $n_i - j + 2$ segments, the average length of each segment is $\frac{T + \text{location}(i, 1) - \text{location}(i, j - 1)}{n_i - j + 2}$. The variable $avg$ is used to denote the average distance of the $n_i - j + 1$
unscheduled instances in the remaining time interval, and is calculated in step 4(c) of Figure 4.5. We consider the ideal position for the next instance to be \( \text{avg} \) slots away from the proceeding instance, such that the remaining instances can be distributed uniformly within the rest of the template. So the ideal position of \( \tau_{i,j} \) is \( \text{location}(i, j - 1) + \text{avg} \), as shown in step 4(d) of Figure 4.5. The vacant time slot that is closest to the ideal position in the schedule window is chosen as the location for the current instance in step 4(e). The procedure may fail to find a vacant time slot for the current instance in two cases. In the first case, if the maximum distance constraint cannot be satisfied in the remaining instances, then the algorithm cannot find a valid schedule window, that is, \( \text{LEFT}_{i,j} \) is larger than \( \text{RIGHT}_{i,j} \). In the second case, there is no vacant time slot in the schedule window. When either case occurs, the algorithm will choose the next vacant slot in the set \( \text{legal}\_first_i \) as the location of the first instance, and repeat the procedure. If no feasible schedule can be found after all of the vacant slots in \( \text{legal}\_first_i \) are examined, then the algorithm rejects the new task.

**Dynamic Heuristic**(\( \tau_i \))

1. If \( m < n_i \), the new task is rejected because the rate requirement cannot be satisfied.

2. If the set \( \text{legal}\_first_i \) is empty, then the new task is rejected because its distance constraint cannot be satisfied.

3. For every \( S_f \) in \( \text{legal}\_first_i = \{S_1, ..., S_v\} \), repeat the following steps until a feasible schedule is found for the new task. If no schedule is found for any \( S_f \in \text{legal}\_first_i \), \( \tau_i \) is rejected.

4. Allocate \( S_f \) to the first instance, and apply the following procedure to schedule all the remaining instances \( \tau_{i,j}, j \in [2, n_i] \).
   
   (a) Calculate \( \text{window}_{i,j} = [\text{LEFT}_{i,j}, \text{RIGHT}_{i,j}] \) according to Equations (4.1) and (4.2). If \( \text{LEFT}_{i,j} > \text{RIGHT}_{i,j} \), there is no valid schedule window. Go back to step 3 to try the next \( S_f \).

   (b) If there is no vacant slot in \( \text{window}_{i,j} \), then go back to step 3 to try the next \( S_f \).

   (c) Otherwise, calculate the average distance, \( \text{avg} \), of the remaining instances in the template.

\[
\text{avg} = \frac{T + \text{location}(i, 1) - \text{location}(i, j - 1)}{n_i - j + 2}
\]

   (d) Calculate the ideal position of the instance as \( \text{location}(i, j - 1) + \text{avg} \).

   (e) Schedule instance \( \tau_{i,j} \) at the vacant slot that is closest to the ideal position in \( \text{window}_{i,j} \).

Figure 4.5: The **Dynamic Heuristic** Algorithm
The basic step of Dynamic greedy is to make a heuristic choice for each of the \( n_i \) instances of \( \tau_i \). If we use binary search, each choice needs \( O(lg(max D_i)) \) time to find the vacant slot that is closest to the ideal position. In the worst case, the procedure will be repeated \( max D_i \) times when all the time slots in \( legal_{first_i} \) are examined. Thus, algorithm Dynamic heuristic has a time complexity of \( O(n_i \times max D_i \times lg(max D_i)) \), which is the same as the time complexity of Dynamic greedy.

### 4.2.2.2 An example of applying the heuristic algorithm

Algorithm Dynamic heuristic is not locally optimal for the current task. That is, Dynamic heuristic may reject a task when there exists a feasible schedule for it in the current template. However, the heuristic scheduling policy of choosing a vacant slot close to the ideal position distributes the instances in the template more uniformly than the one generated by the greedy policy in Dynamic greedy. Thus, it decreases the scheduling jitter of the accepted tasks and prevents the instances from clustering, which may be beneficial for the acceptance of the later incoming tasks.

Apply Dynamic heuristic to Example 4.1 given in Section 4.2.1. In a template of size 12, the vacant slots \( S = \{2, 3, 5, 6, 7, 8, 10, 11, 12\} \) are available to schedule a new task \( \tau_i \) with \( avg D_i = 3 \) and \( max D_i = 6 \). We start with allocating the first slot in the set \( legal_{first_i} \), slot 2, to \( \tau_i,1 \). Since the average distance for the remaining instances is still 3, the ideal position for the second instance is at slot 5 and it is vacant. So the second instance is allocated at 5. By continuing to apply the algorithm for the remaining instances, we obtain a uniform schedule of the 4 instances at slot 2, 5, 8, 11 which is shown in the time line in Figure 4.6. So the scheduling jitter is 0 in this case. Moreover, because clustering is prevented, the next incoming task \( \tau_j \) with \( avg D_j = 4 \) and \( max D_j = 5 \) that is rejected by Dynamic greedy in Example 4.1, can now be accepted by Dynamic heuristic. The scheduling result of \( \tau_j \) by applying Dynamic heuristic is shown in Figure 4.6 at slots \( \{3, 7, 10\} \).

![Figure 4.6: An example of the allocation pattern generated by Dynamic heuristic.](image)

### 4.2.3 An optimization algorithm Dynamic optimization

Although the heuristic algorithm is not locally optimal, we have observed that it has smaller jitter and larger potential for accepting more tasks in the future, compared with the greedy
algorithm which is locally optimal. This leads us to explore an algorithm that combines the advantages of both algorithms. In this section, we transform the original problem into an optimization problem. The solution to the optimization problem, Dynamic optimization, is locally optimal for the new task as the greedy algorithm. It also improves the scheduling jitter and the allocation pattern as does the heuristic algorithm.

4.2.3.1 Description of the optimization algorithm

Given a set of \( m \) vacant time slots \( S = \{S_1, S_2, \ldots, S_m\} \) in a template of size \( T \), and a new task \( \tau_i = (n_i, \max D_i) \), we transform the problem of selecting \( n_i \) vacant time slots constrained by \( \max D_i \) into an equivalent graph problem.

As defined earlier, legal \( \text{first}_i \) contains all the vacant time slots that are smaller than or equal to \( \max D_i \). In order to satisfy the maximum distance constraint, the first instance of \( \tau_i \) must be scheduled at a vacant slot that belongs to the set legal \( \text{first}_i \). Assume that the location of the first instance of \( \tau_i \) is at slot \( S_f \) (\( S_f \in \text{legal first}_i \)). Then the time slot \( S_f + T \) is dedicated to the \((n_i + 1)\)th instance. The objective is to select \( n_i - 1 \) vacant slots from the remaining \( m - 1 \) vacant slots in the interval \( (S_f, S_f + T) \), such that the distance constraint is satisfied. This problem can be transformed to a problem on a graph \( G = (V, E) \) where \( G \) is an acyclic directed graph with \( m + 1 \) vertices. \( V \) is the set of vertices and \( E \) is the set of edges. The vertices represent the \( m + 1 \) vacant slots in the interval \([S_f, S_f + T]\), that is, \( V = \{V_1, V_2, \ldots, V_{m+1}\} \) where vertex \( V_1 \) represents \( S_f \), the first vacant slot in the interval, vertex \( V_2 \) represents the second one, and so on, and vertex \( V_{m+1} \) represents \( S_f + T \), the last one in the interval.

For simplicity of notation, we use \( \text{dis}(V_a, V_b) \) to denote the distance between slot \( S_a \) and slot \( S_b \), where \( S_a \) and \( S_b \) are the vacant time slots represented by \( V_a \) and \( V_b \) in the graph, respectively. We define a directed edge \( E_{a,b} \) from vertex \( V_a \) to \( V_b \) if and only if \( a < b \), and \( \text{dis}(V_a, V_b) \leq \max D_i \). That is, if the distance between the two corresponding time slots does not satisfy the maximum distance constraint, there is no edge between \( V_a \) and \( V_b \). Thus, the original scheduling problem is equivalent to the problem of finding a directed path from vertex \( V_1 \) to \( V_{m+1} \) consisting of \( n_i \) edges. Such a path exists if and only if there exists a feasible schedule in the template for the \( n_i \) instances of \( \tau_i \) with \( \max D_i \) constraint. The vertices on the path correspond to the selected vacant slots in the feasible schedule.

The problem transformation is illustrated in Figure 4.7. In the figure, a template of size 12 has \( m = 5 \) vacant time slots, \( \{1, 4, 6, 8, 10\} \). Assume that the new task is \( \tau_i \) with \( n_i = 3 \) and \( \max D_i = 5 \), and thus \( \text{avg} D_i = 4 \). If \( \tau_{i,1} \) is allocated at slot 1, i.e. \( S_f = 1 \), then the first instance in the next template, \( \tau_{i,4} \) is fixed at slot 13. So vertex \( V_1 \) represents slot 1 and \( V_6 \) represents slot 13. The transformation result is shown in Figure 4.7 below the time line. The original problem of
selecting 3 vacant slots with the constraint of \( \max D_t = 5 \) is now transformed to finding a path from \( V_1 \) to \( V_6 \) with 3 edges. The darkened edges in the figure constitute a path from \( V_1 \) to \( V_6 \) that have three edges. The set of vertices on the path is \( \{ V_1, V_3, V_5, V_6 \} \), which corresponds to a set of vacant time slots \( \{1, 6, 10, 13\} \). Because of the equivalence of the two problems, a feasible schedule result for \( \tau_i \) is the set of vacant slots in the interval \([S_f, S_f + T]\) that are represented by the vertices on the path, which is \( \{1, 6, 10\} \).

![Figure 4.7: An example of the graph transformation.](image)

Although any solution to the transformed graph problem is sufficient to solve the original scheduling problem, we will show that the scheduling jitter can be minimized in addition to satisfying the average rate requirement and the maximum distance constraint. We further define the weight of edge \( E_{a,b} \) according to the definition of the scheduling jitter. For simplicity of calculation, we adopt the definition of absolute jitter [79], where jitter of task \( \tau_i \) is defined as

\[
\sum_{i=2}^{n} \frac{|\text{location}(i,j) - \text{location}(i-1,j) - \text{avg}D_i|}{n_i}
\]

(The formal definition of scheduling jitter based on the calculation of variance will be used in Chapter 5, where a variation of the graph transformation technique is explained.) Let \( w(E_{a,b}) \) denote the weight of the edge \( E_{a,b} \). If the edge \( E_{a,b} \) exists in the graph, \( w(E_{a,b}) = |\text{dis}(V_a, V_b) - \text{avg}D_i| \). Otherwise, the weight of edge \( E_{a,b} \) is \( \infty \), indicating that the edge \((V_a, V_b)\) does not exist. If the distance between slot \( S_a \) and slot \( S_b \) is close to the average distance \( \text{avg}D_i \), then the value of \( w(E_{a,b}) \) is small. The weight of a path is the summation of the weights of all the edges on the path. Intuitively, a path of small weight implies that the allocation pattern of the instances is close to being uniform. According to the definition of the absolute jitter, the shortest path which has the smallest path weight from \( V_1 \) to \( V_{m+1} \) with \( n_i \) edges corresponds to a feasible schedule of \( \tau_i \) with minimum jitter. Therefore, in order to decrease the scheduling jitter and to uniformly schedule the instances, the original scheduling problem is solved by the result of the following optimization problem.

**Optimization problem:** Given a directed acyclic graph \( G = (V, \mathcal{E}) \) where \( V = \{V_1, V_2, ..., V_{m+1}\} \), and each edge \( E_{a,b} \in \mathcal{E} \) has an assigned weight, the problem is to find the minimum weight path from \( V_1 \) to \( V_{m+1} \) that consists of \( n_i \) edges.
We solve the optimization problem by a dynamic programming techniques [21]. Let us define a two dimensional table $\text{shortest}$ of size $(m + 1) \times (n_i + 1)$, where each table entry $\text{shortest}(x, y)$ contains the weight of the shortest path from $V_1$ to $V_x$ that has $y$ edges. The value of the index $x$ is in the range of $[1, m + 1]$ and the value of the index $y$ is in the range of $[0, n_i]$. Each table entry is calculated based on the following recursive relation.

