Time-Constrained Scheduling of Weighted Packets on Trees and Meshes

Micah Adler* Sanjeev Khanna† Rajmohan Rajaraman‡ Adi Rosén*

Abstract

The time-constrained packet routing problem is to schedule a set of packets to be routed through a multi-node network, where every packet has a source and a destination (as in traditional packet routing problems) as well as a release time and a deadline. The objective is to route the maximum number of packets subject to these constraints. This problem was studied in [1], where it was shown that the problem is NP-Complete even when the underlying topology is a linear array. Approximation algorithms were also provided in [1] for the linear array and the unidirectional ring for both the case where packets may be buffered in transit and the case where they may not be.

In this paper, we extend the results of [1] in two directions. First, we consider the more general network topologies of trees and meshes. Second, we associate with each packet a measure of utility, called a weight, and study the problem of maximizing the total weight of the packets that are routed subject to their timing constraints. For the bufferless case, we provide a constant factor approximation for the time-constrained routing problem with weighted packets on a tree, and on a mesh. We also provide a logarithmic approximation for the same problems in the buffered case. These results are complemented by new lower bounds, which demonstrate that we cannot hope to achieve the same results for general network topologies. For example, we show that if n packets are required to follow prescribed paths in an arbitrary graph, then it is NP-Hard to provide even an $n^{1-\epsilon}$-approximation, for any $\epsilon > 0$, to the optimal set of packets that can be routed.

*Department of Computer Science, University of Toronto, 10 King's College Road, Toronto ON, M5S 3G4, Canada. Email: {adiro,micah}@cs.toronto.edu

†Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974–0636, USA. Email: sanjeev@research.bell-labs.com

‡College of Computer Science, 161 Cullinane Hall, Northeastern University, Boston MA 02115, USA. Email: rraj@ccs.neu.edu

1 Introduction

Recent research in communication and interconnection networks has seen a growing emphasis on networks that are capable of delivering packets with timing constraints. For communication networks, the shift to this type of routing from traditional best effort routing is motivated by multimedia applications such as real-time video and audio [13]. For example, a real-time video packet that does not arrive within a given window of time serves little or no purpose. For interconnection networks the analogous shift is motivated by emerging real-time applications that rely on time-constrained communication, such as industrial process control and avionics [20].

Another important aspect of many networks is that different packets may have different levels of importance or usefulness. For example, video image encodings such as JPEG and MPEG have the property that the quality of the image produced is very sensitive to what portion of the encoding is available for decoding, and there is a clear priority of which packets are the most important. Variation in packet utility can also result from differences in the importance of packets sent by different applications. For example, on a network requiring payment for the delivery of packets, some customers may be willing to pay more for improved quality of service. In such a scenario, the network provider would want to deliver the packets that provide the most total revenue.

In this paper, we consider the time-constrained packet routing problem introduced in [1]. The objective is to schedule a set of packets with timing constraints to be routed through a given multi-node network. Each packet is specified by its *source node*, its *destination node*, and its *routing path*, as well as a *release time*, a *deadline*, and a *weight*. The weight of a packet represents the utility provided by that packet; in [1], all weights were assumed to be uniform. A packet cannot start its journey from its source node before its release time, and has to arrive at its destination by the deadline to serve any purpose. Thus, the objective is to maximize the total weight of the packets that arrive by their deadlines. Note that this means that a packet should be dropped if it will not arrive by its deadline, since forwarding such a packet wastes network resources.

The Network Model. We model the network as an undirected graph $G = (V, E)$, where the set of nodes is the set of processors, and the set of edges is the set of communication links. We consider a *synchronous* model, where in each time step each link can transmit one packet in each direction. There are no other limitations on the number of packets that each processor can send/receive at each time step.

We distinguish between two cases, depending on the buffering policies at the nodes. In the *buffered* case, nodes are allowed to store packets that they receive and then transmit them at a later time. In the *bufferless* case a packet is not allowed to be buffered at any node other than the source node. Thus, a packet m that crosses a link to a node v at time step t, has to cross the next link, leaving v, at time step $t+1$. This second case, which is particularly appropriate for optical networks [10], has been studied in [4, 19]. Furthermore, as was seen in [1], the bufferless case is closely related to the buffered case, and can provide important insights into buffered routing.

An instance of the buffered (resp. bufferless) problem that we consider consists of a graph G, and a set of n messages M. Each $m \in M$ is defined by a tuple (s, t, π, r, d, w), where s is the source of the packet, t its destination, π is a simple path that the message has to follow from s to t, r is the release time, d is the deadline and w is its weight, i.e., the benefit accrued if the packet arrives at t by the deadline d. We refer to the length of the path π as the *span* of the message, and we refer to the number of steps that a packet can sit idle between its release time and deadline as the *slack* of the packet. We denote the weight of a packet m by $w(m)$. Given a set S of packets, we let $w(S)$ denote $\sum_{m \in S} w(m)$. The goal of the buffered problem (resp. bufferless problem) is to determine a maximum weight subset $M' \subseteq M$ that can be scheduled using buffers at the nodes (resp. without buffers) so that all messages in M' arrive by their deadline.

Summary of Results. The first results in this framework were given in [1], where linear-array and ring networks were studied and it is assumed that all packets have the same weight. In this paper we extend the results of [1] in two directions. First, we study the more general network topologies of the tree and the mesh. Second, we allow the packets to have different weights, and find an approximate solution to the problem of maximizing the total weight of packets that arrive by their deadlines. Note that [1] showed that even in the case of uniform weights on the linear array, finding the exact solution is NP-Hard, and thus an approximate solution to the problems we consider is the best we could hope to find efficiently. All algorithms we consider are centralized and off-line.

For the tree we present an algorithm for the bufferless case that achieves a constant approximation ratio. For the special case that the packets have uniform weights, this algorithm is a 3-approximation. We also demonstrate that the use of buffers can only increase the weight of the packets successfully delivered by a factor of $O(\log T)$, where T is the minimum of the number of packets in the optimal buffered solution, and the maximum slack of any packet in the optimal buffered solution. Thus, the bufferless approximation algorithm also provides a logarithmic approximation ratio for the buffered case. These results appear in Section 2.

We further demonstrate that the techniques developed for the tree can be applied to the 2-dimensional mesh. We here assume that all of the paths π are dimension order paths. This kind of routing, which is commonly used in practice on the mesh, requires that each packet travels along its source row to the correct column and then along that column to the correct row. All of the results described for the tree also apply to dimension order routing on the mesh. These results can be extended to higher dimensional meshes, where the dependence on the dimension d imposes an additional factor of $d \cdot 2^d$ in the approximation ratios. These results are described in Section 3. We further show that in the case of weighted packets on the linear array (a special case of the tree), the constants of the approximation ratios can be improved, and they match those that have been shown in [1] for the case of unweighted packets. These results appear in Section 4.

We complement the above results by showing that there is a graph G such that unless $NP = ZPP$, there is no polynomial time algorithm for the time-constrained routing problem on G with n packets that would achieve an approximation ratio of $O(n^{1-\epsilon})$. Thus, we cannot hope to achieve a good approximation ratio for all network topologies. Furthermore, this result can be extended to the mesh, provided that the paths defined for the packets can be arbitrary. These results apply to both buffered and bufferless routing, and justify our concentration on the tree and mesh topologies, as well as the focus on dimension order routing for the mesh. These results appear in Section 5.

For the unweighted case of linear arrays, [1] provided a distributed and on-line algorithm for the buffered case that is guaranteed to achieve within a factor of 2 of the optimal (off-line and centralized) bufferless schedule. In this paper, we demonstrate that the analogous result is not possible in the case of a tree, even when all packets have the same weight. In particular, we present a specific tree network and a sequence of packets for which any buffered, deterministic, on-line algorithm would be able to schedule at most one packet, while an off-line bufferless algorithm is able to schedule h packets, where h is the height of the tree. These results appear in Section 6.

Related Work [1] dealt with the unweighted case of the time-constrained routing problem on the linear array topology. To the best of our knowledge, this provides the only previous rigorous analysis of algorithms for the case of packets with arbitrary release times and deadlines. They provide a 2-approximation algorithm for the bufferless case, and show that buffers can increase the number of packets routed by only a logarithmic factor (and in several important cases by only a small constant factor). Thus, the bufferless algorithm can also be used to provide a logarithmic approximation algorithm for the buffered case. In addition, they demonstrate how to simulate the bufferless algorithm, which is off-line and centralized, in a distributed and on-line fashion that requires the use of buffers.

