Effect of Scheduling Jitter on End-to-End Delay in TDMA Protocols *

Libin Dong, Rami Melhem, Daniel Mossé
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
(dong, melhem, mosse)@cs.pitt.edu
Fax: 1-412-624-5249

Abstract

In this paper, we address the problem of guaranteeing end-to-end (ETE) delay of packets in a distributed system where the technique of time division multiplex access (TDMA) is adopted, and the application at the destination node requires to process the packets at a regular rate. We present a time slot allocation algorithm to satisfy the transmission rate requirement, and to minimize the scheduling jitter. Each packet of a stream is delivered in an allocated time slot according to one of the two delivery protocols proposed in this paper. A worst case ETE delay bound is derived for each protocol. The performance of applying our scheduling algorithm to different delivery protocols is compared and evaluated via simulations.

1 Introduction

There has been an increased need for real-time communication services in applications. Predictable and guaranteed service has become one of the critical components of the *quality-of-service* (QoS) requirements, and is one of the main concerns in scheduling real-time traffic.

This paper addresses the issue of scheduling real-time streams in a distributed system where *Time Division Multiple Access (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. In TDMA, time is divided into equal length *slots*, each of which is equal to the transmission time of a message packet. A TDMA *frame/template* contains an integer number of time slots, and the allocation pattern of the template is applied repeatedly to control the transmission of message streams. Since the entire execution schedule is predetermined in TDMA [6], the timing constraints are guaranteed once a feasible template schedule is generated based on the constraints. Moreover, there is no run-time scheduling overhead, because the packets are transmitted according to a predetermined template. Thus, some distributed real-time systems, such as the MARS system [4], adopt TDMA based protocols.

Some applications require that the destination node continuously processes the received packets at a regular rate, which is called the *destination continuity requirement* in this paper. An example is the video-on-demand cable systems implemented in a limited area such as a hotel. In this case, the destination 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 the packets of the message stream at the destination node before it starts to process them. Obviously, this can lead to a large end-to-end (ETE) delay. In order to achieve the destination continuity requirement within a limited ETE delay, we propose a TDMA scheduling algorithm and corresponding 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 four components. The first is the transmission and propagation delay on each link, which does not depend on the scheduling discipline used, and will be factored out of our analysis. The second is the processing delay at each intermediate node. 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, and the processing delay is assumed to be negligible. The third is the buffer delay in each intermediate node, which is determined by the scheduling policy and the delivery protocol. The fourth is the *start-up* delay at the destination node. 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.

In order to control the magnitude of the ETE delay, we derive a TDMA scheduling algorithm to reserve time slots

*This work was supported in part by DARPA under Contract DABT63-96-C-0044, a part of the FORTS project.

for packet transmission. 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 an 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 [5]. A smaller jitter implies a smoother transmission pattern for the stream, such that the destination continuity requirement is easier to enforce. In order to obtain a small ETE delay, we develop a time slot allocation algorithm to satisfy the rate requirement and to minimize the scheduling jitter.

In the context of time slot allocation for a TDMA template, a related work is based on distance constraint problems [5], in which the time interval between every two adjacent allocated time slots to transmit two consecutive packets of the message stream must be within a specified range, such that scheduling jitter can be limited by the distance constraint. Other work has studied scheduling jitter in periodic task models [1], and template-based models [3]. All these algorithms apply to static task sets where the parameters of the set of tasks are known a priory. In this paper, we focus on the communication environments where message streams dynamically arrive and depart.

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 two delivery protocols. 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.

The rest of this paper is organized as follows. Section 2 introduces the system model. Section 3 presents the scheduling algorithm. The three delivery protocols are proposed in Section 4. Section 5 compares the two protocols and evaluates their performance via simulation. It also describes a general protocol that chooses a certain delivery discipline according to application types. The paper is concluded in Section 6.

2 System model

In this paper, we use M to denote a message stream, and M_j for the j^{th} packet of M. The packets of M are transmitted according to the scheduling template, as explained earlier. Let T be the number of time slots in the template. Because of the cyclical application of the template, the slot allocation pattern in the period $[t, t+T-1]$, for any t, will be repeated in every following interval $[t+kT, t+(k+1)T-1]$, for $k > 0$. We assume that all nodes in the system have the same template size, T, which 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. We assume that a route is generated by a certain routing algorithm, and will not be changed. The number of hops in the route is denoted by h, and the nodes on the route are labeled as $N_0, ..., N_h$ with N_0 being the source node and N_h the destination node. The link $L_{k,k+1}$ connects N_k and N_{k+1}.