- $\text{shortest}(1, 0) = 0$. The shortest path from $V_1$ to itself with 0 edges is set to be 0.
- $\text{shortest}(1, y) = \infty$, if $y \neq 0$. The shortest path from $V_1$ to itself with a non-zero edges does not exist.
- $\text{shortest}(x, 0) = \infty$, if $x \neq 1$. The shortest path from $V_1$ to another node $V_x$ with 0 edges does not exist.
- $\text{shortest}(x, y) = \min_{u \leq x} \{\text{shortest}(u, y - 1) + \text{weight}(E_{u,x})\}$, if $x \neq 1$ and $y \neq 0$.

The formula in the last item is based on the observation that the shortest path from $V_1$ to another vertex $V_x$ with $y$ edges must be formed by concatenating two components. The first one is the shortest path from $V_1$ to a certain preceding vertex $V_u$ ($u < x$) with $y - 1$ edges. The second component is the edge $E_{u,x}$ between $V_u$ and $V_x$. The optimal value of $\text{shortest}(x, y)$ is calculated by taking the minimum path weight among all the possible paths. Note that the number of possible $V_u$’s cannot be larger than $\max D_i$, because there is no edge connecting $V_u$ and $V_x$ if $\text{dis}(V_u, V_x)$ exceeds $\max D_i$. The table entry $\text{shortest}(m + 1, n_i)$ is the solution to the optimization problem.

Algorithm Dynamic_optimization is described in Figure 4.8. The algorithm that solves this optimization problem is a locally optimal algorithm for the new task $\tau_i$. Since the recursive calculation needs to be executed at most $\max D_i$ times for all the time slots in the set legal_first$_i$, and each table entry takes at most $O(\max D_i)$ time to compute, the time complexity of this algorithm is $O(m \times n_i \times \max D_i^2)$. It is larger than the time complexity of the other two algorithms. When the system resource is highly utilized such that the number of vacant slots is small, the extra time overhead can be insignificant.

### 4.2.3.2 An example of applying the optimization algorithm

Apply Dynamic_optimization to solve the problem given in Example 4.1 in Section 4.2.1, where the vacant slots $S = \{2, 3, 5, 6, 7, 8, 10, 11, 12\}$ in a template of size 12 are available to schedule a new task $\tau_i$ with $\text{avg} D_i = 3$ and $\max D_i = 6$. According to the definition of the edge weight that considers distribution pattern of the instances, a jitter free schedule corresponds to a path of weight 0 in the transformed graph problem. Since the algorithm Dynamic_optimization searches for the shortest path in the graph problem, it will return the jitter free schedule if there exists one.
Dynamic_optimization(τ_i)

1. If \( m < n_i \), the new task is rejected because the rate requirement cannot be satisfied.

2. If the set legal_first is empty, then the task is rejected because its distance constraint cannot be satisfied.

3. For every \( S_f \) in legal_first = \{S_1, ..., S_e\}, solve the transformed optimization problem, until a feasible schedule is found for the new task.

Figure 4.8: The **Dynamic_optimization** Algorithm

Therefore, by applying the optimization algorithm to Example 4.1, the result is a uniform schedule at slot 2, 5, 8, and 11, the same as the one generated by the heuristic algorithm in Figure 4.6. Moreover, because the occupied slots are prevented from clustering, the next incoming task \( τ_j \) with \( avgD_j = 4 \) and \( maxD_j = 5 \) can be accepted by **Dynamic_optimization**. Recall that in Section 4.2.1, \( τ_j \) is rejected by **Dynamic_greedy**.

### 4.3 Performance evaluation and comparison

All three algorithms presented in this section are simulated in an environment where tasks dynamically arrive and depart. For each message stream \( M_i = (avgD_i, maxD_i) \), \( avgD_i \) is randomly generated with a uniform distribution in the range \([5, 100]\). Template size \( T \) is randomly generated in the range \([500, 1000]\). In the same way as described in Section 3.1.4 for static task simulations, \( maxD_i \) is calculated according to \( avgD_i \) and the relative distance constraint \( RDC \), which is defined as \((maxD_i - avgD_i)/avgD_i\). \( RDC \) is one of the input parameters of the simulator. Moreover, for dynamic tasks, we need to define the connection duration of each task and inter-arrival time between two dynamically arriving task requests. The connection duration of \( τ_i \) is uniformly distributed between 2000 time slots and 5000 time slots. The average system workload is another input parameter to the simulator. Given a certain average system workload, the inter-arrival time of the tasks can be calculated according to the transmission rates of the arriving tasks and the average connection duration.

The system-wide acceptance ratio of the three algorithms when the average system workload is equal to 30\%, 50\%, 70\% and 90\% is plotted in Figure 4.9(a), Figure 4.9(b), Figure 4.9(c), and Figure 4.9(d), respectively. The X-axis of the figures represents the input parameter \( RDC \) which is in the range of \([0, 1]\). The Y-axis represents the system-wide task acceptance ratio, which
is the percentage of successfully scheduled tasks over 10,000 incoming tasks. The three curves in each figure plot the performance of the three algorithms.

Figure 4.9: The acceptance ratio v.s. RDC

When $RDC$ is close to 0, the choices of the possible scheduling pattern is limited, because of the restrictive distance constraints. In this case, local optimality is an important factor for the acceptance ratio. This is why the heuristic algorithm has the worst performance because it is not locally optimal. When $RDC$ is increased, the acceptance ratios of all three algorithms are increased, due to the larger flexibility generated by the relaxed distance constraints. However, in this case, the performance of the heuristic algorithm is improved faster than that of the greedy algorithm, because the allocation pattern generated by the greedy algorithm may hurt the future incoming tasks, as explained earlier in Example 4.1. This shows that the scheduling pattern plays a more important role than the local optimality when the distance constraints are not tight. In all the four figures, algorithm **Dynamic optimization** has the highest system-wide acceptance ratio among the three algorithms,
because it is locally optimal for each arriving task, and it prevents the irregular allocation pattern by solving the optimization problem. This performance advantage of Dynamic optimization is achieved at the cost of time complexity.

In addition to the three algorithms, we also apply an algorithm, such as EDF, which can fully utilize the network resource, and accept the new task as long as there are enough vacant slots in the template to satisfy the transmission rate requirement of the new task. The point on the vertical line of $RDC = 1$ is the acceptance ratio of EDF. Though EDF can fully utilize the resource and achieve the highest acceptance ratio, it cannot guarantee the distance constraints when $RDC < 1$, since the distance between two consecutive instances in a EDF schedule can be as large as twice the length of the period, as pointed out in Chapter 2. At the point of $RDC = 1$, the acceptance ratio of Dynamic optimization is close to the performance of EDF, the highest possible value, even when the system workload is high. Note that the workload indicated in Figure 4.9 is the average load, and thus the actual workload at any given instance of time can be smaller than or larger than the average load. This is why EDF fails to accept some tasks even when the average load is less than 1.

For each accepted task, the simulator measures the scheduling jitter which is the variance of the distances between all neighboring instances. Figure 4.10(a) and Figure 4.10(b) show the average scheduling jitter of all the accepted tasks versus the relative distance constraint when the average system workload is 0.5 and 0.9, respectively. The simulation results under average workload of 30% and 70% follow a similar trend and are not shown in this section. In the figure, the curves of the optimization algorithm and the heuristic algorithm are almost flat, which indicates
that the average jitter is almost the same even when a large distance constraint is allowed by the relaxed $RDC$. On the other hand, the jitter of the greedy algorithm rapidly increases with the increase of $RDC$, especially when the system workload is high. Again, this shows the importance of the allocation pattern for the QoS results. Among the three curves in Figure 10(a) and Figure 10(b), the optimization algorithm has the smallest jitter because the original problem is transformed to a graph optimization problem that searches for the shortest path. The performance of the heuristic algorithm has a scheduling jitter which is very close to that of the optimization algorithm, due to scheduling policy of Dynamic heurist that selects slots based on the ideal positions.

4.4 Adding minimum distance constraint to the task model

The three algorithms derived earlier in this chapter are based on the basic task model where only maximum distance constraint are considered. In this section, the basic task model is extended by adding the minimum distance constraint. We will discuss whether the three algorithms can be modified to solve the new problem with extended task model.

As introduced in Section 3.3, a task defined by the extended task model is represented as $\tau_i = (\text{avg}D_i, \text{min}D_i, \text{max}D_i)$, where $\text{min}D_i \leq \text{avg}D_i \leq \text{max}D_i$.

The algorithm Dynamic greedy makes greedy choices based on the maximum distance constraint. It cannot be applied to take care of both maximum and minimum distance constraints. However, note that a similar greedy policy can be used to solve problems that specify only minimum distance constraints.

In the heuristic algorithm Dynamic heuristic, the calculation of the ideal position is based on the current average distance among the remaining instances in the template. Therefore, the addition of minimum distance constraint will not influence the policy of finding the ideal position and selecting the vacant slot that is closest to the ideal position. However, the calculation of the schedule window is only based on the maximum distance constraint in Section 4.2.2. The following formula defines the left and right boundaries of a valid schedule window for $\tau_{i,j}$, considering both $\text{max}D_i$ and $\text{min}D_i$.

- $LEFT_{i,j}$ takes care of the minimum distance constraint concern in the forward direction, and the maximum distance constraint concern in the backward direction.

$$LEFT_{i,j} = \text{MAX}(\text{location}(i, j - 1) + \text{min}D_i, T + \text{location}(i, 1) - (n_i - j + 1) \times \text{max}D_i) \quad (4.3)$$
• $\text{RIGHT}TT_{i,j}$ takes care of the maximum distance constraint concern in the forward direction, and the minimum distance constraint concern in the backward direction.

$$\text{RIGHT}TT_{i,j} = \text{MIN}(\text{location}(i, j - 1) + \max D_i, T + \text{location}(i, 1) - (n_i - j + 1) \times \min D_i) \quad (4.4)$$

In the algorithm Dynamic optimization, the original scheduling problem is transformed to a graph problem, such that a good scheduling pattern can be found by searching for the shortest path in the corresponding graph. During problem transformation, vertices of the graph are generated based on the vacant time slots in the template. Edges are constructed according to the distance constraint. Since we include the minimum distance constraint in this section, the definition of a directed edge is changed accordingly. That is, a directed edge $(V_a, V_b)$ that connects from vertex $V_a$ to $V_b$ exists if and only if $a < b$, and the distance between the two corresponding time slots is in the range $[\min D_i, \max D_i]$. The other parts of the algorithm, such as the definition of the edge weight, and the dynamic programming technique, can still be applied to schedule tasks with the extended model.

The modified algorithms are simulated with the same input parameters as described for the original simulator in Section 4.3. The distance constraints of each task $\tau_i$ are calculated according to a randomly generated $\text{avg} D_i$ and a relative distance constraint $\text{RDC}_i$. That is, $\max D_i = \text{avg} D_i (1 + \text{RDC}_i)$. If $\text{RDC}_i < 1$, $\min D_i = \text{avg} D_i (1 - \text{RDC}_i)$. Otherwise, if $\text{RDC}_i \geq 1$, $\min D_i = 1$. The comparison of the system-wide acceptance ratio between the original optimization algorithm and the modified optimization algorithm under an average workload of 50% and 90% is shown in Figure 4.11(a) and Figure 4.11(b), respectively. The comparison follows a similar trend to the performance comparison between the two models for static tasks in Figure 3.10. The acceptance ratio of the original task model is better than the modified task model under the same simulation configuration, because the addition of $\min D_i$ makes it more difficult to find a feasible schedule. However, when $\text{RDC} = 0$ and when $\text{RDC} \geq 1$, both task models request the same distance constraints, and the performances of the two models are the same. The difference between the performances of the two models is smaller under a low system workload than that of a high system workload, because the effect of the $\min D_i$ constraint is more significant in the latter case. The performance comparison of the heuristic algorithm is similar to Figure 4.11, and is not shown here.