[15] considers the problem of routing a set of packets with arbitrary deadlines but all with the same release time on the linear array *without* dropping any of the messages. They show that if there exists a feasible schedule then the closest-deadline-first greedy strategy succeeds in routing all the messages.

A number of papers consider the "session model" where packets are introduced at a fixed rate for each of a number of "sessions". Each session consists of a given path from a source to a destination [17, 18, 2]. The aim is to schedule the packets across different links of the network with guaranteed (small) delays that depend on the rates of the sessions and the congestion on the links along the paths of the sessions. In a recent paper [3], *delay requirements* are added to the sessions, so that packets of a given session request to be routed with a delay no larger than the requirement. [3] provides a distributed packet scheduling algorithm such that, if two necessary conditions on the set of packets being routed within this delay requirement are met, then the packets arrive with a delay that is – roughly speaking – at most a logarithmic factor (in the size of the network) larger than the delay requirement. This is in contrast to our work where the deadlines *must* be met.

A number of empirical works have examined routing with timing constraints. In [20] a router architecture is given for messages with deadlines. A minimum-laxity-first protocol is proposed in [24] for transmitting messages with deadlines in a multi-access shared-bus network. Some scheduling policies—such as Virtual Clock [23], Stop-and-Go [6], Rotating Combined Queuing [9]—do not explicitly use message deadlines, but just attempt to keep the worst-case message delay small and bounded. Other relevant experimental work includes [16, 22, 14, 21, 5].

Of course, routing without timing constraints has been the subject of a large number of works; see [11, 12] for a survey. These works usually attack the problem under a *best effort* model, and thus focus on optimizing global performance measures such as the overall completion time of a routing problem, or the maximum and/or expected delay experienced by any packet.

2 Routing on the tree

In this section, we present an approximation algorithm for the problem of time-constrained routing weighted packets on the tree. We consider both bufferless and buffered schedules. In Section 2.1, we present an $O(1)$-approximation algorithm **WT** for bufferless schedules. In Section 2.2, we show that for any problem instance, the total weight of the packets routed in the optimal bufferless solution is within a logarithmic factor of the weight of the packets in any optimal buffered schedule. Thus, **WT** also provides an approximation algorithm for the buffered case.

2.1 Bufferless schedules

We begin by introducing some notation that is needed to describe the algorithm. Choose any node of the tree to be the *root*. We divide the path of each packet into a *rootward* direction (where the packet is moving towards the root) and a *leafward* direction. Note that some packets may consist of only a rootward or only a leafward direction. A packet will be scheduled to travel in the rootward direction along an *up-tree*. An up-tree is a copy of the given tree with a label $t(e)$ on each edge e that satisfies the following condition: if $d(e)$ denotes the number of edges between e and the root, then the quantity $d(e) + t(e)$ is the same for every edge e in the up-tree. The preceding definition of an up-tree is motivated by the following observation: for any packet that is routed in a bufferless schedule, there exists a unique up-tree in which the label of every edge of the packet's rootward path equals the time step at which the packet crosses that edge in the schedule.

We sort the up-trees in terms of their value of $d(e)+t(e)$, where we say that the up-tree with the smallest value of $d(e) + t(e)$ is the *earliest* up-tree and the largest value of $d(e) + t(e)$ is the *latest* up-tree. For each packet, the release time and deadline for that packet define a set of up-trees such that the packet obeys its constraints if and only if it is routed on one of the up-trees in this set. The up-trees in this set form a contiguous set in the sorted order of up-trees from earliest to latest. We say that the packet is *eligible* to be routed on these up-trees. We can also assign packets that only travel in a leafward direction to up-trees. In this case, we associate a scheduled time t for these packets with an up-tree U such that if e' is the first rootward edge from the source of the packet, then the value of $d(e) + t(e)$ for U is $d(e') + t - 1$. This means that a packet that has e' as its first leafward edge will be associated with the same up-tree if it traverses edge e' at time t regardless of whether it has a rootward component or not.

Algorithm **WT**:

Let S be the empty set.
Let N be the set of all packets.
Let $U(1) \dots U(k)$ be the set of all possible up-trees in order from latest to earliest.
For $i = 1$ to k;
    Let $E(i)$ be the set of packets in $N - S$ that are eligible for routing on $U(i)$.
    Prune light eligible packets.
        For every packet p in $E(i)$:
            Let S' be the set of packets in S that would conflict with p if p were routed on $U(i)$.
            If $w(S') \geq w(p)/4$ remove p from $E(i)$.
    Select new packets.

Let $S(i)$ be the empty set.
Repeat until $E(i)$ is empty.
Let p be a packet in $E(i)$ with the last root-ward edge that is furthest from the root.
Remove p from $E(i)$.
Let $S'(i)$ be the set of packets in $S(i)$ that conflict with p if p were routed on $U(i)$.
If $w(S'(i)) < w(p)/4$ remove all packets in $S'(i)$ from $S(i)$ and add p to $S(i)$.
Assign the packets in $S(i)$ to $U(i)$.
Purge conflicting packets.
Remove any packets in S which conflict with packets in $S(i)$.
Add the packets in $S(i)$ to S.
Return S.

As described above, **WT** sequentially processes all of the up-trees in order from the latest to the earliest. Each of the iterations that processes an up-tree may be divided into three phases: *pruning*, *selection*, and *purging*. Each iteration may add new packets to and delete old packets from the solution set S that is maintained during the algorithm. After all of the up-trees are processed, S is returned as the set of packets that are routed. We say that a packet p is *selected in iteration* i, if, during the selection phase of iteration i, p is ever added to the set $S(i)$. We now show that the total weight of the solution returned by **WT** is within a constant factor of the optimal weight. For any instance I, let $ALG(I)$ and $OPT(I)$ denote the set of packets routed by **WT** and in the optimal bufferless schedule, respectively.

Theorem 1 *For any weighted problem instance I for the tree, $w(ALG(I)) = \Omega(w(OPT(I)))$.*

Proof: We shall assign blame for each packet in $OPT(I) - ALG(I)$ to the packets in $ALG(I)$. To justify such a technique, we first state an easily proved lemma that was used in a simpler form in many of the proofs in [1].

Lemma 1 *If there exists a way to assign blame for every packet in $OPT(I) - ALG(I)$ to the packets in $ALG(I)$ such that the total blame assigned for any packet $p \in OPT(I) - ALG(I)$ is at least $w(p)$, and the total blame assigned to any packet q is at most $x \cdot w(q)$, then $w(ALG(I)) \geq \frac{1}{x+1} w(OPT(I))$.*

In order to assign the blame for packets, we make use of a directed graph, which we refer to as the *blame graph*. Each node in the blame graph is colored red, blue, or green, and has an associated weight. We first describe the set of blue and green nodes and the edges between them.

Blue and green nodes. For every instance that a packet p is selected during the selection phase of the processing of an up-tree, say $U(i)$, we have a distinct node $\langle p, i\rangle$ in the blame graph. If a packet p selected in iteration i is discarded subsequently (either within iteration i or in subsequent iterations), then we color the node $\langle p, i\rangle$ blue; otherwise, we color it green. Thus, we have exactly one green node for every packet in the final solution returned by **WT**. There can be more than one blue node for a packet. Consider a blue node $\langle p, i\rangle$. By the definition of a blue node, after the selection of p in iteration i one of the following events occurs: either p is removed subsequently in the selection phase of iteration i or the packet p is removed in the purging phase of iteration $i' > i$. Let p' denote the unique packet selected in iteration $i' \geq i$ that causes the removal of the packet p. In the blame graph, we have a directed edge from the blue node $\langle p, i\rangle$ to the (blue or green) node $\langle p', i'\rangle$. There is no out-going edge from a green node. Finally, we set the weight of a blue or green node $\langle p, i\rangle$ to $w(p)$.