We define the *input pattern* of a stream M at a node 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 *allocation pattern* of M at a node is the set of time slots that are allocated to M by the scheduling algorithm at 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 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 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 propagation 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, which is characterized as $M = (n, d)$. The ETE deadline of M is represented by d, which means that the actual ETE delay must be smaller than or equal to d. We represent the average transmission rate of M by n, in terms of the number of time slots that need to be allocated to M in the template of size T. Given T and n, the average distance between any two consecutive instances of M is equal to $avgD = \frac{T}{n}$. In order to prevent the overflow or underflow of the receiving buffer at the destination node, the average transmission rate is equal to the processing rate at the destination.

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 one packet in every $avgD$ 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 one packets per $avgD$ time slots.

3 Scheduling algorithm at each node

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 $M = (n, d)$ arrives at a certain node, some time slots in the template of this node are already allocated to existing message streams. We assume that 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 $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ be the set of time slots allocated by the scheduling algorithm to the n instances of M in the template, where S_j is the location of the j^{th} instance of M. 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, $S_{j+n} = S_j + T$.

Given the schedule $\mathcal{S}$, the set of temporal distances between all neighboring instances in the template is $\mathcal{D} = \{D_1, D_2, ..., D_n\}$, where $D_j = S_{j+1} - S_j$ 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 $\mathcal{S} = \{1, 2, 5\}$. Then $\mathcal{D} = \{1, 3, 2\}$. In this example, the packets of M will be transmitted at time slot $\{1, 2, 5, 7, 8, 11, 13...\}$ according to the schedule, and the corresponding distances between every two adjacent instances are $\{1, 3, 2, 1, 3, 2...\}$. The sum of all the D_j is equal to T, that is, $\sum_{j=1}^{n} D_j = T$.

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 $\mathcal{D}$ 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 $E[X] = \sum_{j=1}^{n} \frac{1}{n} D_j = T/n = avgD$. Given a schedule, the scheduling jitter is calculated by the following equation, where $Var[X]$ stands for the variance of X.

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

The scheduling problem at each node is formally stated as follows. Given a template of size T and a set of m vacant time slots $\{t_1, t_2, ..., t_m\}$ 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.

This problem can be transformed 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_i represents the i^{th} vacant time slot in the template, t_i. A directed edge $E_{i,j}$ points from vertex V_i to V_j where $i \neq j$. We explain the meaning of $E_{i,j}$ and define its weight $w(E_{i,j})$ as follows.

1. When $i < j$, $w(E_{i,j}) = (t_j - t_i - avgD)^2$, which means that if vacant slots t_i and t_j are chosen for two consecutive instances of a message stream, then distance between these two adjacent instances, $t_j - t_i$, contributes a factor of $(t_j - t_i - avgD)^2$ to the calculation of the scheduling jitter.
2. When $i > j$, $w(E_{i,j}) = (t_j + T - t_i - avgD)^2$, which means that if vacant slots t_i and $t_j + T$ are chosen for two consecutive instances of a message stream, then distance between these two adjacent instances, $t_j - t_i$, contributes a factor of $(t_j + T - t_i - avgD)^2$ to the calculation of the scheduling jitter.

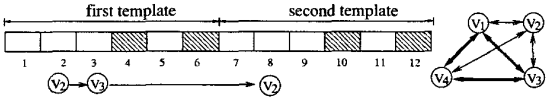


Figure 1. Example of graph transformation.

Example 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 $avgD = \frac{6}{3} = 2$. The graph transformation for Example 1 is illustrated in Figure 1, where the shaded slots indicate the ones occupied by previously allocated streams. A directed complete graph of 4 vertices is shown beside the time line, 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 $\mathcal{S} = \{S_1, ..., S_n\}$ can be represented by a cycle of n vertices in the graph where each vertex represents an allocated slot S_j. The weight of the cycle is calculated as the summation of the weights of all the edges in the cycle. According to Equation (1), the weight of the cycle is equal to n times the scheduling jitter of $\mathcal{S}$. Thus, a schedule of n instances with the minimum jitter corresponds to a cycle of n vertices with shortest weight.