### 4.5 Summary

In this chapter, I derived three algorithms, Dynamic greedy, Dynamic heuristic and Dynamic optimization to schedule dynamic real-time tasks with average rate requirements and
distance constraints. Both minimum distance constraint and maximum distance constraint are discussed for the three algorithms. The performance of the three algorithms is evaluated and compared in terms of time complexity, acceptance ratio and scheduling jitter via simulation. The characteristics of the three algorithms are summarized in Table 4.1.

<table>
<thead>
<tr>
<th>algorithm</th>
<th><strong>Dynamic greedy</strong></th>
<th><strong>Dynamic heuristic</strong></th>
<th><strong>Dynamic optimization</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>both minD and maxD</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>local optimal</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>time complexity</td>
<td>$O(n_i \cdot \max D_i \cdot \lg(\max D_i))$</td>
<td>$O(n_i \cdot \max D_i \cdot \lg(\max D_i))$</td>
<td>$O(m \cdot n_i \cdot \max D_i^2)$</td>
</tr>
<tr>
<td>scheduling jitter</td>
<td>large</td>
<td>small</td>
<td>smallest</td>
</tr>
<tr>
<td>acceptance ratio</td>
<td>bad in most situations</td>
<td>good in most situations</td>
<td>best in all situations</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of the three algorithms

The comparison of the three algorithms shows that the heuristic algorithm is suitable for systems that need to schedule dynamic real-time traffic with low run time overhead and small scheduling jitter. The acceptance ratio of such a system can be improved by switching to the greedy algorithm when the distance constraint of the arriving task is tight. The optimization algorithm should be used in systems where the acceptance ratio is critical, while a certain scheduling overhead is tolerable, or when the time slots in the template are highly utilized such that the time overhead is not significant.
Chapter 5

Scheduling distributed tasks with destination continuity requirement

In the previous chapter, we have studied the problem of scheduling dynamic tasks on a single resource. This chapter extends the problem by considering communication tasks in a distributed system that allows message streams to dynamically arrive and depart. Each isochronous message stream is transmitted from a source node to a destination node via several intermediate nodes. A specific application requirement that will be addressed in this chapter is the destination continuity requirement, that is, the application at the destination node is required to process the packets continuously at a regular rate [27]. The motivation of the problem and the definition of the system model will be introduced in Section 5.1 and Section 5.2.

The scheduling algorithms presented in Chapter 4 allocates time slots to a dynamically arriving task on a single resource, such that the average rate requirement and distance constraint of the new task are satisfied. In this chapter, the message streams are characterized by a general model that only specifies average transmission rate requirements, with no explicit distance constraint. However, since distance constraint is closely related to scheduling jitter, we adapt the algorithms presented in Chapter 4 to meet the average rate requirement of each message stream, and to minimize scheduling jitter at each intermediate node. The algorithms will be described in Section 5.3.

Once a schedule is generated for a message stream at every intermediate node, the packets of the message stream are delivered in the allocated time slots according to one of the three delivery protocols that will be proposed in Section 5.4. A worst case end-to-end (ETE) delay bound is derived for each protocol to provide ETE delay analysis. The performance of applying our scheduling algorithm to different delivery protocols is compared and evaluated via simulation in Section 5.5. The results show that by minimizing scheduling jitter, we can achieve an end-to-end delay which is much better than the one provided by the worst-case analysis.

Since this chapter focuses on the study of communication tasks, we will use $M$ to denote a message stream. However, the algorithm and protocols derived in this chapter can also be applied to schedule computation or controlling tasks in the distributed transaction model introduced in Section
2.1.4.2, where data needs to be transmitted and processed through a pipeline of distributed nodes, such as sensors, processors, actuators and controllers.

5.1 Destination continuity requirement and problem introduction

This chapter addresses the issue of scheduling real-time streams in a distributed system where TDMA is used. TDMA and its variations have been widely used in various network architectures implemented for transmission of digital information over wired, optical or wireless communication channels. TDMA is also adopted as the network MAC protocol in distributed real-time systems, such as the MARS system [34]. As introduced in Chapter 2, our template-based scheduling paradigm provides a time slot allocation scheme for TDMA protocols.

In some applications, the source node repeatedly transmits data as packets of a message stream at a certain rate. Each of the intermediate nodes processes the packets and transmits the results to the next node. The destination node has a continuity requirement, which enforces continuous processing of the packets at a regular rate. For example, in a distributed military avionics system, data collected by the sensors is sent through several processors to a controller where an assessment task is periodically invoked to process the data and make corresponding decisions [58]. Another application with destination continuity requirement is the video-on-demand systems where the client node needs to playback the video frames regularly at a rate of 30 frames per second, in order to maintain a smooth human perception of the video.

The achievement of the destination continuity requirement depends on the predictability of packet transmissions. If the destination node cannot be sure about the arriving time of future packets, the only solution to satisfy the destination continuity requirement is to buffer all of the packets of the message stream at the destination node before it starts to process them. Obviously, this can lead to a large ETE delay. In this chapter, the ETE delay from the source node to the destination is the main concern. In order to achieve the destination continuity requirement within a limited ETE delay, we propose in this chapter a TDMA scheduling algorithm and the transmission protocols.

The ETE delay is defined as the length of the interval between the time when the source node begins to transmit the packets of a message stream and the time when the destination node begins to continuously process the packets. It is calculated as the sum of the following components.

1. Transmission and propagation delay on each link.
2. Processing delay in each intermediate node from the source to the destination.
3. Buffer delay (or waiting time) in each intermediate node.
4. **start-up delay** at the destination node.

Because the transmission and propagation delay does not depend on the scheduling discipline used, we will factor them out of our analysis. Once a stream is accepted, a scheduling TDMA template is generated such that the transmissions of the packets are controlled accordingly. Thus, there is no scheduling overhead. The processing delay at each node is assumed to be uniform, and can be factored out. Consequently, our focus lies in the last two items. The buffer delay is determined by the scheduling policy and the delivery protocol. In order to satisfy the continuity requirement, the destination node needs to accumulate some packets before it delivers the first packet to the processing application, which causes the **start-up delay**.

In some applications, such as the video-on-demand example, the ETE delay is not critical since the user may be able to tolerate a certain delay before the packets begins to be processed. However, a long ETE delay will compromise the user satisfaction. We consider this as an application with **soft** ETE delay constraints. On the other hand, in some critical real-time systems, such as the military avionics application example, violation of the ETE delay constraints may lead to serious system misbehavior. A stringent ETE deadline is specified between the sending of the packets and the processing of packets at the destination node [58]. This type of applications with stringent ETE deadline are said to have **hard** ETE delay constraints.

In order to control the magnitude of the ETE delay, we derive a TDMA scheduling algorithm to reserve time slots for packet transmission. The time slots in a TDMA template are allocated to message streams according to a scheduling algorithm. Because of the destination continuity requirement, the intermediate nodes should not transmit the packets at a rate that is smaller than the processing rate at the destination. Thus, the scheduling algorithm needs to guarantee a certain transmission rate. Moreover, scheduling jitter is the important concern for scheduling algorithms in most communication applications, which is defined as the variance of the temporal distances between all neighboring time slots allocated to the stream [55]. A smaller jitter implies a smoother transmission pattern for the stream, which means that the destination continuity requirement is easier to achieve. In order to obtain a small ETE delay, the objective of the scheduling algorithm in this chapter is to satisfy the rate requirement and to minimize the scheduling jitter.

We focus on communication environments where message streams dynamically arrive and depart. In Chapter 4, several solutions are derived to schedule dynamic tasks with distance constraints. We modify these algorithms in this chapter to generate a schedule that satisfies the average rate requirement and minimizes the jitter, subject to the current available time slots in the template.

After the schedule is determined by the scheduling algorithm, the packets of a message stream will be buffered at each intermediate node before they are transmitted in the allocated time
slots. In order to achieve the destination continuity requirements, we describe three delivery protocols. The first one is NED, standing for No Extra Delay at the intermediate nodes. At the intermediate nodes, each arriving packet of a message stream will be sent out immediately in the next available time slot that is allocated to this stream according to the schedule. This protocol completely relies on the destination node to smooth the jitter before it starts to process the packets. The second protocol is called WED (With Extra Delay at the intermediate nodes), in the sense that the intermediate nodes may delay the arriving packets for a certain time, even if there are available allocated time slots for the packet. The objective is to smooth the traffic on its route and reduce the start-up delay at the destination node. In the third protocol, Low-Delay-NED, the intermediate nodes deliver packets with no extra delay, as in the NED scheme, but the destination node has an efficient start-up scheme. We will describe the three protocols, and derive a worst case delay bound for each of them. The three schemes will be compared in terms of advantages and disadvantages, and their performance will be evaluated via simulations.

5.2 System model

As stated in the Section 5.1, we are concerned here with the problem of satisfying the destination continuity and ETE delay requirements of real-time message streams in a distributed system. Since we focus on the current arriving message stream in this chapter, the index of the message stream is not important. Thus, we use $M$ to denote a new message stream, and $M_j$ to denote the $j^{th}$ packet of $M$. The packets of $M$ are transmitted according to a scheduling template, where the slot that is allocated to the $j^{th}$ packet of $M$ is denoted as $location(j)$. Because of cyclical application of the template, the slot allocation pattern in the interval $[t, t + T - 1]$, for any $t$, will be repeated in every interval $[t + kT, t + (k + 1)T - 1]$, for $k > 0$. We assume that all nodes in the system have the same template size, and the template size $T$ is supplied as a system parameter.

The packets of a message stream are transmitted from the source node to the destination node through a set of intermediate nodes. The number of hops on the route is denoted by $h$, and the nodes on the route are labeled as $N_0, N_1, ..., N_h$ with $N_0$ being the source node and $N_h$ the destination node. The link $L_{k,k+1}$ connects node $N_k$ and node $N_{k+1}$.

We define the allocation pattern of $M$ at a node as $\{location(1), ..., location(n)\}$, the set of time slots that are allocated to $M$ by the scheduling algorithm at the node. The input pattern of a stream $M$ at a node is defined as the pattern in which the packets of $M$ are received by the node. Similarly, the output pattern at a node refers to the pattern in which the packets of $M$ are transmitted from the node. The output pattern of a node may or may not be equivalent to the allocation pattern, according to different delivery protocols that will be described later. We assume that the transmission and propagation delay can be ignored on the intermediate nodes, as explained
in Section 5.1. Thus, the output pattern of node \( N_k \) is equivalent to the transmission pattern on the link \( L_{k,k+1} \), which is also equivalent to the input pattern of the next node \( N_{k+1} \). That is, if node \( N_k \) sends a packet in time slot \( t \), then the packet is propagated on link \( L_{k,k+1} \) and received by node \( N_{k+1} \) in slot \( t \). The purpose of this assumption is to simplify the discussion. The results will not be influenced if this assumption is relaxed, as long as the ignored delays are constants.

A general connection-oriented communication protocol consists of three parts, the connection establishment, the delivery discipline at the intermediate nodes, and the start-up discipline at the destination node.

At the connection establishment stage, the source node initiates a request for a new message stream \( M \). As described in Chapter 4, in systems where the template size is fixed, applications specify the average transmission rate in terms of \( n \), the number of time slots that need to be allocated in the template of size \( T \). Thus, each stream \( M \) is characterized as \( M = (n, d) \) where \( n \) is the average rate requirement and \( d \) is the ETE deadline of \( M \), indicating that the actual ETE delay must be smaller than or equal to \( d \). The average transmission rate of \( M \) can also be represented by \( \text{avg} D = \frac{T}{n} \), the average distance between any two consecutive packets of \( M \) in the template. Therefore, an equivalent characterization of message stream \( M \) is \( M = (\text{avg} D, d) \). In order to prevent the overflow or underflow of the receiving buffer at the destination node, the processing rate at the destination is equal to the average transmission rate at each intermediate node.