Red nodes. The blame graph also contains some number of red nodes for every packet in $OPT(I)$ that is never selected in **WT** and hence does not have any associated blue or green nodes. Consider a packet x that is never selected in **WT** and yet is routed along up-tree $U(i)$ in the optimal solution. Since x is neither routed on $U(i)$ nor assigned to any $S(j)$, it follows that one of the following two events must have occurred during the processing of $U(i)$ in **WT**: (i) x is removed in the pruning phase, or (ii) when x is considered in the selection phase, x is not selected. We now consider each of the two cases. Let S be the set of packets in the solution at the start of iteration i.

We first consider the case in which x is removed in the pruning phase of iteration i. Then, the subset S' of packets in S that conflict with the routing of x along $U(i)$ satisfies the inequality: $w(S') \geq w(x)/4$. In this case, we have a red node $\langle x, q\rangle$ in the blame graph for each packet q in S'. For each q in S', we have a directed edge from $\langle x, q\rangle$ to $\langle q, i'\rangle$, where $i' < i$ is the iteration in which the packet q has been selected most recently. (We note that by our definition of blue and green nodes, we have a node $\langle q, i'\rangle$ in the blame graph.) We set the weight of the red node $\langle x, q\rangle$ to $w(x)w(q)/w(S')$, which is at most $4w(q)$. Thus, the total weight of all of the red nodes corresponding to x equals $w(x)$.

We next consider the case in which x is accepted in the pruning phase and yet is not selected in the selection phase of iteration i. This implies that a routing of x along $U(i)$ conflicts with a set $S'(i)$ of packets already selected in $S(i)$ such that $w(S'(i)) \geq w(x)/4$. As in the first case, we have a red node $\langle x, q\rangle$ for each packet q in $S'(i)$, and for each q in $S'(i)$, we have a directed edge from $\langle x, q\rangle$ to $\langle q, i\rangle$. We set the weight of the red node $\langle x, q\rangle$ to $w(x)w(q)/w(S'(i))$, which is at most $4w(q)$. Again, the total weight of all of the red nodes corresponding to x equals $w(x)$. This completes the description of the red nodes and their incident edges.

The blame graph as defined above is a forest, in which each tree has a green root, and all edges in the tree are directed towards this root. This observation follows from the following facts: (a) each non-root node has out-degree 1, and (b) each node has an edge to a node that is either not dropped, or dropped at a later point in the execution of **WT**, thus implying that there are no cycles. Furthermore, all the red nodes in the blame graph appear as leaves in their respective trees. We refer to each tree in the blame graph as a *blame tree*.

Assigning blame. We are now ready to describe how we assign blame. For each packet p in $OPT(I)-ALG(I)$, if there is a blue node $\langle p,i\rangle$ in a tree with root $\langle r,i'\rangle$, then all the blame for packet p is assigned to the packet r. If there are two or more blue nodes for p then one is chosen arbitrarily. If there is no blue node for p, then there is one or more red nodes for p, $\langle p,q_1\rangle \ldots \langle p,q_k\rangle$, with roots $\langle r_1,i_1\rangle \ldots \langle r_k,i_k\rangle$ (where it is possible that $r_a = r_b$, for $a \neq b$). The blame for packet p is assigned to the nodes $r_1 \ldots r_k$, where the amount of blame received by r_ℓ is equal to the weight of $\langle p,q_\ell\rangle$. Thus, the total blame assigned for each packet in $OPT(I)-ALG(I)$ is equal to the weight of the packet, and the total blame assigned to each packet q in $ALG(I)$ is at most the weight of the blue and red nodes in the tree rooted at the green node $\langle q,i\rangle$. Therefore, Lemma 1 implies that Theorem 1 follows from the following lemma.

Lemma 2 *The total weight of the red and blue nodes in any blame tree is at most 17 times the weight of the green root.*

We prove Lemma 2 by the following sequence of claims.

Claim 1 *The total weight of the red children of any node x is at most $8w(x)$.*

Proof: We have already seen that the weight of any red node is at most four times the weight of its parent. Thus, we only need to show that every node has at most 2 red children. Consider a blue or green node $x = \langle p,i\rangle$ with a red child, say $\langle q,p\rangle$. Let $U(j)$ be the up-tree along which q is routed in the optimal solution. We now show that a routing of packet q along $U(j)$ conflicts with a routing of packet p along $U(i)$ either on the last rootward edge traversed by p or on the first leafward edge traversed by p. This is sufficient to prove the desired claim that $\langle p,i\rangle$ has at most two children since only one packet in the optimal solution can traverse those edges for any routing of packet p.

We now consider two cases depending on the phase in which the packet q is discarded in iteration j. We first consider the case where the packet q is accepted in the pruning phase and is not selected in the selection phase. In this case, p must belong to the set $S'(j)$ of packets that were selected in the selection phase prior to considering q and conflict with the routing of q along $U(j)$. Moreover, $i=j$. By our ordering in the selection phase, the packet p must have a last rootward edge that is no closer to the root than the last rootward edge of the packet q. Since the routing of packet p along $U(i)$ conflicts with the routing of packet q along $U(i)$, one of the following two claims hold: either packets q and p conflict on the up-tree, in which case q must contain the last rootward edge of the packet p, or they conflict during the leafward portion of their routing, which in this case can only happen if they have the same first leafward edge.

We now consider the case in which the packet q is discarded in the pruning phase of iteration j. In this case, p must have been selected prior to iteration j; that is, $i<j$. Since the up-trees $U(j)$ and $U(i)$ are different, it follows that the routing of q along $U(j)$ and the routing of p along $U(i)$ can conflict only during the leafward portion. However, for two paths that only travel in a leafward direction to overlap, one must contain the first leafward edge of the other. If those two packets traverse that edge at the same time, then the packet that starts its leafward journey at an earlier time must contain the first leafward edge of the other packet. Since we process the the up-trees from latest to earliest, packet p starts its leafward journey at a later time than packet q, and thus the routing of packet q along $U(j)$ contains the first leafward edge of the routing of p along $U(i)$. ■

Claim 2 *The total weight of the blue children of any node x is at most $w(x)/2$.*

Proof: Any blue or green node $x = \langle p,i\rangle$ can have a blue child $y = \langle q,j\rangle$ in one of two ways: (i) $j=i$ and in the selection phase of iteration i, packet q is first selected and is later discarded in favor of packet p, and (ii) $j<i$ and packet q is selected in iteration j and is discarded in favor of packet p during the purging phase of iteration i. In either case, the total weight of the blue children, which equals the total weight of the packets removed, is at most $w(x)/4$. ■

Claim 3 *For any node x in a blame tree, the total weight of all of the subtrees rooted at the blue children of x is at most $9w(x)$.*

Proof: We prove by induction on the maximum distance of the node x from any leaf. Clearly the claim is true when the node x is a leaf (i.e., when the distance is 0). For the inductive step, we can assume that the claim is true for all of the children of x. By Claim 2, the total weight of the blue children of x is at most $w(x)/2$. Thus, by the inductive hypothesis, the total weight of the subtrees rooted at all of the blue grandchildren of x is at most $9w(x)/2$. By Claim 1, the total weight of the red children of the blue children of x is at most $4w(x)$. Adding the three kinds of weights together, we obtain $9w(x)/2 + 4w(x) + w(x)/2 = 9w(x)$. ■

Proof of Lemma 2: To prove the Lemma, we see by Claim 3 that the total weight of the subtrees rooted at the blue children of the root r is at most $9w(r)$, and by Claim 1, the total weight of the red children of the root is at most $8w(r)$. ■

Theorem 1 now follows from Lemmas 1 and 2. ■

We can also show better constants for the unweighted case on the tree.

Theorem 2 *For any problem instance I for the tree where all packets have the same weight, $w(ALG(I)) \geq w(OPT(I))/3$.*

Proof: (sketch) The proof follows along the lines of Theorem 1. However, since all packets have the same weight, any packet that is ever selected by the algorithm will appear in the final solution. This means that there are no blue nodes in the blame trees, and thus each blame tree consists only of a green node with its red children.

Furthermore, the weight of any red child is now bounded from above by the weight of its parent. We saw in the proof of Claim 1 that any node can have at most two red children. Thus, the total weight of the red nodes in any blame tree is now at most twice the weight of the green root.

