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 transmitted 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 schedule the maximum number of packets subject to deadline constraints. This problem is studied in [1], where it is shown that the problem is NP-Complete even when the underlying topology is a linear array. Approximation algorithms are 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 2-dimensional 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 scheduled subject to their timing constraints. For the bufferless case, we provide constant factor approximation algorithms for the time-constrained scheduling problem with weighted packets on trees and meshes. We also provide logarithmic approximations 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 k packets are required to follow prescribed paths in an arbitrary graph, then unless NP = ZPP, there is no polynomial-time $k^{1-\epsilon}$-approximation, for any $\epsilon > 0$, to the optimal set of packets that can be scheduled.

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 [16]. For example, a real-time video packet

*A preliminary version of this paper appeared in the proceedings of SPAA 99.

†Department of Computer Science, University of Massachusetts, Amherst, MA 01002, USA. Email: micah@cs.umass.edu. Portions of this research were done while this author was at the Department of Computer Science, University of Toronto.

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. E-mail: sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship.

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

¶Department of Computer Science, University of Toronto, Toronto, Canada. Current address: Department of Computer Science, Technion, Haifa 32000, Israel. Email: adiro@cs.technion.ac.il.

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 [25].

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 notion 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 are 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 or 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 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 [13], has been studied in [5, 23]. Furthermore, as is shown 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 k 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, 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. The solution has to also include a valid schedule for the packets in M'.

Summary of Results. The first results in this framework are 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 it is shown in [1] 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; otherwise it is a 10-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, the maximum slack of any packet in the optimal buffered solution, and the size of the tree. 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. Dimension order routing, which is commonly used 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 on general topologies, the time-constrained routing problem is hard to approximate to within $\Omega(k^{1-\epsilon})$ for any $\epsilon > 0$, unless NP=ZPP; here k denotes the total number of packets. Furthermore, this result can be extended to the mesh, provided that the paths defined for the packets are not necessarily the shortest paths. These results apply to both buffered and bufferless routing, and taken together, they strongly motivate our concentration on specific network topologies of practical interest such as the tree and the mesh, 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] provides 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 also demonstrate that the analogous result is not possible in the case of a tree, even when all packets have the same weight. We give a lower bound showing that for any on-line deterministic algorithm for the problem, the competitive ratio is $\Omega(\log n)$, where n is the size of the tree. This lower bound is achieved on trees of depth $\Theta(\log n)$. These results appear in Section 6.

Related Work. In [1], the unweighted case of the time-constrained routing problem on the linear array topology is studied. 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.

Also, [19] 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 [21, 22, 3]. 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 [4], sessions with *delay requirements* are considered, so that packets of a given session request to be routed with a delay no larger than the requirement. In [4], a distributed packet scheduling algorithm is described that has the following property: 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 studies have examined routing with timing constraints. In [25] a router architecture is given for messages with deadlines. A minimum-laxity-first protocol is proposed in [29] for transmitting messages with deadlines in a multi-access shared-bus network. Some scheduling policies, such as Virtual Clock [28], Stop-and-Go [8], Rotating Combined Queuing [12], do not explicitly use message deadlines, but just attempt to keep the worst-case message delay small and bounded. Other relevant experimental work includes [20, 27, 18, 26, 7].

Of course, routing without timing constraints has been the subject of a large number of works; see [14, 15] for a survey. These studies 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 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 of 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 a logarithmic 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 rootward path of the packet 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 schedule for such a packet with an up-tree U, such that if e' is the first leafward edge from the source of the packet, and it traverses it at time t, 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 and traverses it at time t will be associated with the same up-tree 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 $U(i)$.
    Repeat until $E(i)$ is empty.
        Let p be a packet in $E(i)$ with the last rootward edge that is furthest from the root.
        Remove p from $E(i)$.
        Let S' be the set of packets in S that conflict with p if p were routed on $U(i)$.
        If $w(S') < w(p)/2$
            Remove all packets in S' from S and add p to S.
            Assign the packet p to $U(i)$.
            Remove the packet p from N.
Return S.

As described above, **WT** sequentially processes all of the up-trees in order from the latest to the earliest. Each iteration may add new packets to and may 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. Within each iteration the packets that are eligible for the up tree, and have not been scheduled, are considered in a specific order. Observe that once a packet is scheduled, it is never considered again, even if it is later deleted from the solution. We show below that the total weight of the solution returned by **WT** is within a constant factor of the weight of the optimal solution. 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. We prove the following theorem.