Upon receiving the request of a new message stream, an intermediate node reserves time slots in the template by executing the scheduling algorithm. If there is no sufficient bandwidth for the new stream at any node, then the request is rejected. When the request reaches the destination node, the ETE delay is computed. If the actual ETE delay is larger than the specified ETE deadline \( d \), the request is also rejected.

After a request is accepted, the source node begins to transmit the packets of the stream. We assume that the source node delivers the packet continuously at a regular rate of \( n \) packets in every \( T \) time slots. When a packet is received by an intermediate node, it is buffered either with or without extra delay, as determined by the delivery protocol. Each received packet is transmitted at an allocated time slot according to the schedule on the node.

When the packets arrive at the destination node, the start-up discipline determines the time to start delivering the packets to the processing application, which will process the packets continuously at the rate of \( n \) packets per \( T \) time slots.
5.3 Scheduling algorithm at each intermediate node

As explained in the previous section, the scheduling algorithm presented in this section is executed by each intermediate node during the connection establishment stage. The new message request \( M = (\text{avgD}, d) = (n, d) \) is the input to the scheduling algorithm.

We address the scheduling problem in a system that allows the message streams to dynamically arrive at and depart from the system. When the request of a new message stream arrives at a certain node, some time slots in the template of this node are already allocated to existing message streams. We adopt the assumption explained in Section 4.1 that the template size is a fixed system parameter \( T \). Moreover, when the new message stream is scheduled in the template, the allocation patterns of the already scheduled streams are not influenced. Therefore, once a message stream is accepted and allocated in the template at a node, its schedule will not be changed until the stream terminates.

Let \( \{\text{location}(1), \ldots, \text{location}(n)\} \) be the set of time slots allocated by the scheduling algorithm to the \( n \) instances of \( M \) in the template. Because of the repetition of the template in the schedule, the location of the \((j + n)\)th instance is \( T \) time slots away from the \( j\)th instance, that is, \( \text{location}(j + n) = \text{location}(j) + T \).

Given the schedule \( \{\text{location}(1), \ldots, \text{location}(n)\} \), the set of temporal distances between all neighboring instances in the template is \( D = \{D_1, D_2, \ldots, D_n\} \), where \( D_j = \text{location}(j + 1) - \text{location}(j) \) for all \( j \in [1, n] \). For example, assume that in a template of size 6, the schedule of a stream consisting of three time slots is \( \{1, 2, 5\} \). Then \( D = \{1, 3, 2\} \). In this example, the packets of \( M \) will be transmitted at time slot \( \{1, 2, 5, 7, 8, 11, 13\ldots\} \) according to the schedule, and the corresponding distances between every two adjacent instances are \( \{1, 3, 2, 1, 3, 2\ldots\} \). The sum of all the \( D_j \) is equal to \( T \), as shown in the following equation.

\[
\sum_{j=1}^{n} D_j = \sum_{j=1}^{n} (\text{location}(j + 1) - \text{location}(j)) = \text{location}(n + 1) - \text{location}(1) = T
\]  

(5.1)

Recall that the scheduling jitter is defined as the variance of the temporal distances between all adjacent instances of the stream. Let a random variable \( X \) be the distance between any two adjacent instances. As shown in the previous example, the pattern of distance values in \( D \) will be repeated cyclically, which implies that the probability of \( X = D_j \) is equal to \( 1/n \) for all \( j \in [1, n] \). So the expected value of \( X \) is \( E[X] = \sum_{j=1}^{n} \frac{1}{n}D_j = T/n = \text{avgD} \). Given a schedule, the scheduling jitter is calculated by the following equation, where \( \text{Var}[X] \) stands for the variance of \( X \).

\[
\text{Var}[X] = E[(X - E[X])^2] = \sum_{j=1}^{n} \frac{1}{n}(D_j - \text{avgD})^2 = \frac{\sum_{j=1}^{n} (D_j - \text{avgD})^2}{n}
\]  

(5.2)
The scheduling problem at each intermediate node is formally stated as follows.

**Problem statement**: Given a template of size $T$ and a set of $m$ vacant time slots $\{S_1, S_2, \ldots, S_m\}$ in the template, the problem is to allocate $n$ vacant time slots to the new message stream $M$, such that the scheduling jitter is minimized.

In Section 4.2.3, an algorithm *Dynamic optimization* is proposed to minimize scheduling jitter by transforming the original scheduling problem to a graph problem, and using a dynamic programming technique to find the shortest path in the graph. The same idea can be applied here. However, since there is an explicit specification of maximum distance constraint in the problem of Section 4.2.3, the algorithm presented previously makes use of $\max D$ to limit the search space, which can be seen from the time complexity of *Dynamic optimization* that depends on $\max D$. In this section, we present a variation of the problem transformation technique, though the basic policy is still the same.

We transformed the scheduling problem to an equivalent graph problem $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{G}$ is a directed complete graph with $m$ vertices, $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. Each vertex $V_j$ represents the $j^{th}$ vacant time slot in the template, $S_j$. We define the weight of $V_j$ as $w(V_j) = S_j$. A directed edge $E_{j,k}$ points from vertex $V_j$ to $V_k$. We explain the meaning of $E_{j,k}$ and define its weight $w(E_{j,k})$ as follows.

1. When $j < k$, $w(E_{j,k}) = (w(V_k) - w(V_j) - \text{avg}D)^2$, which means that if vacant slots $S_j$ and $S_k$ are chosen for two consecutive instances of a message stream, then distance between these two adjacent instances, $S_k - S_j$, contributes a factor of $(S_k - S_j - \text{avg}D)^2$ to the calculation of the scheduling jitter.

2. When $j > k$, $w(E_{j,k}) = (w(V_k) + T - w(V_j) - \text{avg}D)^2$, which means that if vacant slots $S_j$ and $S_k + T$ are chosen for two consecutive instances of a message stream, then distance between these two adjacent instances, $S_k + T - S_j$, contributes a factor of $(S_k + T - S_j - \text{avg}D)^2$ to the calculation of the scheduling jitter.

3. If $j = k$, $w(E_{j,k}) = \infty$, indicating that the edge is meaningless.

![Figure 5.1: An example of the problem transformation.](image)

**Example 5.1**: A template of size 6 has $m = 4$ vacant time slots $\{1, 2, 3, 5\}$. A new message stream requests to allocate $n = 3$ instances in the template. So $\text{avg}D = \frac{6}{3} = 2$. 
The graph transformation for Example 5.1 is illustrated in Figure 5.1. In the figure, the shaded slots indicate the ones occupied by previously allocated streams. A directed complete graph of 4 vertices is shown beside the timeline, where $V_1, V_2, V_3, V_4$ corresponds to the vacant time slots 1, 2, 3, 5, respectively. The meaning of the edge $E_{2,3}$ and $E_{3,2}$ is illustrated in the figure. The calculation results of their weights are $w(E_{2,3}) = 1$ and $w(E_{3,2}) = 9$. For clarity, other edges of the graph are not shown under the template.

A schedule $SCED = \{\text{location}(1), ..., \text{location}(n)\}$ can be represented by a cycle of $n$ vertices in the graph where each vertex represents an allocated slot $\text{location}(j)$. The weight of the cycle is calculated as the summation of the weights of all the edges in the cycle. According to Equation (5.2), the weight of the cycle is equal to $n$ times the scheduling jitter of $SCED$. Thus, a schedule of $n$ instances with the minimum jitter corresponds to a cycle of $n$ vertices with the smallest weight. We call a cycle with the smallest weight to be a shortest cycle. In Figure 5.1, the bold edges constitute a cycle. Since the weight of each edge in this cycle is equal to 0, this is the shortest cycle with 3 vertices. Thus, we obtain an optimal schedule $\{1, 3, 5\}$.

We solve this shortest cycle problem by using a dynamic programming technique [21]. Note that a shortest cycle that contains vertex $\text{location}(i)$ is equivalent to a shortest path from $\text{location}(i)$ back to itself. Let us define a $m \times m$ matrix $\Pi^{(k)}$, where each element $\pi_{i,j}^{(k)}$ is the weight of the shortest path from $V_i$ to $V_j$ with exactly $k$ edges. In order to find the shortest cycle with $n$ edges, we construct a series of matrices $\Pi^{(0)}, \ldots, \Pi^{(n)}$. The smallest element among the diagonal elements $\pi_{i,i}^{(n)}$ in the matrix $\Pi^{(n)}$ gives the weight of the shortest cycle. Since a path from $V_i$ to $V_j$ with $k$ edges can be constructed by a path from $V_i$ to $V_q$ with $k-1$ edges plus the edge $E_{q,j}$, each element in the matrix is calculated by the following recursive relation.

$$
\pi_{i,j}^{(k)} = \begin{cases} 
    w(E_{i,j}) & \text{if } k = 1 \\
    \text{MIN}_{1 \leq q \leq m} \{\pi_{i,q}^{(k-1)} + w(E_{q,j})\} & \text{if } 1 < k \leq n
\end{cases}
$$

Note that in the matrix $\Pi^{(n)}$, we only need to calculate $\pi_{i,i}^{(n)}$ for all $1 \leq i \leq m$. In this algorithm, the matrices only record the weight of the shortest path. The path itself can be traced backward via a series of assistant matrices that record the predecessor vertex of the last edge on the path. The solution of Example 5.1 by applying the dynamic programming technique is shown below. In the last matrix, the shortest cycle has a weight 0, and consists of vertex $V_1$, $V_3$ and $V_4$, which correspond to slot 1, 3, 5, respectively. Thus, a uniform allocation pattern with jitter 0 is found for this example.

$$
\Pi^{(1)} = \begin{bmatrix} 
\infty & 1 & 0 & 4 \\
9 & \infty & 1 & 1 \\
4 & 9 & \infty & 0 \\
0 & 1 & 4 & \infty
\end{bmatrix}, \Pi^{(2)} = \begin{bmatrix} 
4 & 4 & 2 & 0 \\
1 & 2 & 5 & 1 \\
0 & 1 & 4 & 8 \\
8 & 1 & 0 & 2
\end{bmatrix}, \Pi^{(3)} = \begin{bmatrix} 
0 & \ldots & \ldots \\
\ldots & 2 & \ldots \\
\ldots & \ldots & 0 \\
\ldots & \ldots & \ldots & 0
\end{bmatrix}
$$

(5.4)
In summary, the algorithm \textit{min-jitter} selects \( n \) out of the \( m \) vacant time slots in the template to minimize the jitter. The algorithm is described in Figure 5.2.

\textit{min-jitter}

1. If \( m < n \), the new stream is rejected because the rate requirement cannot be satisfied.

2. If \( n = 1 \), allocate the first vacant time slot in the template to the new stream. If \( n = m \), allocate all the vacant time slots to the new stream.

3. Solve the transformed shortest cycle problem. The schedule is constructed from the set of the time slots that correspond to the vertices in the shortest cycle.

Figure 5.2: The \textit{min-jitter} Algorithm

Since the algorithm needs to calculate \( n \) matrixes of size \( m \times m \), and each matrix element takes at most \( m \) time to compute, the time complexity of this algorithm is \( \theta(m^3 n) \), where \( m \) is the number of vacant slots and \( n \) is the number of instances need to be allocated in the template.

A heuristic algorithm similar to \textbf{Dynamic heuristic} presented in Section 4.2.2 can also be used to minimize the scheduling jitter. Since the message model of this chapter does not have distance constraints, the way to calculate the scheduling window can be simplified. The \textit{LEFT} boundary of the scheduling window for the instance \( M_j \) is just \( \text{location}(j - 1) + 1 \). The \textit{RIGHT} boundary of the scheduling window needs to take care of the average rate requirement. Assume that message stream \( M \) requires to allocate \( n \) slots, and there are \( m \) vacant slots \( \{S_1, \ldots, S_m\} \) in the template. When the current instance is \( M_j \), we still have \( n - j \) instances need to be scheduled in the remaining part of the template. Thus, in order to satisfy the requirement of \( n \) instances in the template, the instance \( M_j \) cannot be scheduled at a vacant slot that is later than \( S_{m-(n-j)} \), which is the \textit{RIGHT} boundary of the scheduling window. As in \textbf{Dynamic heuristic}, we calculate a scheduling window for each instance of the new message stream, compute the ideal position based on the average distance of the remaining instances, and select the vacant slot from the scheduling window that is the closest to the ideal position.