We solve this shortest cycle problem using a dynamic programming technique [2]. Note that a shortest cycle that contains vertex V is equivalent to a shortest path from V 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 $\Pi^{(0)}$, ..., $\Pi^{(n)}$. 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.

$$\pi_{i,j}^{(k)} = \begin{cases} w(E_{i,j}) & (k=1) \\ min\{\pi_{i,q}^{(k-1)} + w(E_{q,j})\} & (1 < k \leq n) \end{cases} \quad (2)$$

The calculation of the matrix is shown below by by applying the dynamic programming technique to Example 1. In this example, 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.

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

In summary, the algorithm *min-jitter* selects n out of the m vacant time slots in the template to minimize the jitter. The algorithm is described as follows.

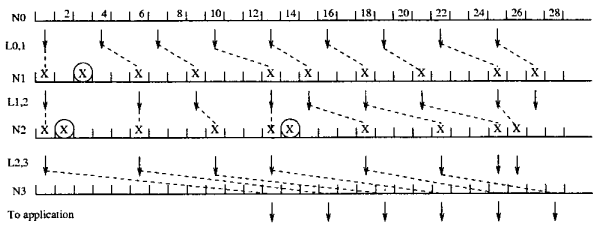
min-jitter

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.

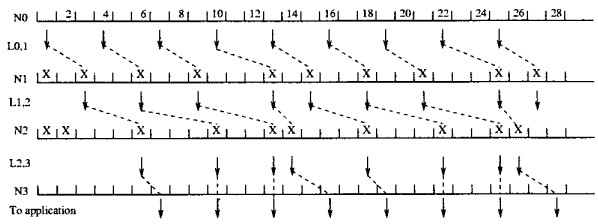
4 Delivery protocols

Once a new message stream is accepted, and a schedule is generated at each intermediate node during the connection establishment stage, the packets of the stream are delivered at each node according to one of two delivery protocols, NED and WED. 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 to the largest ETE delay in the worst case is derived for each scheme. In later subsections, we will use the following example to explain the two protocols.

Example 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 2, 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 an allocated slot. 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 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.



(a) scheduling result by applying NED to Example 2



(b) scheduling result by applying WED to Example 2

Figure 2. An ETE example of 3 hops

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 2(a). 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 changed 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 ns_k. In Figure 2(a), there is no slot skipping after slot 3 at node N_1, nor after slot 14 at N_2. So $ns_1 = 4$ and $ns_2 = 15$. The following theorem about the no-skip state and ns_k is stated without proof, because of the limit of space.

Theorem 1: When the buffering scheme of NED is applied, each node will reach the no-skip state. Moreover, $ns_k \leq ns_{k-1} + T - 1$.

In Figure 2(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 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 the following.

detecting discipline

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.

Because N_{h-1} reaches the no-skip state at time ns_{h-1}, the *detecting discipline* decides that N_h can start the processing at time $ns_{h-1} + T - 1$. In Figure 2(a), N_3 can detect the no-skip state of N_2 by receiving 4 packets from time 18 to time 26. So the start-up time decided by the detecting discipline is 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 to transmit the first packet. The ETE delay upper bound provided by the *detecting discipline* is thus $h(T - 1)$.

Before N_h detected the no-skip state, N_h receives packets at some unpredictable rate. The accumulation in the receiving buffer can be used to allow for an earlier start-up. Since the no-skip state must be reached at N_{h-1} by time $t_0 + (h - 1)(T - 1)$, we consider this time 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 as follows. 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.

approaching discipline

1. Once N_h receives the first packet, it sets a time-out value to be $t_0 + (h - 1)(T - 1)$.
2. When a new packet arrives, decrease the time-out value by $avgD$. 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 is smaller than or equal to the time-out value, N_h starts to deliver the packets to the application.

The ETE delay bound provided by the *approaching discipline* is $(h - 1)(T - 1)$. In the example of Figure 2(a), N_1 starts to transmit the packets at time 1. So the estimated bound is $1 + 2 \times 11 = 23$. When N_3 receives the 4^{th} packet at time 13, the time-out value has been decreased to 11. So it can immediately start to deliver the packets to the processing application at time 13.