We again assign the blame for the packets in $OPT(I) - ALG(I)$ to the packets in $ALG(I)$. For any node p in $OPT(I) - ALG(I)$, there must be one or more red nodes $\langle p, q_1 \rangle \ldots \langle p, q_k \rangle$, with roots $\langle r_1, i_1 \rangle \ldots \langle r_k, i_k \rangle$ in the blame graph (where it is possible that $r_a = r_b$, for $a \neq b$). The blame for packet p is assigned to the nodes $r_1 \ldots r_k$, where the amount of blame received by r_ℓ is equal to the weight of $\langle p, q_\ell \rangle$. However, now the total weight assigned to any packet in $ALG(I)$ is at most twice the weight of that packet. The theorem follows from Lemma 1. ■

2.2 Buffered schedules

We now turn to the question of how much buffering can help on the tree. We shall see that it is limited, and thus algorithm **WT** can also serve as a buffered approximation algorithm. For any problem instance I, let $OPT_B(I)$ denote the set of packets routed in an optimal buffered solution, and let $|OPT_B(I)|$ denote the number of packets in $OPT_B(I)$. Let $\sigma(I)$ be the maximum slack of any packet in $OPT_B(I)$. Let $T(I) = \min(|OPT_B(I)|, \sigma(I))$.

Theorem 3 *For any problem instance I for the tree, $w(OPT_B(I))$ is $O(\log T(I) \cdot w(OPT(I))$.*

Note that in [1], we see that there is a problem instance I for the linear array, (a special case of the tree) such that buffers do increase the number of packets that can be routed by a factor of $\Omega(\log T)$. Thus, the result of Theorem 3 is optimal in terms of the number of packets and the maximum slack. The instance in [1] requires a linear array of length $T(I)$.

Proof: Without loss of generality, we may assume that the given instance I is such that the optimal buffered schedule routes all of the packets in I. We now show that without buffers, we can still route packets with a total weight of $\Omega(w(I)/(\log T(I)))$. In fact, we show that algorithm **WT**, applied to the set I achieves this weight. Our comparison of **WT** with the optimal buffered schedule is similar to our analysis in Section 2.1, and also uses techniques developed for comparing bufferless schedules to buffered schedules developed in [1]. The main idea of the following proof is that of defining a blame graph.

The blame graph. The green and the blue nodes of the blame graph as well as the edges between them are defined exactly as in Section 2.1. The definition of red nodes, however, is different. In the proof of Theorem 1, all the red nodes associated with a packet were defined on the basis of a single up-tree, which is the up-tree along which the packet is routed in the optimal schedule. In the optimal buffered schedule, however, a packet may be routed in a rootward direction along a number of different up-trees. Therefore, for any packet p in the optimal buffered schedule that does not have a green or a blue node, the blame graph has a collection of red nodes for each eligible up-tree for p.

We now define the red nodes. Consider a packet p for which there is no green or blue node in the blame graph. Let k be the number of up-trees on which p is eligible. Let $U(i)$ be any such up-tree. Since p is neither routed on $U(i)$ nor ever selected in the selection phase of any iteration, exactly one of the following two events must have occurred during iteration i of **WT**: (i) p was removed in the pruning phase, or (ii) p was accepted in the pruning phase but not selected in the selection phase. We now consider each of the two cases.

If p was removed in the pruning phase of iteration i, then the subset S' of packets in S that conflict with the routing of x along $U(i)$ satisfies the inequality $w(S') \geq w(p)/4$. In this case, we have a red node $\langle p, r \rangle$ in the blame graph for each packet r in S'. For each q in S', we have a directed edge from $\langle p, q \rangle$ to $\langle q, i' \rangle$, where $i' < i$ is the iteration in which the packet q has been most recently selected. We set the weight of the red node $\langle p, q \rangle$ to $w(p)w(q)/(kw(S'))$, which is at most $4w(q)/k$. Thus, the total weight of all of the red nodes corresponding to p equals $w(p)/k$.

We next consider the case in which p is accepted in the pruning phase and yet is not selected in the selection phase of iteration i. This implies that a routing of p along $U(i)$ conflicts with a set $S'(i)$ of packets already selected in $S(i)$ such that $w(S'(i)) \geq w(p)/4$. In this case, we have red node $\langle p, q \rangle$ for each packet q in S'. As in the first case, for each q in $S'(i)$, we have a directed edge from $\langle p, q \rangle$ to $\langle q, i \rangle$. We set the weight of the red node $\langle p, q \rangle$ to $w(p)w(q)/(w(S'(i))k)$, which is at most $4w(q)/k$. Thus, the total weight of all of the red nodes corresponding to x equals $w(p)/k$. This completes the description of the blame graph.

Assigning blame. To prove Theorem 3, we use Lemma 1 and an assignment of blame analogous to the one used in Section 2.1. For each packet p in $OPT(I) - ALG(I)$, if there is a blue node $\langle p, i \rangle$ in a tree with root $\langle r, i' \rangle$, then all the blame for packet p is assigned to the packet r. If there are two or more blue nodes for p then one is chosen arbitrarily. If there is no blue node for p, then there must be one or more red nodes for p, $\langle p, q_1 \rangle \ldots \langle p, q_k \rangle$, with roots $\langle r_1, i_1 \rangle \ldots \langle r_k, i_k \rangle$. The blame for packet p is assigned to the nodes $r_1 \ldots r_k$, where the amount of blame received by r_ℓ is equal to the weight of $\langle p, q_\ell \rangle$. Thus, the total blame assigned for each packet in $OPT(I) - ALG(I)$ is equal to the weight of the packet, and the total blame assigned to each packet in $ALG(I)$ is at most the weight of the blue and red nodes in the tree rooted at the green node corresponding to that packet. Therefore, to prove Theorem 3, it suffices to place a suitable upper bound on the total weight of the blue and red nodes in the blame graph. We now do this in Claims 4 and 5. Let $H(T)$ denote the harmonic sum $\sum_{i=1}^{T(I)} 1/i$.

Claim 4 *The total weight of the red children of any node $\langle p, i \rangle$ is at most $16H(T)w(x)$.*

Proof: Let S_k denote the set of packets q of slack at most k such that $\langle q, p\rangle$ is a red child of $\langle p, i\rangle$. Consider a packet q in S_k. It follows that there exists an up-tree $U(j)$ such that the routing of q along $U(j)$ conflicts with the routing of p along $U(i)$. Moreover, following the reasoning of Claim 1, we can show that a conflict occurs either on the last rootward edge traversed by p or on the first leafward edge traversed by p.

Let e_1 and e_2 be the last rootward and the first leafward edges, respectively of p's path. Let t be the time step at which p will cross its last rootward edge if routed along $U(i)$. For any q in S_k, the aforementioned conflict with p implies that there is an eligible bufferless routing of q such that q crosses either e_1 at time t or e_2 at time $t+1$. Since q has slack at most k, it follows that in the optimal buffered solution, q must either traverse e_1 in the time interval $[t-k+1, t+k-1]$ or e_2 in the time interval $[t-k+2, t+k]$. Since only one packet can cross any edge at any time in the optimal schedule, it follows that there are at most $4k$ packets ($2k$ for the last rootward edge and $2k$ for the first leafward edge) in S_k. Since the weight of any red node $\langle p, q\rangle$ with slack exactly k is at most $4w(p)/k$, the total weight of the red children of $\langle p, i\rangle$ is at most:

$$\sum_{k=1}^{T(I)}(|S_k|-|S_{k-1}|)\frac{4w(p)}{k} \leq \sum_{k=1}^{T(I)}\frac{16w(p)}{k} = 16H(T(I))w(p).$$

■

Since the definition of the blue nodes has not changed, Claim 2 still holds, and thus we can prove the following.