The heuristic algorithm can also fully utilize the resource, as the algorithm \textit{min-jitter} does. That is, both algorithms accept a new stream \( M \) if and only if there are at least \( n \) vacant slots in the template. Thus, the two algorithms have the same performance in terms of system-wide acceptance ratio. Simulation results show that the scheduling jitter generated by the heuristic algorithm is larger than \textit{min-jitter}, but very close to the optimal one generated by \textit{min-jitter}. The performance comparison follows a similar trend to Figure 4.10(a) and Figure 4.10(b). The main advantage of the heuristic algorithm is the much smaller time complexity compared to that of \textit{min-jitter}, especially
when the workload is low. One extreme example is when all the time slots in the template are vacant. Assume that \( T = 12 \), and the first new stream requires \( n = 4 \) slots in the template. The heuristic algorithm only takes a time complexity of \( \Theta(1) \) to schedule each instance, and return the uniform schedule \( \{1, 4, 7, 10\} \) after \( \Theta(n) \) time. However, the optimal solution \textit{min-jitter} needs to calculate 4 matrices of size \( 12 \times 12 \) in this case.

One possible solution that takes advantage of both algorithms is to set a threshold for system utilization. When the utilization is smaller than the threshold, the heuristic algorithm is applied to schedule the new streams, since the advantage of the time complexity is more significant than that of scheduling jitter. Conversely, when the utilization is larger than the threshold and the number of vacant slots is small, the optimal algorithm is applied.

### 5.4 Delivery protocols

The scheduling algorithm in the previous section provides an allocation pattern for message stream \( M \) that satisfies its rate requirement and minimizes the scheduling jitter. Assume that the message stream is accepted, and a schedule is generated by each intermediate node during the connection establishment stage. The packets of the stream are delivered at each node according to one of three delivery protocols, NED, WED and Low-delay-NED. In each protocol, we will describe the connection establishment scheme, the buffering scheme at the intermediate nodes and the start-up scheme at the destination. Moreover, a bound of the largest ETE delay in the worst case is derived for each scheme. In later sections, we will use the following example to explain the three protocols. The result of applying the three schemes to this example is shown in Figure 5.3.

**Example 5.2:** A message stream \( M \) requires transmission at the rate of 4 packets in a template of size 12, from node \( N_0 \) to node \( N_3 \). In Figure 5.3, the time line at each intermediate node shows the allocation pattern generated by the scheduling algorithm at the node, where a time slot with a ‘X’ stands for slot allocated to \( M \). So the allocation pattern of node \( N_1 \) is \( \{1, 3, 6, 9\} \), and the allocation pattern of node \( N_2 \) is \( \{1, 2, 6, 10\} \). The down-arrows on the link \( L_{k,k+1} \) represent transmissions of packets on the link between node \( N_k \) and node \( N_{k+1} \), which indicates the output pattern of \( N_k \) and the input pattern of \( N_{k+1} \). The first row of down-arrows on link \( L_{0,1} \) indicates the regular output pattern from the source node \( N_0 \). The last row of down-arrows in each sub-figure shows the regular delivery of packets from the destination node to the processing application. A dotted line connects an arriving packet to the time slot at which it is transmitted according to a certain buffering discipline.
Figure 5.3: An ETE example of 3 hops, $T = 12$, $n = 3$, $avgD = 4$
5.4.1 Delivery with No Extra Delay at intermediate nodes (NED)

The buffering scheme of NED is straightforward in the sense that the packet arriving at each intermediate node is sent out as soon as possible in the next available allocated time slot. However, the existence of scheduling jitter in the input and output pattern may lead to the skipping of allocated slots, as shown in Figure 5.3(a) that applies NED to Example 5.2. In the figure, a circled time slot indicates a skip. At node $N_1$, though slot 3 is allocated for the stream, it is skipped because the receiving buffer is empty at that moment.

When the packets arrive at $N_2$, more skips take place, and the predictable allocation pattern generated by the scheduling algorithm will be destroyed along the route to the destination. However, we will show that this slot skipping will not occur after a certain time as long as the source node $N_0$ is continuously transmitting packets at the required rate. At the first node $N_1$, skipping slots implies that the instantaneous transmission rate of $N_1$ is lower than the receiving rate from $N_0$. So the packets will be accumulated in the receiving buffer at $N_1$, until the transmission rate is equal to the receiving rate. This means that, from this time on, there will be no more skipping. We define the no-skip state of a node to be the state at which the node will not skip any allocated time slot until the termination of the stream. Once $N_1$ reaches the no-skip state, it will keep on sending $n$ packets in every $T$ slots to $N_2$. The above argument implies that $N_2$ will also reach the no-skip state, and so on. The time to reach the no-skip state at node $N_k$ is defined as the time slot immediately after the last skipped slot, and is denoted as $n_{sk}$.

In Figure 5.3(a), there is no slot skipping after slot 3 at node $N_1$, nor after slot 14 at $N_2$. So $n_{s1} = 4$ and $n_{s2} = 15$. We prove the following theorem about the no-skip state and $n_{sk}$.

**Theorem 1:** When the buffering scheme of NED is applied, each node can reach the no-skip state. Moreover, $n_{sk} \leq n_{sk-1} + T - 1$.

**Proof:** Theorem 1 can be proved by induction. We begin with the correctness of the first intermediate node $N_1$.

Assume that the source node $N_0$ starts from time $t_0$ to continuously send packets at the required rate. Then $n_{s0} = t_0$. The receiving buffer at $N_1$ has 0 packets before time $n_{s0}$. Note that $N_1$ has an input pattern of receiving $n$ packets in every $T$ time slots, and an allocation pattern that schedules $n$ slots in every $T$ time slots.

If no skip occurs at $N_1$ in the interval $[n_{s0}, n_{s0} + T - 1]$, then obviously no skip will occur in the future, because both the input pattern and the slot allocation pattern in the interval are repeated. So $N_1$ has reached the no-skip state at time $n_{s0}$.

Assume that $N_1$ skipped $b(b > 0)$ time slots in the interval $[n_{s0}, n_{s0} + T - 1]$. Thus, there are $b$ packets accumulated in its receiving buffer before time $n_{s0} + T$. Both the input pattern and the allocation pattern in the interval $[n_{s0}, n_{s0} + T - 1]$ are repeated in the next interval of
Assume that in this interval, node $N_1$ still skips a slot at time $t_s$, $ns_0 + T \leq t_s \leq ns_0 + 2T - 1$. Skipping slot $t_s$ implies that the receiving buffer of $N_1$ is empty at time $t_s$, that is, $N_1$ has finished transmitting all the received packets before $t_s$. Moreover, the next packet arrives at a time later than $t_s$. Let $c$ be the number of packets that $N_1$ has received in the time interval $[ns_0 + T, t_s]$. Because the first $b$ allocated slots in this interval are used to transmit the $b$ leftover packets in the buffer, there must be at least $b + c$ time slots allocated in the interval $[ns_0 + T, t_s - 1]$ to make the receiving buffer empty before $t_s$. Thus, in the previous interval $[ns_0, t_s - T - 1]$, $N_1$ also has received $c$ packets from $N_0$, and at least $b + c$ slots are allocated in this interval. Because there is 0 accumulation in the receiving buffer before $ns_0$, there should be at least $b$ skipped slots before slot $t_s - T$. It has to be exactly $b$ because we assumed that only $b$ slots are skipped in the first template. Then the buffer is empty before $t_s - T$. Since the next packet arrives at $N_1$ at a time later than $t_s - T$, slot $t_s - T$ should also be skipped in addition to the previous $b$ skipped slots, which violates the assumption. So no skip can occur after time $ns_0 + T$. $N_1$ reaches the no-skip state at a time earlier than or equal to $ns_0 + T - 1$. Figure 5.4 illustrates the proof of correctness for $N_1$.

![Figure 5.4: Illustration of the proof of correctness at $N_1$](image)

We have proved that the theorem is correct for $N_1$. Assume that it is correct for $N_k$, it is easy to prove that it is also correct for $N_{k+1}$ by following the same reasoning. □

In Figure 5.3(a), $ns_2 = ns_1 + 11 = ns_1 + T - 1$, that is, the time to reach the no-skip state at $N_2$ is exactly $T - 1$ slots later than $ns_1$. So the bound in Theorem 1 is a tight bound for $ns$.

The no-skip state is important for the destination node to decide when to start to deliver the packets to the application. Before the last intermediate node $N_{h-1}$ reaches the no-skip state, the destination node $N_h$ receives packets with no predictable pattern. The receiving rate is lower than $n$ packets per $T$ slots because of the skipping. In order to guarantee the destination continuity requirement, $N_h$ cannot begin processing until it is sure that the receiving buffer will not underflow in the future (i.e., there is always a packet for it to process when it needs to).

Once the last intermediate node $N_{h-1}$ reaches the no-skip state, it transmits packets to the destination node $N_h$ according to the allocation pattern of $N_{h-1}$ which guarantees $n$ packets in every $T$ slots. So $N_h$ can detect that $N_{h-1}$ has reached the no-skip state when it receives $n$ packets within $T$ time slots. We call this the detecting discipline. It can be simply implemented by using
a FIFO queue of length \( n - 1 \) to store the receiving time of the last \( n - 1 \) packets, as described in Figure 5.5.

1. The arrival times of the first \( n - 1 \) packets are enqueued.

2. When \( N_h \) receives a new packet, it compares the receiving time with the one at the front of the queue.

3. If the difference is smaller than or equal to \( T \), then \( N_h \) has received \( n \) packets within \( T \) time slots. \( N_h \) immediately starts to deliver the packets to the processing application.

4. Otherwise, \( N_h \) dequeues the front element and enqueues the new receiving time at the end.

Figure 5.5: The Detecting discipline

Because \( N_{h-1} \) reaches the no-skip state at time \( ns_{h-1} \), the detecting discipline decides that \( N_h \) can start up the processing at time \( ns_{h-1} + T - 1 \). In Figure 5.3(a), \( N_3 \) can detect the no-skip state of \( N_2 \) by receiving 4 packets from time 18 to time 26. So a start-up time can be decided by the detecting discipline at slot 26, which is equal to \( ns_2 + 11 \). According to Theorem 1, it is guaranteed that \( N_{h-1} \) can reach the no-skip state at a time no later than \( t_0 + (h - 1)(T - 1) \), where \( t_0 \) is the time when \( N_0 \) starts the transmission of stream \( M \). Thus, the ETE delay upper bound provided by the detecting discipline is thus \( h(T - 1) \).

Before \( N_h \) detects the no-skip state, \( N_h \) has been receiving packets at some unpredictable rate. The accumulation in the receiving buffer can be used to allow for an earlier start-up. From the detecting discipline, we know that the destination node can safely start to process the packets from time \( t_0 + h(T - 1) \), which can be considered as the bound of the latest time for \( N_h \) to start-up. If, at a certain time, there is already enough packets in the buffer for \( N_h \) to process at the rate of one packet in every \( avgD \) time slots until the bound is approached, then the start-up decision can be made immediately. We name this start-up scheme the approaching discipline which is described in Figure 5.6. Note that the source node can transmit a packet containing \( t_0 \) before it starts to deliver the data packets to inform the destination node of \( t_0 \). The destination node will discard the first packet after reading the value of \( t_0 \) from it.

The ETE delay bound provided by the approaching discipline is \( (h - 1)(T - 1) \). In the example of Figure 5.3(a), \( N_1 \) starts to transmit the packets at time 1. So the estimated bound is \( 1 + 3 \times 11 = 34 \). By time slot 19, \( N_3 \) has received 5 packets from \( N_2 \). Since \( 19 + 3 \times 5 = 34 \), \( N_3 \) can immediately start to deliver the packets to the processing application at time 19, which is earlier than the start-up decision made by the detecting discipline at time 26.
1. Once $N_h$ receives the first packet, it sets a time-out value to be $t_0 + h(T - 1)$.