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, $ns_k = ns_{k-1} + T - 1$. 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 start up. However, a late skipping before ns_k 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 ns. 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 using *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.
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 $(h-1)(T-1)$, the smaller of the two bounds provided by 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 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.

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 M, 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 at node N_k, in which every element corresponds to a possible arrival time for the first packet at N_k. We use S_j^k to denote the time slot allocated to the j^{th} instance of M in the template of node N_k. The set $\{S_1^{k-1}, S_2^{k-1}, ..., S_n^{k-1}\}$ 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 is received by N_k at time S_j^{k-1}, and is transmitted by N_k at time S_i^k so that no future skipping will occur at N_k, then the local delay pair at N_k is (S_j^{k-1}, y_j) where $y_j = S_i^k - S_j^{k-1}$. The set of local delay pairs at node N_k, $\mathcal{P}_k = \{(S_j^{k-1}, y_j), j \in [1, n]\}$, can be found from the following algorithm.

construct-pairs

1. Assume that n packets arrive at N_k in the slots $\{S_1^{k-1}, ..., S_n^{k-1}\}$. Find $\{z_1, ..., z_n\}$, 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 $z_n = S_g^k$, which means that the n^{th} received packet is transmitted by N_k at the location of the g^{th} instance. Then $\{S_{g-n+1}^k, ..., S_g^k\}$ is the set of time slots at which each packet will be transmitted by N_k without slot skipping. The j^{th} packet received at time S_j^{k-1} will be transmitted at time S_{g-n+j}^k by N_k, for $j \in [1, n]$.
3. Construct the set $\mathcal{P}_k$ with pairs (S_j^{k-1}, y_j) for all $j \in [1, n]$, where $y_j = S_{g-n+j}^k - S_j^{k-1}$.

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 time S_n^{k-1} will be transmitted at the location S_g^k, then there must be $g-n$ skipped slots at N_k in the interval $[1, S_g^k]$. 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 n continuously allocated slots S_{g-n+1}^k,...,S_g^k.

By applying this method to Example 2, Figure 2(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 2(b) where M_1 is delivered by N_1 at the second allocated slot. In this case, the set of local delay pairs is $\{(1, 2), (4, 2), (7, 2), (10, 3)\}$.

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 M_1 is received by node N_k at time t. We define a function $\overline{mod}(t)$ that returns the modulus of the division of t by T if it is not 0. Otherwise, $\overline{mod}$ returns T. Let $t' = \overline{mod}(t)$. So $t' \in [1, T]$, and $t - t' = i \times T$ for some integer i. Because of the template repetition, an arrival at t is equivalent to an arrival at t'. Hence, find a local delay pair (S_j^{k-1}, y_j) at N_k in the set $\mathcal{P}_k$ such that $S_j^{k-1} = t'$. Then y_j is the shortest amount of time that the first packet 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 $\overline{mod}(t)$ and the set $\mathcal{P}_h$, 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 ev-

ery $avgD$ 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*. The ETE delay of a packet is calculated by using an *accumulated delay pair* $\langle z, u \rangle$ at each node N_k, which means that the packet transmitted by N_k at time z has experienced an accumulated delay of u time slots from the source node to N_k.

exact-ETE-delay

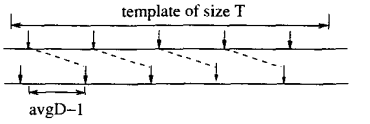
1. At the first node N_1, assume that a local delay pair (x, y) is in $\mathcal{P}_1$, indicating that an input at x is delayed at N_1 by y slots and is transmitted at time $z = x + y$. Then form an accumulated delay pair $\langle z, y \rangle$.
2. The accumulated delay pair is transmitted to the next node.
3. When node N_k $(1 < k \leq h)$ receives the accumulated delay pair $\langle z, u \rangle$, it finds a local delay pair (x_j, y_j) in $\mathcal{P}_k$ such that $x_j = \overline{mod}(z)$. The accumulated delay pair is updated to $\langle z + y_j, u + y_j \rangle$, 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 $\langle z, u \rangle$.