Claim 5 *For any node x in a blame tree, the total weight of the subtrees rooted at the blue children of x is at most $17w(x)H(T)$.*

Proof: As in the proof of Claim 3, we apply an induction on the maximum distance of the node x from any leaf. Clearly the claim is true when x is a leaf. For the inductive hypothesis, we assume that the claim is true for all of the children of x. The total weight of the blue children of x is at most $w(x)/2$. Thus, by the inductive hypothesis, the total weight of the subtrees rooted at all of the blue grandchildren of x is at most $17w(x)H(T)/2$. By Claim 4, the total weight of the red children of the blue children of x is at most $8w(x)H(T)$. Adding the three kinds of weights together, we obtain $17w(x)H(T)/2 + 8w(x)H(T) + w(x)/2 \leq 17w(x)H(T)$. ■

Claims 4 and 5 imply that the total blame assigned to each packet $p \in ALG(I)$ is at most $33H(T)w(p)$. Since the total blame assigned for each packet in $I - ALG(I)$ is equal to the weight of the packet, Theorem 3 follows directly from Lemma 1. ■

3 Dimension order routing on the mesh

In this section, we consider the problem of dimension order routing on the mesh subject to deadline and capacity constraints. In dimension order routing, every packet is routed first along the row edges to the correct column and then routed along the column edges to the correct row. We present a dimension order routing algorithm **WM**, which achieves a constant factor approximation ratio for the bufferless case. We also analyze the performance of this algorithm for the buffered case. In both scenarios, with only a factor of 2 penalty in the performance ratio, the dimension order routing algorithm can be converted into an algorithm for routing on the mesh where the packets are allowed to use either dimension order paths or reverse dimension order paths, i.e., paths that travel first along a column and then along a row. To do so, we run the algorithm first using dimension order paths, and then on the reverse dimension order paths, and take the better of the two solutions. All of the results described for the mesh can also be extended to higher dimensional meshes, where the dependence on the dimension d imposes an additional factor of $d \cdot 2^d$ in each of the results.

The basic idea underlying **WM** is the same as that underlying **WT**. The main technical difficulty to overcome is the fact that on the mesh, there no longer are clear "rootward" and "leafward" directions. For the tree, we defined the notion of up-trees that capture the rootward portions of the paths in any bufferless schedule. Similarly, we introduce the notions of *lr-mesh* and *rl-mesh*. An lr-mesh is a copy of the set of the row edges of the given mesh with a label $t(e)$ on each edge e that satisfies the following property: if $d(e)$ denotes the number of edges between e and the rightmost node on the row in which e lies, then the quantity $d(e)+t(e)$ is the same for all row edges. Similarly, an rl-mesh is a copy of the set of the row edges of the given mesh with a label $t(e)$ on each edge e that satisfies the following property: if $d(e)$ denotes the number of edges between e and the leftmost node on the row in which e lies, then the quantity $d(e)+t(e)$ is the same for all row edges.

The definitions of an lr-mesh and an rl-mesh are motivated by the following observation: for any packet that is dimension order routed in a bufferless schedule, there exists a unique lr-mesh or rl-mesh in which the label of every row edge of the packet's path equals the time step at which the packet crosses that edge in the schedule. If the row edges of the path are traversed from left to right, then the associated structure is an lr-mesh; otherwise, it is an rl-mesh.

We sort the set of all lr-meshes and the set of all rl-meshes separately, in terms of their value of $d(e)+t(e)$, where we say that the lr-mesh (resp., rl-mesh) with the smallest value of $d(e)+t(e)$ is the *earliest* lr-mesh (resp., rl-mesh) and the one with the largest value of $d(e)+t(e)$ is the *latest* lr-mesh (resp., rl-mesh). For each packet, the release time and deadline constraints for that packet define a set of lr-meshes or rl-meshes such that the packet obeys its constraints if and only if it is routed on one of the lr-meshes or rl-meshes in this set. The lr-meshes (resp., rl-meshes) in this set form a contiguous set in the sorted order of lr-meshes (resp., rl-meshes) from earliest to latest. We say that the packet is *eligible* to be routed on these lr-meshes (resp., rl-meshes). We also assign packets that only travel in a vertical direction to lr-meshes. As in the tree algorithm, we do this in such a way that a packet with e' as its first vertical edge will be associated with the same lr-mesh if it traverses edge e' at time t regardless of whether it has a left to right component.

Algorithm **WM**:

Let S be the empty set.
Let N be the set of all packets.
Let $L(1) \ldots L(k)$ be the set of all possible lr-meshes in order from latest to earliest.
Let $R(1) \ldots R(k)$ be the set of all possible rl-meshes in order from latest to earliest.
For $i = 1$ to k;
  Let $E(i)$ (resp., $F(i)$) be the set of packets in $N - S$ eligible for routing on $L(i)$ (resp., $R(i)$).
  Prune light eligible packets.
    For every packet p in $E(i)$:
      Let S' be the set of packets in S that would conflict with p if p were routed on $L(i)$.
      If $w(S') \geq w(p)/4$ remove p from $E(i)$.
    Repeat the above for-loop with $E(i)$ and $L(i)$ replaced by $F(i)$ and $R(i)$, respectively.
  Select new packets.
    Let $S(i)$ be the empty set.
    Repeat until $E(i)$ is empty.
      Let p be the packet in $E(i)$ for which the rightmost horizontal edge that is leftmost.
      Remove p from $E(i)$.
      Let $S'(i)$ be the set of packets in $S(i)$ that conflict with p if p were routed on $L(i)$.
      If $w(S'(i)) < w(p)/4$ remove all packets in $S'(i)$ from $S(i)$ and add p to $S(i)$.
    Repeat the above repeat-loop with $E(i)$, $L(i)$, and the phrase "rightmost" replaced by $F(i)$, $R(i)$, and "leftmost", respectively.
    Assign the packets in $S(i)$ to $L(i)$ or $R(i)$ as appropriate.
  Purge conflicting packets.
    Remove any packets in S which conflict with packets in $S(i)$.
  Add the packets in $S(i)$ to S.
Return S.

As is evident from the definition of the algorithm, the basic approach of **WM** is the same as the approach of **WT**. The primary difference is that while **WM** considers two different classes of mesh subgraphs, the lr-meshes and the rl-meshes , **WT** considers only one class of up-trees. However, we can still use the same analysis framework developed in Section 2 and the difference between the two algorithms affects only a small portion of the analysis. Therefore, we only present an outline of our proof for the mesh, indicating the primary differences between the analyses.

For any instance I, let $OPT(I)$ and $ALG(I)$ denote the total weight of the optimal bufferless solution and the solution computed by **WM** respectively. Our main result for bufferless dimension order routing on the mesh is the following.

Theorem 4 *For any weighted problem instance I for the mesh, the total weight of the packets routed by algorithm* **WM** *is $\Omega(w(OPT(I)))$.*

Proof: The main idea behind the proof of Theorem 4, as in Section 2, is to assign blame for each packet in $OPT(I) - ALG(I)$. We use the framework provided by Lemma 1 and define a blame graph consisting of red, blue, and green nodes. The definitions of the nodes and edges in the blame graph are exactly as those for the tree (see Section 2.1). Also, the blame assigned for each packet in $OPT(I) - ALG(I)$ is determined from the blame graph in exactly the same way as for the tree.

The remainder of the analysis proceeds exactly as in the case of the tree except for the proof of Claim 1, which states that the total weight of the red children of any node x is at most $8w(x)$. We now sketch the proof of Claim 1 for the mesh. Consider a blue or green node $\langle p, i\rangle$ with a red child $\langle q, p\rangle$. It follows from the definition of a red child that the packet q is selected in the optimal solution. Let j be such that q is either routed along $L(j)$ or $R(j)$. Without loss of generality, we may assume that q is routed along $L(j)$ in the optimal solution. As in the case of the tree, we now consider two cases depending on the phase in which packet q is removed in iteration j of **WM**. If q is removed in the pruning phase, then $i < j$. We invoke the same argument as is used for the tree to prove that the routing of packet q along $L(j)$ conflicts with a routing of p along $L(i)$ or $R(i)$, as the case may be, on the first vertical edge traversed by p.

The case in which q is discarded in the selection phase has to be treated somewhat differently from the analysis for the tree. As in the case of the tree, this case happens only if $i = j$. If the routing of p is along $L(i)$, then we can invoke the same argument about the particular ordering according to which the packets are considered in the selection phase to show that the routing of q along $L(i)$ conflicts with the routing of p along $L(i)$ on either the rightmost horizontal edge or the first vertical edge of p's path. If, instead, the routing of p is along $R(i)$, then since we assume that all of the edges are bidirectional, a routing of p along $R(i)$ and a routing of q along $L(i)$ can conflict only along the first vertical edge of p. Thus Claim 1 is proved. ■

We can also show that **WM** provides us with a 3-approximation for the case where all packets have the same weight. Furthermore, we can also place an upper bound on the approximation ratio of **WM** with respect to buffered schedules using an analysis similar to the one given in Section 2.2. Let $T(I)$ denote the minimum of the maximum slack in instance I and the number of packets in $OPT(I)$. The proof of the following theorem is deferred to the full version of the paper.