2. When a new packet arrives at $N_h$, increase a counter that records the number of packets accumulated in the buffer of $N_h$. This means that $N_h$ can start avgD time slots earlier, since it has one more packet to process before reaching the time-out value.

3. If the current time plus the time to process all the currently buffered packets is larger than or equal to the time-out value, $N_h$ immediately starts to deliver the packets to the application.

Figure 5.6: The Approaching discipline

We combine both the detecting discipline and the approaching discipline at $N_h$. So $N_h$ starts delivery to the processing application either because it has detected the no-skip state, or it has reached the timeout. Consider a situation where for each node $N_k$, $n_{sk} = n_{sk-1} + T - 1$. That is, the no-skip state is reached at the latest possible time. In this case, the detecting discipline is not efficient for $N_h$ to start up. However, a late skipping before $n_{sk}$ implies that the receiving buffer is empty at $N_k$, and all the received packets have been transmitted toward the destination before the skipping. So $N_h$ should have a fast accumulation in its receiving buffer, and the approaching discipline can decide an earlier start-up time. On the other hand, a slow accumulation at $N_h$ implies that the intermediate nodes skips at the beginning of the template, which results in an early $n_s$. In this case, the detecting discipline is more efficient than the approaching one. So these two disciplines complement each other.

In summary, the NED protocol is abstracted as follows.

1. At the connection establishment stage, each intermediate node finds the schedule according to min-jitter.

2. When the message stream starts to be transmitted, each intermediate node sends the packets at the earliest possible slot based on the schedule. No extra buffering delay is introduced.

3. The destination node applies the detecting and the approaching disciplines to make the start-up decision.

The ETE delay bound of the NED protocol scheme is $h(T - 1)$, which is provided by both the detecting discipline and the approaching discipline. However, scheduling jitter at the intermediate nodes largely influences the ETE delay. A small scheduling jitter leads to a small probability of skipping in the intermediate nodes, thus to a small start-up delay at the destination node. For example, when the scheduling jitter is 0, that is, the instances of the stream are uniformly allocated in
the templates of all the intermediate nodes, then slot skipping does not occur. Since our scheduling algorithm minimizes the scheduling jitter, the simulation results in section 5.5.3 will show that by applying the min-jitter algorithm and the NED protocol, the ETE delay is typically much smaller than the worst case bound.

5.4.2 Delivery With Extra Delay at intermediate node (WED)

As presented in the previous section, protocol NED transmits each packet at the earliest possible slot, such that the buffering delay at each intermediate node is minimized. However, slot skipping at the intermediate nodes makes it hard for the destination node to predict the arriving pattern. In this section, the objective of scheme WED is to get rid of skipping by introducing extra delay before the transmission of the first packet at each intermediate node. Since by doing this, the input/output patterns at all the nodes are predictable, the destination node can start to deliver packets to the application as early as possible.

For a stream $z$, we define a local delay pair at node $N_k$ as $(S, y)$, which means that if the first packet of the stream arrives at $N_k$ at time slot $S$ ($S \in [1, T]$), then it needs to be delayed at least $y$ time slots to prevent slot skipping in the future at $N_k$. Any delay value that is smaller than $y$ time slots will cause future skipping.

Since it is not certain when the first packet will arrive at node $N_k$, we need to construct a set of $n$ local delay pairs for $M$ at each node $N_k$, in which every element corresponds to a possible arrival time for the first packet at $N_k$. We use $location^k(j)$ to denote the time slot allocated to the $j^{th}$ instance of $M$ in the template of node $N_k$. The set $\{location^{k-1}(1), ..., location^{k-1}(n)\}$ contains all the possible time slots at which $N_{k-1}$ may deliver the first packet to node $N_k$ in the interval $[1, T]$. If the first packet arrives at $N_k$ from $N_{k-1}$ at $location^{k-1}(j)$, and is transmitted by $N_k$ at $location^k(i)$ such that no future skipping will occur at $N_k$, then one local delay pair at $N_k$ is $(x_j, y_j)$ where $x_j = location^{k-1}(j)$ and $y_j = location^k(i) - location^{k-1}(j)$. The set of local delay pairs at node $N_k$ is denoted as $P_k = \{(x_j, y_j), j \in [1, n]\}$, which can be found from the Construct-pairs algorithm shown in Figure 5.7.

The method first finds the output slots corresponding to the $n$ arrivals at node $N_k$ as if NED is applied. If the $n^{th}$ packet that is received by $N_k$ at $location^{k-1}(n)$ will be transmitted by NED at $location^k(g)$, then there must be $g - n$ skipped slots at $N_k$ in the interval $[1, location^k(g)]$. So instead of delivering the first packet immediately at time $z_1$, and skipping allocated slots later, WED skips the first $g - n$ allocated slots at the beginning, and delays the transmission of the first packet until the location of the $(g - n + 1)^{th}$ instance, such that packets $M_1, ..., M_n$ are transmitted by $N_k$ at continuously allocated slots $location^k(g - n + 1), ..., location^k(g)$.
1. Assume that \( n \) packets arrive at \( N_k \) in the slots \( \{\text{location}^{k-1}(1), \ldots, \text{location}^{k-1}(n)\} \). Find \( \{z_1, \ldots, z_n\} \), the set of time slots for \( N_k \) to transmit these \( n \) packets if each packet is transmitted immediately in the next available allocated slot at \( N_k \), as would be done in NED scheme.

2. Assume that \( z_n = \text{location}^k(g) \), which means that the \( n^{th} \) received packet is transmitted by \( N_k \) in NED scheme at the location of the \( g^{th} \) instance. Then \( \{\text{location}^k(g - n + 1), \ldots, \text{location}^k(g)\} \) is the set of time slots at which each packet will be transmitted by \( N_k \) without slot skipping. That is, the \( j^{th} \) packet received at \( \text{location}^{k-1}(j) \) will be transmitted at \( \text{location}^k(g - n + j) \) by \( N_k \) to avoid skipping, for \( j \in [1, n] \).

3. Construct the set \( \mathcal{P}_k \) with pairs \((x_j, y_j)\) for all \( j \in [1, n] \), where \( x_j = \text{location}^{k-1}(j) \) and \( y_j = \text{location}^k(g - n + j) - \text{location}^{k-1}(j) \).

Figure 5.7: The \texttt{Construct-pairs} algorithm

By applying this method to Example 5.2, Figure 5.3(a) shows that if \( N_1 \) receives 4 packets at slots \( \{1, 4, 7, 10\} \), then by applying NED, the \( 4^{th} \) input corresponds to the \( 5^{th} \) allocated slot at \( N_1 \), which indicates that 1 skipping has occurred. So the first packet needs to be delayed until the 2nd instance to avoid the skipping, as shown in Fig 5.3(b) where \( M_1 \) is delivered by \( N_1 \) at the second allocated slot. In this case, the set of local delay pairs is \( \{(1, 2), (4, 2), (7, 2), (10, 3)\} \).

Lemma 1 Assume that there is no skipping at node \( N_{k-1} \), and the local delay pair \((x_j, y_j)\) is derived from the algorithm \texttt{Construct-pairs}. If the first packet of \( M \) is transmitted from \( N_{k-1} \) to \( N_k \) at time \( x_j = \text{location}^{k-1}(j) \), then \( y_j \) is the shortest time that \( M_1 \) needs to be buffered at \( N_k \) to avoid skipping.

Proof: First, it is obvious from the description of \texttt{Construct-pairs} that delaying \( M_1 \) which arrives at time \( x_1 \) by \( y_1 \) slots can avoid skipping at \( N_k \). The algorithm shows that if \( M_1 \) is received by \( N_k \) at slot \( x_1 \), then \( M_j \), which arrives at time \( x_j \), needs to be delayed by \( y_j \) time slots. Now we need to discuss the case where the first packet \( M_1 \) arrives at time \( x_j \), instead of \( x_1 \). We will prove that this does not introduce skipping if \( M_1 \) is delayed by the same amount of time, \( y_j \).

Algorithm \texttt{Construct-pairs} shows that \( y_j \) is the shortest time that an input at time \( x_1 \) needs to wait to avoid future skipping at node \( N_k \). Since we assumed that there is no skipping at node \( N_{k-1} \), the input pattern of node \( N_k \) is repeated in every time interval of size \( T \). Moreover, the allocation pattern of \( N_k \) is also repeated, so an input at time \( x_1 + T \) cannot be delayed by less than
y₁ slots to avoid future skipping. Now the first packet of the stream, M₁, is received by N_k at time $x_j$. If M₁ is delayed for less than $y_j$, then the packet that is received at time $x₁ + T$ must also be delayed for less than $y₁$, in order to prevent any skipping in the interval $[x_j, x₁ + T]$. However, this will lead to some skipping after time $x₁ + T$. Thus, $y_j$ is the shortest time that $M₁$ needs to be delayed to avoid skipping any allocated slot at $N_k$ if $M₁$ arrives at $N_k$ at time $x_j$. □

If the first packet of message stream M arrives at node $N_k$ at a time that is later than $T$, we can easily decide how long it needs to be delayed at $N_k$. Assume that the first packet $M₁$ is received by node $N_k$ at time $t$. We define a function $\overline{\text{mod}}(t)$ that returns the modulus of the division of $t$ by $T$ if it is not 0. Otherwise, $\overline{\text{mod}}$ returns $T$. Let $t' = \overline{\text{mod}}(t)$. So $t' \in [1, T]$, and $t - t' = k \times T$ for some integer $k$. Because of the template repetition, an arrival at $t$ is equivalent to an arrival at $t'$. Hence, when a packet is received at time $t$ on $N_k$, find a local delay pair $(x_j, y_j)$ in the set $P_k$ such that $x_j = t' = \overline{\text{mod}}(t)$. Then $y_j$ is the shortest amount of time that $M₁$ needs to be delayed at node $N_k$ to avoid slot skipping.

Note that in WED, each intermediate node needs to guarantee that no allocated slot is skipped during transmission. This is similar to the destination continuity requirement, in the sense that both need to continuously deliver the packets in a certain pattern once the delivery starts. So the destination node can also use the set of local delay pairs to determine the start-up time. When the destination node receives the first packet of the stream at time $t$, it can decide the start-up time according to $\overline{\text{mod}}(t)$ and the set of delay pairs at $N_k$, in the same way as described for the intermediate node.

The set of local delay pairs provides a way to calculate the exact ETE delay that is needed to achieve the destination continuity requirement. It relies on the fact that all the packets of the stream have the same ETE delay, because the packets are delivered regularly at the rate of 1 packet every $avgD$ slots at both the source node and the destination node. So we only need to trace the delay of one packet, as described in the algorithm exact-ETE-delay in Figure 5.8. The ETE delay of a packet is calculated by using an accumulated delay pair $(z, u)$ at each node $N_k$, which means that the packet that arrives at $N_k$ at time $z$ has experienced an accumulated delay of $u$ time slots from the source node to node $N_k$.

In the exact-ETE-delay algorithm, $N₁$ initiates the accumulated delay pair according to a randomly picked local delay pair. On every subsequent node $N_k$, a received pair $< z, u >$ indicates that the packet which is transmitted by $N_{k-1}$ and received by $N_k$ at time $z$ has experienced a delay of $u$ slots when it reaches $N_k$. So $N_k$ finds the local delay value $y_j$ for this packet, which implies that the packet will be transmitted by $N_k$ at time $z + y_j$, and the accumulated delay value for this packet is increased to $u + y_j$ at $N_k$.

In summary, the following steps abstract the behavior of the protocol WED.
1. At the first node $N_1$, assume that a local delay pair $(x, y)$ is in $\mathcal{P}_1$, indicating that an input at $x$ is delayed at $N_1$ by $y$ slots and is transmitted at time $z = x + y$. Then form an accumulated delay pair $(z, y)$.

2. The accumulated delay pair is transmitted to the next node.

3. When node $N_k$ ($1 < k \leq h$) receives the accumulated delay pair $(z, u)$, it finds a local delay pair $(x_j, y_j)$ in $\mathcal{P}_k$ such that $x_j = \text{mod}(z)$. The accumulated delay pair is updated to $(z + y_j, u + y_j)$, and transmitted to the next intermediate node.