In the *exact-ETE-delay* algorithm, N_1 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.

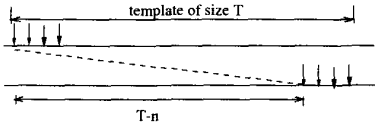
In summary, the following steps abstract the behavior of the protocol WED. Figure 2(b) shows the result of applying WED to Example 2. Since the packets are delayed by the exact amount to achieve continuity at the destination, the ETE delay is shorter than the one shown in Figure 2(a), where the worst-case estimate is used.

1. At the connection establishment stage, each intermediate node N_k needs to execute the following steps. The destination node only executes the second and the third steps.
   (a) Find the allocation pattern $\{S_1^k, ..., S_n^k\}$ based on the scheduling algorithm *min-jitter*.
   (b) Construct the set of local delay pairs $\mathcal{P}_k$.
   (c) Calculate the accumulated delay pair.
   (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.

The ETE delay bound is determined by the scheduling jitter. The effect is illustrated in Figure 3. Figure 3(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 3(b) shows the worst case where the scheduling jitter is the largest and the instances 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 $(h-1)(T-n)$.



(a) with 0 jitter



(b) with the largest jitter

Figure 3. the best and the worst case

5 Performance evaluation

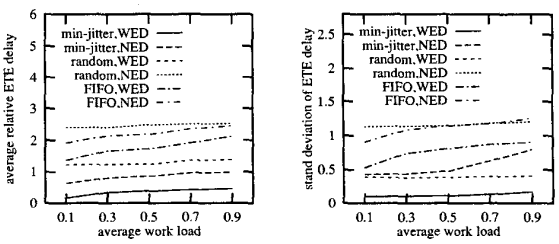
We presented two delivery protocols in the previous section. They are compared in terms of connection control, intermediate node processing and ETE delay as follows.

1. NED has a simpler connection establishment scheme than WED, since WED needs more overhead to find the local delay pairs, to transmit additional information, and to store the additional information at each intermediate node.
2. During the transmission stage, the delivery schemes of NED is straightforward. WED needs to deliberately delay the first packet according to the set of delay pairs stored at each node.
3. In terms of ETE delay, WED achieves a shorter ETE delay than NED to satisfy the destination continuity requirement, subject to the allocation pattern on each intermediate node.

We simulated the NED and WED protocols in a distributed system consisting of 20 nodes with dynamic message arrivals and departures. 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 *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 two other scheduling algorithms, namely, *FIFO-schedule* and *random-schedule*. 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 4(a) shows the average relative ETE delay of all the tasks versus system workload. The 6 curves represent the performance of the 6 combinations. 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. 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. Figure 4(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.



(a) average ETE delay (b) standard deviation

Figure 4. Performance of different workload

6 Conclusion

In this paper, we address the communication QoS problem in which ETE delay of a message stream in a distributed system needs to be guaranteed, and at the same time, the destination continuity requirement needs to be satisfied. We solve the problem based on a TDMA slot allocation algorithm *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 *min-jitter* substantially improves the ETE delay when compared to other algorithms in which scheduling jitter is not taken into consideration. Two 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.

References

[1] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. *International Conference on Real-Time Computing Systems and Applications*, pages 62–69, November 1999.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction To Algorithms*. The MIT Press, Cambridge, Massachusetts London, England, 1994.

[3] L. Dong, D. Mosse, and R. Melhem. Time Slot Allocation for Real-Time Messages with Negotiable Distance Constrains. *Proceedings of IEEE Real-time Technology and Applications Symposium*, pages 131–136, June 1998.

[4] T.M. Galla and R. Pallierer. Cluster Simulation-Support for Distributed Development of Hard Real-Time Systems using TDMA-Based Communication. *Proceedings of EuroMicro Conference on Real-Time Systems*, June 1999.

[5] C.W. Hsueh and K.J. Lin. Schedulability Comparisons Among Periodic and Distance-Constrained Real-Time Schedulers. *International Conference on Real-Time Computing Systems and Applications*, pages 60–66, October 1997.

[6] C.D. Locke. Software Architecture for Hard Real-Time Applications:Cyclic Executives vs. Fixed Priority Executives. *Real-Time Systems Journal*, 4:37–53, 1992.