Theorem 5 *For any problem instance I for the mesh, the total weight of the packets in the optimal buffered schedule is $O(w(OPT(I)) \cdot \log T(I))$.*

4 Routing weighted packets on the linear array

Since a linear array is a special case of the tree, the algorithm of Section 2 also provides constant factor and logarithmic approximations for bufferless routing and buffered routing, respectively, for the linear array. The constants involved in the results, however, are significantly larger than the constants established in the results for routing unweighted packets on the linear array [1]. In this section, we prove that the approximation ratios that have been achieved for unweighted packets on the linear array can also be achieved for the problem of routing weighted packets on the linear array. In addition, for the case of the linear array, we can show an upper bound on the ratio between the optimal buffered solution and the optimal bufferless solution in terms of the maximum span of the packets. This gives us a stronger result for the buffered case of weighted packets on the linear array.

4.1 Bufferless scheduling

Our algorithm **WA** for bufferless scheduling is analogous to the algorithm of [1] for unweighted packets. Before we present the algorithm, we review some notation.

Let the n nodes of the array be numbered 1 through n from left to right. Since the edges are bidirectional, we will assume that all of the packets are moving left to right on the array. As in [1], we define a *scan line*. A scan line is a copy of the original linear array in which each edge $(i, i+1)$ is given a label $t(i)$ such that the sum $t(i) - i$ is a constant over all i. The primary motivation behind the concept of a scan line is the following: for any packet that is routed in a bufferless schedule, there exists a unique scan line in which the label of every edge of the packet's path equals the time step at which the packet crosses that edge in the schedule.

Let L denote the set of scan lines. For a given scan line, any packet p that is eligible to be routed along ℓ defines a *segment* p_ℓ in ℓ corresponding to the path taken by p. If the scan line ℓ is clear from the context, we will drop the subscript ℓ from p_ℓ and thus identify the packet and the associated segment.

Algorithm **WA**:

Let S be the empty set.
Let M be the set of all packets.
For each scan line ℓ;
  Let M_ℓ be the set of packets remaining in M eligible to be routed along ℓ.
  Find a maximum-weight set $N_\ell \subseteq M_\ell$ of packets whose segments with respect to ℓ do not intersect.
  Add N_ℓ to S and set M to $M - N_\ell$.
Return S.

A crucial step in the processing of every scan line in algorithm **WA** is the computation of a maximum-weight set of nonintersecting segments from a given collection of segments. This computation can be done in polynomial time by a simple dynamic programming algorithm.

We now prove that **WA** is a 2-approximation algorithm. For a given set X of packets, let $w(X)$ denote the sum of the weights of the packets in X. For problem instant I, let $ALG(I)$ and $OPT(I)$ denote the total weight of the packets selected by **WA** and in the optimal bufferless schedule, respectively.

Theorem 6 *For any I, $w(ALG(I)) \geq w(OPT(I))/2$.*

Proof: Let $OPT_\ell(I)$ denote the subset of packets in $OPT(I) - S$ that are routed along scan line ℓ. Suppose $w(ALG(I)) < w(OPT(I))/2$. Then for some ℓ, $OPT_\ell(I) > N_\ell$, contradicting the optimality of N_ℓ. ∎

4.2 Buffered scheduling

In this section, we extend a result of [1] to establish that **WA** serves as a useful buffered approximation algorithm as well. In particular, for every input instance I, we compare the performance of the optimal bufferless solution, $OPT(I)$, with that of the optimal buffered solution $OPT_B(I)$. As in [1], we express the comparison in terms of the parameter $\Lambda(I) = \min\{\sigma(I), \delta(I), |I|\}$, where $\sigma(I)$ is the maximum slack in I and $\delta(I)$ is the maximum span in I.

Theorem 7 *For any problem instance I with weighted packets, we have*

$$w(OPT_B(I)) \leq 4(\log(\Lambda(I)) + 1) \cdot w(ALG(I)).$$

Proof: We will show that $w(OPT_B(I)) \leq 4(\lceil \log x \rceil + 1) \cdot w(ALG(I))$ for each $x \in \{\sigma(I), \delta(I), |I|\}$. We start with $x = \sigma(I)$. Divide the set I into $\lceil \log(\sigma(I)) \rceil$ sets such that the ith set I_i consists of the packets in I that have slack at least 2^i and less than $2^{i+1} - 1$. Let $OPT_B^i(I)$ denote the set $OPT_B(I) \cap I_i$. We now prove that $w(OPT_B^i(I) - ALG(I))$ is at most $4w(ALG(I))$.

Consider any scan line ℓ. Let X_ℓ be the set of packets in $OPT_B^i(I) - ALG(I)$ that are relevant to ℓ. Since the slack of each packet in $OPT_B^i(I)$ is less than 2^{i+1}, the number of packets in X_ℓ that intersect a given point on the scan line is less than 2^{i+2}. Thus the set of packets X_ℓ can be partitioned into 2^{i+2} groups such that each group forms a set of nonintersecting segments on the scan line ℓ. Since $X_\ell \subseteq M_\ell$ and N_ℓ is a set of packets corresponding to a maximum-weight set of nonintersecting segments on ℓ, we have the following inequality for each scan line ℓ:

$$w(X_\ell) < 2^{i+2} w(N_\ell). \tag{1}$$

Since each packet in $OPT_B^i(I) - ALG(I)$ contributes to at least 2^i scan lines, we obtain the following inequality on adding Equation 1 over all scan lines.

$$w(OPT_B^i(I) - ALG(I)) \leq 4w(ALG(I)). \tag{2}$$

We now prove the desired claim as follows.

$$\begin{aligned} & w(OPT_B(I) - ALG(I)) \\ \leq \; & \sum_i w(OPT_B^i(I) - ALG(I)) \\ \leq \; & 4\lceil \log \sigma(I) \rceil w(ALG(I)) \end{aligned}$$

We next show it for $x = \delta(I)$. Divide the set I into $\lceil \log(\delta(I)) \rceil$ sets such that the ith set I_i consists of the packets in I that have span at least 2^i and less than

$2^{i+1}-1$. Let $OPT_B^i(I)$ denote $OPT_B(I) \cap I_i$. We now prove that $w(OPT_B^i(I) - ALG(I))$ is at most $4w(ALG(I))$.

Consider any scan line ℓ. Let X_ℓ be the set of packets in $OPT_B^i(I) - ALG(I)$ that are partially routed along scan line ℓ in the optimal solution. Let Y_ℓ denote the multiset of packets in which each packet in X_ℓ occurs a number of times equal to the number of edges traversed by the packet along ℓ in the optimal solution. Since the span of each packet in $OPT_B^i(I)$ is less than 2^{i+1}, the number of packets in Y_ℓ that intersect a given point on the scan line is less than 2^{i+2}. Thus the multiset Y_ℓ can be partitioned into 2^{i+2} groups such that each group forms a set of nonintersecting segments on the scan line ℓ. Since $X_\ell \subseteq M_\ell$ and N_ℓ is a set of packets corresponding to a maximum-weight set of nonintersecting segments on ℓ, we obtain the following inequality.

$$w(Y_\ell) < 2^{i+2} w(N_\ell). \tag{3}$$

Since each packet in $OPT_B^i(I) - ALG(I)$ has a multiplicity of at least 2^i in the union of the multisets Y_ℓ, we obtain the following inequality on adding Equation 3 over all scan lines: $w(OPT_B^i(I) - ALG(I)) \leq 4w(ALG(I))$. We now prove the desired claim as follows.

$$\begin{array}{ll} & w(OPT_B(I)) \\ \leq & \sum_i w(OPT_B^i(I) - ALG(I)) + w(ALG(I)) \\ \leq & (4\lceil \log \delta(I) \rceil + 1) w(ALG(I)). \end{array}$$

Finally, consider the case $x = |I|$. We first observe that **WA** routes all of the packets that have slack at least $|I|+1$. Therefore, it remains to consider only those packets in $OPT(I)$ that have slack at most I. By our earlier argument concerning $\delta(I)$, it follows that $w(OPT(I))$ is at most $4(\lceil \log |I| \rceil + 1)$. ∎

5 Hardness of off-line approximations

In this section, we establish a lower bound on the approximability of the time-constrained routing problem. All of the results described involve only packets with no slack, and thus apply both to the buffered and bufferless case. Furthermore, the lower bound applies to the special case where all packets have the same weight.