4. After executing the previous step at the destination node, the exact ETE delay is equal to $u$ in the final result of $(z, u)$.

Figure 5.8: The exact-ETE-delay algorithm

1. At the connection establishment stage, each intermediate node $N_k$ needs to execute the following steps. The destination node only executes step (b) and (c).

   (a) Find the allocation pattern $\{\text{location}^k(1), \ldots, \text{location}^k(n)\}$ based on the scheduling algorithm min-jitter.

   (b) Construct the set of local delay pairs $\mathcal{P}_k$ based on the allocation pattern of the current node and the previous node.

   (c) Calculate the new accumulated delay pair based on the local delay pairs and the accumulated delay pair received from the previous node.

   (d) Transmit the local allocation pattern and the accumulated delay pair to the next node.

2. Upon receiving the first packet of the stream, each node, including the destination node, decides how long it needs to be delayed according to the set of local delay pairs.

3. Once node $N_k$ starts to deliver $M_1$, it will deliver the subsequent packets at the next allocated slots.

Figure 5.3(b) shows the result of applying WED to Example 5.2. Since the packets are delayed by the exact amount of slots to achieve continuity at the destination, the ETE delay is shorter than the one shown in Figure 5.3(a), where the worst-case estimate is used.

In WED, the ETE delay bound is determined by the scheduling jitter. The effect is illustrated in Figure 5.9. Figure 5.9(a) shows the best case where the scheduling jitter is 0 and the instances of the stream are distributed uniformly in the template at each node. So the largest
delay a packet can experience at one node is $avgD - 1$. The ETE delay bound in this case is $(h - 1)(avgD - 1)$. Figure 5.9(b) shows the worst case where the scheduling jitter is the largest and the packets of the stream are all clustered together in the template at each node. So the largest delay a packet can experience at one node is $T - n$. The ETE delay bound in this case is $(h - 1)(T - n)$.

![Diagram of template of size T with avgD-1](a) input and output pattern of $N_k$ with 0 jitter

![Diagram of template of size T with T-n](b) input and output pattern of $N_k$ with largest jitter

Figure 5.9: the largest delay at $N_k$ in the best and worst case

### 5.4.3 Low-Delay-NED

In the previous sections, we have shown that the protocol NED has a simpler delivery scheme at the intermediate nodes than that of WED, but a longer ETE delay. This result will be confirmed by simulations in Section 5.5. In this section, we propose an Low-Delay-NED protocol that combines the simple buffering scheme of NED with the low ETE delay of WED.

Let $ned_{j}^{k}$ and $wed_{j}^{k}$ be the time slots to transmit the packet $M_{j}$ at node $N_{k}$ when NED and WED are applied, respectively. Theorem 2 states that at an intermediate node $N_{k}$, NED and WED will generate equivalent output patterns after a certain time. Recall that $ns_{k}$ is the time when an intermediate node $N_{k}$ reaches the no-skip state if protocol NED is applied.

**Theorem 2** For all $j$ such that $ned_{j}^{k} \geq ns_{k}$, it is true that $ned_{j}^{k} = wed_{j}^{k}$ at each node $N_{k}$.

Theorem 2 can be proved based on Theorem 1 and Lemma 1 by induction on the number of intermediate nodes, similar to the proof of Theorem 1. The intuition is that although a node transmits the first packet of a stream as early as possible in the NED protocol, some allocated slots will be skipped later, while WED avoids the future skip occurrences by skipping the same amount of allocated slots at the beginning, before the delivery of the first packet. For instance, in the example
shown in Figures 5.3(a) and Figure 5.3(b), we have shown that \( n_{s1} = 4 \) and \( n_{s2} = 15 \) if NED is applied. At node \( N_1 \), the output pattern shown in Figure 5.3(a) is the same as that in Figure 5.3(b) after time 4. It is also true at \( N_2 \) after time 15.

As introduced in Section 4.1, the start-up scheme of NED is based on the worst-case estimate, because the input pattern of \( N_h \) is unpredictable when there are slot skipping at intermediate nodes. However, Theorem 2 shows that after time \( n_{s_{h-1}} \), the output pattern of node \( N_{h-1} \) in NED will be the same as the one generated by WED. Thus, the input pattern of \( N_h \) can be predicted if we know the output pattern of \( N_{h-1} \) generated by WED protocol.

In order to obtain the predictable input pattern of the destination node in WED, the connection establishment scheme of Low-Delay-NED is the same as that of WED. That is, in addition to generating the allocation pattern via the scheduling algorithm, each node constructs the local delay pairs and calculates the exact ETE delay that is needed to achieve the destination continuity requirement.

After the connection is established, each intermediate node transmits the packet at the next available slot, as in the NED protocol. Because of Theorem 2, we do not need to delay the packets as in WED.

Since the buffering scheme of NED delivers the packets at the earliest possible time at each intermediate nodes, \( ned^{h-1}_j < wed^{h-1}_j \) before time \( n_{s_{h-1}} \). That is, before \( n_{s_{h-1}} \), a packet arrives at node \( N_h \) earlier than it would if WED were applied. According to Theorem 2, after time \( n_{s_{h-1}} \), the input pattern of \( N_h \) is the same as if the WED protocol were applied. So \( N_h \) can start to deliver the packets at the same time as in WED. The continuity will not be influenced by the early arrivals of the packets before \( n_{s_{h-1}} \). Because during the connection establishment, the destination node obtained the exact ETE delay that is needed to guarantee the continuity requirement, it can start to deliver the packets to the application at time \( t_0 + ETE_{delay} \), where \( t_0 \) is the time when the source node starts to transmit packets and \( ETE_{delay} \) is the exact ETE delay value \( u \) in the accumulated delay pair \( < z, u > \).

In summary, Low-Delay-NED adopts the same connection establishment scheme and destination start-up scheme as WED, and it uses the same buffering scheme as NED at the intermediate nodes. The ETE delay bound is \( (h - 1)(T - n) \), the same as that of the WED protocol.

Figure 5.3(c) applies Low-Delay-NED to Example 5.2. The intermediate nodes \( N_1 \) and \( N_2 \) delivers the packets as early as possible, as does the NED protocol in Figure 5.3(a). Since \( t_0 = 1 \) and the exact ETE delay value is 7, the destination node \( N_3 \) starts to process the packets at time 8, same as the WED scheme in Figure 5.3(b).
5.5 Performance evaluation

In this section, we compare the characteristics of the three delivery schemes, propose a general protocol for different types of applications according to the features of the three delivery schemes, and present simulation results.

5.5.1 Qualitative comparison of protocols

We presented three delivery protocols in the previous section. They are compared in terms of connection control, intermediate node processing and ETE delay as follows.

1. NED has the simplest connection establishment scheme. Both WED and Low-Delay-NED have more connection establishment overhead to find the local delay pairs, to transmit additional information, and need more space to store the additional information at each intermediate node during the connection establishment stage.

2. During the transmission stage, the delivery schemes of NED and Low-Delay-NED are straightforward. A general TDMA implementation in hardware [65] is sufficient to achieve the functionality. WED needs to deliberately delay the first packet according to the set of delay pairs stored at each node, and needs more space overhead to store the local delay pairs.

3. In terms of ETE delay, WED and Low-Delay-NED achieves the shortest ETE delay to satisfy the destination continuity requirement, subject to the allocation pattern on each intermediate node. NED has longer ETE delay because each intermediate node does not have any information about the other nodes on the route. This lack of system-wide information eliminates the predictability. Consequently, a worst-case estimation approach has to be applied.

In summary, the advantage of NED is the simplicity of implementation. WED has the best ETE delay. Low-Delay-NED simplifies the implementation of WED, while still keeping the optimal ETE delay. Therefore, applications should choose between NED and Low-Delay-NED, according to QoS requirements.

5.5.2 A general protocol

As described in Section 5.2, when a request for a new message stream \( M = (n, d) \) is submitted, the connection establishment scheme will accept the stream if both the transmission rate requirement \( n \) and the ETE deadline \( d \) can be satisfied. The transmission rate requirement can be guaranteed in the scheduling algorithm \( \text{min-jitter} \). We describe a general protocol to discuss the acceptance test of a stream by guaranteeing its ETE deadline \( d \), and the choice of the appropriate
scheme based on the characteristic of the applications. We assume that there are enough vacant slots at each intermediate node to satisfy the transmission rate requirement, such that the scheduling algorithm \textit{min-jitter} will not reject the stream.

1. A message stream with soft ETE delay constraints is always accepted. Protocol NED is selected, and \textit{min-jitter} is applied to obtain the allocation patterns. The simulation results show that the combination of NED with \textit{min-jitter} supplies a good ETE performance.

2. When the ETE deadline \(d\) is explicitly specified, and is larger than the worst case delay bound, \(h(T - 1)\), of NED, the new request is also accepted, and the combination of NED and \textit{min-jitter} is again selected to simplify the implementation and to guarantee the ETE deadline.

3. If the hard ETE deadline is smaller than \(h(T - 1)\), \textit{min-jitter} and Low-Delay-NED are applied. The exact ETE delay is obtained by the destination node during the connection establishment. If it is smaller than or equal to \(d\), the new request is accepted. Otherwise, the new request is rejected. So once a request is accepted, its hard ETE deadline can be guaranteed.

We apply \textit{min-jitter} for all types of applications, even the ones with very loose ETE delay constraints. The reason for using \textit{min-jitter} is that it not only supplies a short ETE delay for the current stream by achieving the minimal jitter, but also causes the vacant slots in the template to be distributed in a uniform manner, such that future streams with tight ETE delay constraints are able to achieve small scheduling jitter, and thus large probability of acceptance.

5.5.3 Simulation results

We have analyzed in the previous sections the worst case ETE delay bound for each protocol. We also explained that the ETE delay of all the protocols are largely influenced by scheduling jitter. In this section, simulation studies show the advantage of applying the scheduling algorithm \textit{min-jitter} to improve the ETE delay.

Since Low-Delay-NED has the same ETE delay as WED, we simulated the NED and WED protocols in a distributed system consisting of 20 nodes with dynamic message arrivals and departures. The parameter \(n\) of each message stream is a random number uniformly distributed in \([1, 50]\). At each node, random dynamic traffic are generated to achieve a certain average system workload. An ETE stream randomly selects \(h - 1\) nodes as the intermediate nodes, where \(h\) is uniformly distributed in the range \([5, 20]\). We apply the \textit{min-jitter} scheduling algorithm to generate the allocation pattern at each intermediate node. Both NED and WED are applied to deliver an ETE stream under the same system configuration, and the ETE delay of the stream is measured for each case. Moreover, in order to illustrate the impact of scheduling jitter over ETE delay, we
also simulated another two schedule algorithms, *FIFO-schedule* and *random-schedule*, which do not consider scheduling jitter. At each node, *FIFO-schedule* allocates the first *n* available time slots in the template to message stream *M*, and *random-schedule* randomly picks *n* vacant slots in the template. Thus, the 6 combinations of applying the three scheduling algorithms and two delivery protocols are simulated, under a system with an average workload of 10%, 30%, 50%, 70% and 90%, respectively. In each system configuration, ETE delay of 5,000 streams are measured.

Note that the parameters *h* and *avgD* unfairly affect the ETE delay, and the ETE delay values of two streams are not comparable if the parameters are different. We define relative ETE delay being equal to $\frac{ETE\_delay}{h \times avgD}$, which represents the average delay a packet experiences at each node, normalized to *avgD*. Figure 5.10 plots the relative ETE delays of all 5,000 streams when the average system workload is 70%, and each of the 6 combinations is applied. The results under other system workloads show similar trends, and are not shown here. When the same scheduling algorithm is applied, WED provides a shorter relative ETE delay than NED as expected. For each delivery protocol, *min-jitter* achieves the best performance among the three scheduling algorithms, because *min-jitter* tries to distribute the instances of a stream as uniformly as possible in the template at each node, which will benefit the ETE delay of the stream. Moreover, uniformly distributing the instances implies that the vacant slots in the template are also distributed in a uniform manner, which will benefit the future arriving streams to obtain a smaller scheduling jitter, and thus a smaller ETE delay. Therefore, the relative delay values of all tasks are distributed within a much smaller range by *min-jitter* as shown in Figure 5.10, indicating that the performance of all the streams are similarly good.

Figure 5.11(a) shows the average relative ETE delay of all the tasks versus system workload. The 6 curves represent the performance of the 6 combinations. Again, the figure shows that the performance of *min-jitter* is better than the other algorithms, even when protocol NED is applied. The ETE delay is slightly increased with the increase of the system workload, because it is harder to have a uniform allocation pattern when more time slots are occupied in the template. Figure 5.11(b) shows the standard deviation of the relative ETE delay for different system loads. It strengthens the conclusion that the combination of *min-jitter* with WED is the best, since it achieve the shortest ETE delay for most of the streams, with the smallest standard deviation.

From the figures, we can see that the combination of *min-jitter* and WED provides a relative ETE delay that is smaller than *avgD* for most of the streams, even under a system workload of 90%. Note that in the best situation as shown in Figure 5.9(a) where scheduling jitter is 0 at each node, the largest delay a packet can experience at one node is $avgD - 1$. This implies that our algorithm achieves an ETE delay performance that is close to the best case by minimizing scheduling jitter at the intermediate nodes.
5.6 Summary

In this chapter, we addressed the communication QoS problem in which ETE delay of a message stream in a distributed system needs to be considered, and at the same time, the destination continuity requirement needs to be satisfied. We solve the problem based on a TDMA slot allocation algorithm \textit{min-jitter}, which satisfies the average transmission rate requirement and minimizes the scheduling jitter. The effect of the scheduling jitter on the ETE delay is shown using simulation results. The scheduling algorithm \textit{min-jitter} substantially improves the ETE delay when compared to other algorithms in which scheduling jitter is not taken into consideration.

Three protocols are proposed, each of which includes a connection establishment scheme, a buffering scheme at the intermediate nodes, and a start-up scheme at the destination node. Protocol NED has the simplest implementation, and has the least implementation overhead among the three protocols. However, the acceptance test to guarantee the ETE deadline is conservative and uses a worst-case delay bound. Hence, NED is preferred in applications with soft ETE delay constraints, and applications with hard but loose ETE delay constraints. Both WED and Low-Delay-NED can obtain the exact ETE delay subject to the allocation patterns of the stream at each intermediate node. Thus, the acceptance test to guarantee the ETE deadline is accurate. Therefore, WED and Low-Delay-NED are suitable for applications with hard and tight ETE deadlines. Low-Delay-NED produces the same ETE delay as WED, but with a simpler buffering scheme.
Figure 5.10: relative ETE delays of 5,000 tasks, 70% system workload
Figure 5.11: Performance under different system workload
Chapter 6
Conclusion

Scheduling hard real-time tasks with stringent timing constraints is a general framework within which various timing constraints have been studied. A large body of literature has been developed to guarantee the timely execution of periodic real-time tasks, where instance invocation rate is considered the most important timing constraint. There is also research work derived for real-time tasks that require distance constraints, the relative timing constraints between two consecutive instances. An examination of the current state of the periodic task scheduling algorithms and the pinwheel scheduling algorithms reveals that the two fields reached a level of considerable maturity in the sense that efficient algorithms have been derived and various extensions have been studied. However, there is a lack of research on systems where both timing constraints exist. In practice, many real-time applications have both invocation rate requirements and distance constraints. We broaden the real-time research field by proposing a new task model which many real-time applications fall under.

Thus, the first contribution of this thesis is the proposed new task model that combines the timely requirements of both the periodic task model and the distance constraint task model. In the new model, a task has a rate requirement which represents the frequency of the task invocations, similar to the period requirement in the periodic model, and distance constraints, similar to MXDC (Maximum Distance Constraint) model. Moreover, we discuss a concept of distance constraint that is more general than that of the MXDC model because it takes both maximum and minimum distance constraints into consideration.

Although most of the research work in real-time assumes priority-driven systems, time-driven systems are crucial in some situations where run-time scheduling overhead cannot be accommodated. A review and comparison of these two basic scheduling paradigms shows that the time-driven paradigm is more suitable for hard real-time tasks with stringent timing constraints and strict overhead requirements. We adopt the template-based scheduling approach in this thesis, which is a practical implementation of the time-driven paradigm. We explore the application of the
template-based approach to find solutions for varied scheduling problems that are related to our new task model.

The second contribution of this thesis is the algorithms presented to schedule a set of static tasks both on a single resource and in a system with multiple resource constraints, such as a crossbar switch and a WDMA optical star coupler. Since the requirements of the static tasks are known beforehand, we generate an efficient schedule by minimizing the template size. A fixed point scheme is used to calculate the smallest size of a scheduling template, according to the rate requirements of the tasks. In each time slot of the template, the algorithms schedule a task instance of the highest priority which is based on the schedule window and the relaxability of the distance constraint of each task. The schedule window of each instance is calculated relative to the previously allocated instances, taking into consideration the distance constraints in both the forward direction and the backward direction. In the study of the crossbar switch and WDMA star coupler cases, input conflicts and output conflicts are taken into consideration in the scheduling algorithm, in addition to the rate requirements and distance constraint specifications of the message streams. Simulation performance results of the algorithms are presented and compared to the pinwheel algorithms and their derivatives which tie the maximum distance constraint to the average transmission rate of a task. The results show that higher schedulability is achieved when the rate requirements and distance constraint specifications of the real-time tasks are decoupled, especially when the system workload is high.

The third contribution of this thesis is the algorithms presented to solve the scheduling problem in a dynamic system where tasks dynamically arrive and depart. Three different scheduling algorithms are presented to allocate the vacant time slots in a template to the instances of a dynamic task by making greedy choices, non-greedy heuristic choices and optimization choices, respectively. The greedy algorithm can efficiently find a feasible schedule for the new task if there exists one. However, the greedy algorithm may generate large scheduling jitter, which can lead to rejecting the future arriving tasks, because of the clustered distribution pattern of the remaining vacant time slots. The optimization algorithm transforms the original scheduling problem to an equivalent graph problem, and minimizes scheduling jitter via finding the shortest path in the graph. Thus, the optimization algorithm has the smallest scheduling jitter and the largest system-wide acceptance ratio among the three algorithms. However, the time complexity of the optimization algorithm is costly, especially when the system is lightly loaded. The heuristic algorithm applies a scheduling policy similar to that used for the static tasks, and schedules the instances in the template within a scheduling window according to the ideal position. Although the heuristic algorithm is not locally optimal for the current arriving task, it combines the advantages of both the greedy algorithm and
the optimization algorithm in the sense that its time complexity is as efficient as that of the greedy algorithm, and it distributes the instances as uniform as possible, as the optimization algorithm does.

The fourth contribution of this thesis is the scheduling algorithms and delivery protocols presented in a distributed system for applications with destination continuity requirement. Adopting policies similar to the ones proposed for dynamic tasks in a single resource, the algorithms minimize scheduling jitter and satisfy the average transmission rate of a dynamically arriving stream at each intermediate node. Once the stream is successfully scheduled and accepted, the first delivery protocol NED decides that each intermediate node sends every packet of a message stream immediately in the next available time slot that is allocated to this stream according to the schedule. This protocol completely relies on the destination node to smooth the jitter before it starts to process the packets. In the second protocol WED, the intermediate nodes may delay the arriving packets for a certain time, even if there are available allocated time slots for the packet. The objective is to smooth the traffic on its route and reduce the start-up delay at the destination node. In the third protocol Low-Delay-NED, the intermediate nodes deliver packets with no extra delay, as in the NED scheme, but the destination node has an efficient start-up scheme, as in the Low-Delay-NED protocol.

In summary, the conclusions resulting from this research are twofold. First, the new task model is proposed to represent many real world applications, which broadens the research field of real-time applications. Moreover, high schedulability can be achieved by decoupling the rate requirement and distance constraint specification of a task. Second, we have shown that the template-based scheduling paradigm is an efficient way to guarantee stringent timeliness in real-time applications. In particular, this research work solves some practical problems that have not been extensively studied, such as how to schedule dynamic tasks with special timing constraints, and how to incorporate both minimum distance constraint and maximum distance constraint such that the scheduling jitter will be limited.
Chapter 7

Future work

We believe that the research work presented in this thesis is systematically integrated, in the sense that various extensions have been studied and efficient algorithms have been derived based on the timeliness requirement of a new task model. However, there is no limit to pursuing completeness in research areas. In this chapter we discuss several possible avenues to further extend and apply this research work in the future.

The first extension is to relax the assumption that the execution time of each task instance is a unit time, though this assumption is practical in many real-world systems, especially for the communication applications. An extended task model can include the parameter of a worst case execution time, say $C_i$, that is not always equal to 1. In this case, the model can be applicable to more general computation and control applications, and the communication systems can support message packets of variable sizes. Since the distance constraint specifies the temporal distance between the completion times of two consecutive instances, a feasible schedule needs to allocate $C_i$ slots to task $\tau_i$ within a time interval of $\max D_i$ slots starting from the finishing time of the previous instance. Because our proposed task model defines distance constraint between every two adjacent time slots, we can simply apply the derived algorithms to solve the general problem by guaranteeing that one time slot in every $\max D_i/C_i$ time slots is allocated to task $\tau_i$. However, this solution that provides more stringent timing constraints cannot be efficient in terms of schedulability. Further study is needed to derive an efficient solution for the extended task model.

The second direction in which the result of this dissertation could be extended is to apply our algorithms to a real-world application that requires a scheduling paradigm similar to the one considered in this proposal, such as the Bluetooth wireless system that allows users to make both voice and data connections between various wireless devices. The Bluetooth technology supports up to 7 ‘slave’ devices to be connected with a ‘master’ radio in one device. TDMA is applied to allocate time slots to the master and slaves. Currently, a simple round robin algorithm is applied to schedule the time slots in the TDMA template [14]. However, since different slave devices have different service property, round robin may not be the best option. Further research work is needed
to study the characteristics of the real world applications, such that the task models and system models proposed in this thesis can be applicable.

The third option to extend this research work is to consider fault tolerance issues. Many applications that have stringent timing requirements also have to meet high dependability requirements. In order to satisfy this requirement for dependability, the system must be able to continue to produce correct and timely outputs despite the presence of faults. Faults can be divided into three groups according to their duration: transient, temporary and permanent faults. Transient faults last for a short duration and affect only one task. Temporary faults are also of short duration but affect multiple tasks: every task executing at the time at which the fault occurs is affected. Permanent faults are of long duration. All the tasks using the faulty component are affected by the permanent fault. One of the most common techniques to achieve error recovery in general purpose systems is hardware redundancy, which can be used to tolerate all three kinds of faults, especially the permanent faults. Time redundancy is another technique to tolerate transient or temporary faults. After a fault is detected, extra time is spent to perform a certain fault recovery routine such that the correct outputs can still be generated within the timing constraints. It is a rather complex problem to combine our task model with the fault tolerance requirements, because of the strict timing constraints specified by the distance constraints. However, the issue is important in some critical systems, such as the satellite transmission applications. If we assume that the fault can be detected immediately after a task instance finishes its execution, a straightforward way to tolerate a transient fault is to transform the original constraint $max D_i$ of each task $r_i$ to a more strict maximum distance constraint $max D_i - 1$. Once a transient fault is detected at the end of a time slot when a task instance terminates, the following time slot is scheduled to execute the recovery routine, instead of the normal execution based on the scheduling template. In this way, the task instances that are scheduled after the faulty instance will be delayed by at most one time slot. Since we apply $max D_i - 1$ as the maximum distance constraint, the original constraint $max D_i$ is not violated. The minimum distance constraint is not influenced in this situation. Obviously, this method leads to worse schedulability because of the more strict distance constraints, and the fault model is limited to the strict assumptions. Serious research work is needed to find solutions for this fault tolerance problem.
Bibliography
Bibliography