We first provide strong evidence that there does not exist a polynomial-time $n^{1-\epsilon}$-approximation, for any $\epsilon > 0$, for our problem with n packets on arbitrary topologies. For this result, all paths are shortest paths. We then show that if the specified paths are not required to be shortest paths, then the preceding lower bound on the approximation ratio holds even when the underlying graph is a mesh. This result uses specified paths that are not dimension order paths, and thus complements the constant-factor and logarithmic-factor upper bounds established in Section 3 under the assumption that all routing is along dimension order paths.

Theorem 8 *There is a family graphs $\mathcal{G} = G_1, G_2, \ldots,$ such that for any $\epsilon > 0$, there is no polynomial-time $n^{1-\epsilon}$-approximation algorithm for the time-constrained routing problem with n packets on G_n, unless NP=ZPP.*

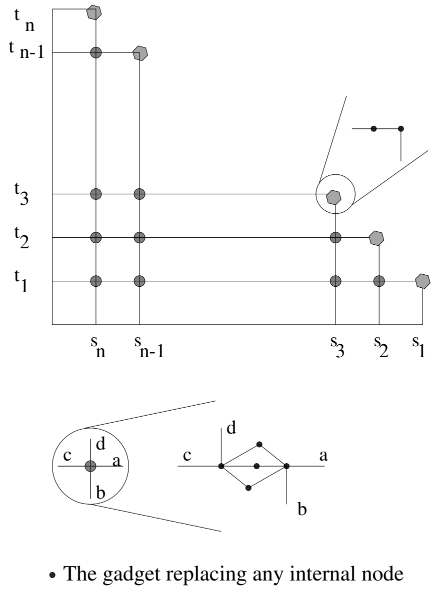
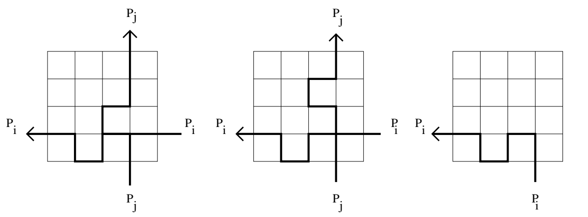


Figure 1: Graph used in the proof of Theorem 8

Proof: We give a reduction from the independent set problem which is known to be $n^{1-\epsilon}$-hard to approximate, unless $NP = ZPP$ [8]. Our starting point is input $G = (V, E)$ to the independent set problem; let n denote $|V|$. We use a *mesh-like graph*, and specify packets and routes through a reduction from G; see Figure 1 below. Note that the same mesh-like graph will be used for every input graph G to the independent set problem, with $n = |V|$. Our construction here is similar to the one used in [7] for showing the hardness of bounded delay disjoint paths problem.

For each $i \in [1..n]$, the packet p_i with source s_i and destination t_i is included in the problem instance. In other words, there is a packet for each vertex in G. Assume the nodes of the mesh are labeled as in the Cartesian plane with s_1 located at $(0,1)$ and t_n at $(n, n+1)$. Then the path π_i that packet p_i must follow is given by the sequence $s_i = (0,i), (1,i), ..., (i,i), (i,i+1), ..., t_i = (i, n+1)$. It is easy to see that for any $i \neq j$, the paths π_i and π_j intersect exactly once, at location (i,j). To complete our construction, we complete the description of the mesh-like graph. Specifically, we replace each mesh node by a suitable gadget graph. Each gadget graph has the property that the path of a packet will take exactly two units of time in traversing through it. All nodes along the diagonal of the mesh are simply replaced by a path of length two. All other nodes of the form (i,j) inside the mesh are replaced by the gadget as shown in Figure 1. Now, to complete the specification of the path of any packet, we do the following: If $(i,j) \in E$, π_i and π_j are required to use the central portion of the gadget, so that they share the same length-two path while passing through the node (i,j). If $(i,j) \notin E$ then these paths use different portions of the gadget, and thus have no shared edges in their paths. It is easy to verify that each path π_i has length $(3n+1)$. Finally, we set the release time r_i of the ith packet to be $3(i-1)$ and the deadline d_i to be $r_i + (3n+1)$ (i.e. no slack is given).



If i and j are adjacent in G   If i and j are non-ajacent in G   The diagonal gadget

Figure 2: Gadgets used in the proof of Theorem 9

Our main claim now is that for any $i < j$, packets starting at s_i and s_j at their respective release times, arrive at the node (i, j) simultaneously. To see this, notice that the packet from s_i arrives at (i, j) at time $r_i + 3j - 2 = 3i + 3j - 5$. The arrival time of the packet from s_j on the other hand is given by $r_j + 3i - 2 = 3i + 3j - 5$.

It follows from our construction above that if $(i, j) \in E$, then any such pair of packets must collide while passing through (i, j). Therefore, any subset of packets chosen to be routed must correspond to an independent set in G. Moreover, it is easy to see that every independent set in G corresponds to a set of packets that can be routed without conflicts. The claimed result now follows from the hardness of the independent set problem. ∎

Theorem 9 *For any $\epsilon > 0$, there is no $n^{1-\epsilon}$-approximate algorithm for our problem even when the underlying graph is a mesh, unless NP=ZPP.*

Proof: The mesh-like graph constructed for Theorem 8 loses its mesh structure once we replace the nodes with the gadgets. This can be handled by suitably modifying our gadget graphs as shown below in Figure 2. The rest of the proof is similar to Theorem 8; we defer the details to the full version of the paper. ∎

6 On-Line Lower Bound for Trees

For the unweighted case of linear arrays, [1] provided a distributed and on-line algorithm for the buffered case that is guaranteed to achieve within a factor of 2 of the optimal (off-line and centralized) bufferless schedule. We here show that the analogous result is not possible in the case of a tree, even when all packets have the same weight. We derive this lower bound by presenting a specific tree network and a sequence of packets for which any (deterministic) on-line algorithm would be able to schedule at most one packet, while an off-line algorithm would be able to schedule h packets, where h is the height of the tree. In this sequence of packets, each packet has zero slack, and thus the buffered and bufferless cases are identical. Furthermore, our lower bound applies to the class of "pre-emptive" on-line algorithms, algorithms that are allowed to drop a packet that has already been scheduled.

Theorem 10 *The competitive ratio of any on-line algorithm for time-constrained routing on an n-node tree is $\Omega(\log n)$.*

The lower bound tree. For any nonnegative integer i, let $T(i)$ denote a complete binary tree of height i. The tree T employed in our lower bound consists of a root r with a single child that forms the root of the complete binary tree $T(h)$. Thus, for each i in the range $0 \le i \le h$, every node of T at height i is the root of a tree $T(i)$. The lower bound tree is illustrated in Figure 3(a).

The set of packets. The lower bound instance consists of $2h$ packets that are determined according to the actions of the given on-line algorithm OL. For each time step i in the range $0 \le i < h$, we introduce two new packets. Each packet is destined to a leaf node and has deadline time h. It thus follows that every packet has zero slack. Since all packets are traveling in a leafward direction, it also follows that if the paths of two packets intersect along an edge, then at most one of them can be scheduled in any valid schedule. While presenting the lower bound instance, we will also develop the argument that leads to the lower bound of h on the competitive ratio. In particular, we show that any on-line algorithm OL can schedule at most one of the $2h$ packets, while there exists an off-line schedule that schedules at least one packet among the two introduced in each of the h time steps.

At time step 0, we introduce two new packets p_0 and q_0. The source of each packet is the root r of T and the destinations are the leftmost and the rightmost leaves, respectively, of the tree. Since the packets p_0 and q_0 collide on the sole edge leading from r, only one of the packets may be scheduled in any valid schedule. We let the off-line schedule select the packet that is not selected by the on-line algorithm OL; if OL does not schedule any of the packets, the off-line schedule selects an arbitrary packet. Thus, at the start of time step 1, the following two properties hold. First, at most one packet p is scheduled by OL. Second, if p exists, then it is destined to either the leftmost leaf or the rightmost leaf of the subtree $T(h-1)$ rooted at the child of r. Furthermore, the particular leafward edge traversed by p while leaving the root of $T(h-1)$ is not traversed by the packet selected in the off-line schedule.

In general, at the start of time step i, we will maintain the following two invariants: (i) at most one packet p has been scheduled by OL, and (ii) if p exists, then it is destined to either the leftmost leaf or the rightmost leaf of a subtree $T(h-i)$, and the particular leafward edge traversed by p while leaving the root of this subtree $T(h-i)$ is not traversed by any of the packets selected in the off-line schedule. At time step i, we introduce two new packets p_i and q_i. If p exists, then without loss of generality we may assume that p is destined to the leftmost leaf of $T(h-i)$. We set the source of both p_i and q_i to be the root of $T(h-i)$ and the destinations to be the leftmost and rightmost leaves, respectively of the left subtree of $T(h-i)$. Figure 3(b) illustrates the packets p, q, and p_i. It follows that the three packets p, p_i, and q_i all collide on the edge between the root of $T(h-i)$ and its left child. Therefore, the on-line algorithm can schedule at most one of the three packets. If the on-line algorithm continues to schedule p or drops p in favor of p_i, then we include q_i in the off-line schedule; otherwise, we include p_i in the off-line schedule.

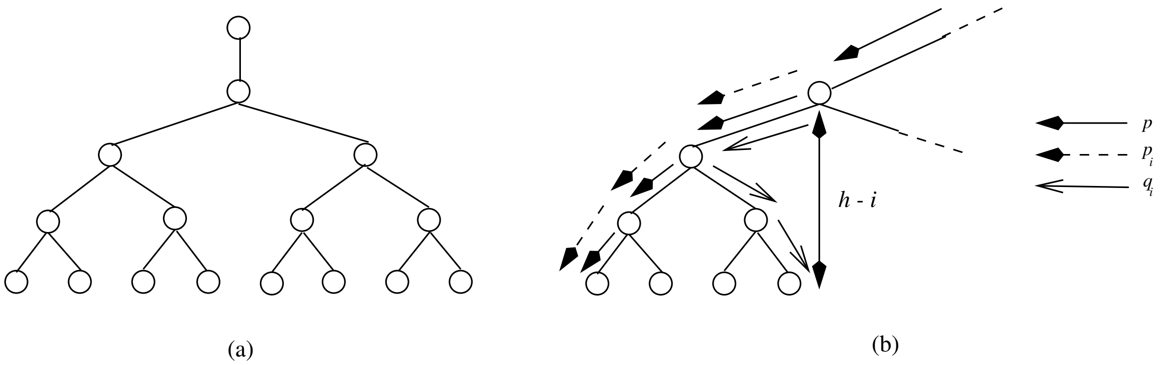


Figure 3: (a) Lower bound tree for $h = 4$; (b) Time step i: p is a packet currently scheduled by the on-line algorithm, p_i and q_i are new packets injected at time step i.

We now verify that the two invariants hold at the start of time step $i+1$. Clearly invariant (i) holds since, as mentioned above, all of the three packets currently available for scheduling, i.e., p, p_i, and q_i, collide on an edge. For the first part of invariant (ii), we note that p and p_i are destined to the leftmost leaf of a subtree $T(h-i-1)$, while q_i is destined to the rightmost leaf of this subtree. The second part of invariant (ii) follows directly from invariant (ii) of time step i and the choice made in the off-line schedule at time step i.

We note that in the above construction, exactly one of packets p_i and q_i, for each i, is included in the off-line schedule. Thus, the total number of packets in the off-line schedule is h. On the other hand, it follows from invariant (i) at the start of time step h that the on-line algorithm schedules at most one packet. Therefore, the competitive ratio of the on-line algorithm is at least h. This completes the proof of Theorem 10.

References

[1] M. Adler, A.L. Rosenberg, R.K. Sitaraman, and W. Unger (1998), "Scheduling time-constrained communication in linear networks". *10th ACM Symposium on Parallel Algorithms and Architectures* 269-278.

[2] M. Andrews, A. Fernandez, M. Harchol-Balter, F.T. Leighton, L. Zhang (1997): General dynamic routing with per-packet delay guarantees of O(distance + 1/session rate). *38th IEEE Symp. on Foundations of Computer Science*.

[3] M. Andrews and L. Zhang (1999): Packet routing with arbitrary end-to-end delay requirements. *31st ACM Symp. on Theory of Computing*.

[4] S.N. Bhatt, G. Bilardi, G. Pucci, A.G. Ranade, A.L. Rosenberg, E.J. Schwabe (1996): On bufferless routing of variable-length messages in leveled networks. *IEEE Trans. Comp. 45*, 714-729.

[5] R. Games, A. Kevsky, P. Krupp, L. Monk (1995): Real-time communications scheduling for massively parallel processors. *Real-Time Technology and Applications Symp.*, 76-85.

[6] S.J. Golestani (1991): Congestion-free communication in high-speed packet networks. *IEEE Trans. Communications 39*, 1802-1812.

[7] V. Guruswami, S. Khanna, B. Shepherd, R. Rajaraman and M. Yannakakis (1999). Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. To appear in *31st ACM Symp. on Theory of Computing*.

[8] J. HÅSTAD. Clique is hard to approximate within $n^{1-\epsilon}$ (1996). *37th IEEE Symp. on Foundations of Computer Science*, 627–636.

[9] J.H. Kim and A.A. Chien (1996): Rotating Combined Queuing (RCQ): Bandwidth and latency guarantees in low-cost, high-performance networks. *23rd Intl. Symp. on Computer Architecture*, 226-236.

[10] C. Lam, H. Jiang, V.C. Hamacher (1995): Design and analysis of hierarchical ring networks for shared-memory multiprocessors. *Intl. Conf. on Parallel Processing*, I:46-50.

[11] F.T. Leighton (1992): Methods for message routing in parallel machines (invited survey). *24th ACM Symp. on Theory of Computing*.

[12] F.T. Leighton (1992): *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Morgan Kaufmann, San Mateo, CA.

[13] J. Liebeherr (1995): Multimedia networks: issues and challenges. *IEEE Computer 28* (4) 68-69.

[14] J.-P. Li and M.W. Mutka (1994): Priority based real-time communication for large scale wormhole networks. *Intl. Parallel Proc. Symp.*, 433-438.

[15] K.-S. Lui and S. Zaks (1998): Scheduling in synchronous networks and the greedy algorithm. *Theoretical Comp. Sci.*

[16] M.W. Mutka (1994): Using rate monotonic scheduling technology for real-time communications in a wormhole network. *Wkshp. on Parallel and Distr. Real-Time Computing Systs. and Applications.*

[17] A.K. Parekh and R.G. Gallager (1993): A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Networking 1*, 344-357.

[18] A.K. Parekh and R.G. Gallager (1994): A generalized processor sharing approach to flow control in integrated services networks: the multiple-node case. *IEEE/ACM Trans. Networking 2*, 137-150.

[19] A. Pietracaprina and F.P. Preparata (1995): Bufferless packet routing in high-speed networks. Typescript, Brown Univ.

[20] J. Rexford, J. Hall, K.G. Shin. (1996): A router architecture for real-time point-to-point networks. *23rd Intl. Symp. Computer Architecture.*

[21] A. Saha (1995): Simulator for real-time parallel processing architectures. *IEEE Ann. Simulation Symp.*, 74-83.

[22] L.R. Welch and K. Toda (1994): Architectural support for real-time systems: issues and trade-offs. *Intl. Wkshp. on Real-Time Computing Systs. and Applications.*

[23] L. Zhang (1990): Virtual clock: A new traffic control algorithm for packet switching networks. *ACM SIGCOMM*, 19-29.

[24] W. Zhao, J.A. Stankovic, K. Ramamritham (1990): A window protocol for transmission of time-constrained messages. *IEEE Trans. Computers 39*, 1186-1203.