Theorem 1 *For any weighted problem instance I for the tree, $w(ALG(I)) \geq w(OPT(I))/10$.*

For the purpose of the proof we will define a *blame graph*, which will formulate a "blame" for each packet in $OPT(I) - ALG(I)$ in terms of packets in $ALG(I)$. 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. Every time **WT** schedules a packet p, a new node is added to the graph. This node is associate with the packet p, and is colored green at the time of its addition to the graph. We denote it $\langle p \rangle$. When p is scheduled and added to S, if some other packets are dropped from S, then the nodes corresponding to these packets change their color from green to blue. At such an event we also add an edge from each of the nodes that turned blue, to the new, green, node $\langle p \rangle$.

It follows that at any time, we have a green node for every packet in S, and a blue node for every packet that was previously in S, and was later dropped from S. In addition, for any blue node there is a single outgoing edge, while there is no outgoing edge from a green node. The graph is then a forest, consisting of trees with green roots, and all other nodes being blue. The edges in each tree represent, for each blue node, the packet to be "blamed" for its removal from S.

Red nodes. When the algorithm terminates we may have a number of packets in $OPT(I) - ALG(I)$ that do not have any corresponding node in the blame graph. For each such packet we add a single or several red nodes to the graph. Let q be a packet in $OPT(I) - ALG(I)$, and let $U(i)$ be the up-tree along which packet q is scheduled in the optimal solution $OPT(I)$. Since q has no blue or green node, it follows that it was never scheduled by the algorithm. In particular it was not scheduled when up-tree $U(i)$ was considered. It follows that at that time there was a set $S' \subseteq S$ of scheduled packets that conflicted with scheduling q on $U(i)$, such that $w(S') \geq w(q)/2$. Clearly for every $x \in S'$ there is a (green or blue) node in the blame graph since at the time $U(i)$ was considered there was a green node for every $x \in S'$. We now add a red node $\langle q, x \rangle$ for every $x \in S'$. We also add an edge from $\langle q, x \rangle$ to the node $\langle x \rangle$. Observe that after adding the red nodes the graph is still a forest, all the roots are still green, and all the red nodes are leaves.

Assigning weights. We now assign weights to the nodes in the blame graph. Any green or blue node $\langle p \rangle$ is assigned the weight $w(p)$ of the packet p. As to the red nodes, we group them according to the non-scheduled packet that they represent. That is, for every packet q in $OPT(I) - ALG(I)$ that was never scheduled we consider the set of nodes $\langle q, x \rangle$, for $x \in S'$, where S' is as defined above. Node $\langle q, x \rangle$ is assigned weight $w(q) \cdot \frac{w(x)}{w(S')}$. Observe that the total weight assigned to the nodes $\langle q, x \rangle$ for a given q is $w(q)$, and that the weight of a node $\langle q, x \rangle$ is at most $2w(x)$.

Note now that the weight of the optimal solution $OPT(I)$ is upper bounded by the total weight of all the nodes in the blame graph, and that the weight of the solution of **WT** is equal to the total weight of the green nodes. In the following claims and lemmas we will show that the total weight of the green nodes is at least one tenth of the total weight of all the nodes in the blame graph.

Claim 1 *The total weight of the red children of any (blue or green) node $\langle x \rangle$ is at most $4w(x)$.*

Proof: We have already seen that the weight of any red node is at most twice the weight of its parent. Thus we only need to show that every node has at most two red children.

Consider a blue or green node $\langle x \rangle$ with a red child, say $\langle q, x \rangle$. Let $U(i)$ be the up-tree along which packet x was scheduled by **WT** (when the blue or green node was added to the blame graph). Let e_1 be the last rootward edge traversed by x, and e_2 be the first leafward edge traversed by x, and let t_1 and t_2 be the times at which packet x traverses e_1 and e_2, respectively, when routed along $U(i)$. We will show that if a schedule for packet q conflicts with the schedule of x along $U(i)$, then either q traverses e_1 at t_1, or q traverses e_2 at t_2. Any red node represents a packet whose schedule according to the optimal solution conflicts with the schedule selected by **WT** for its parent. Since in the optimal solution only one packet can traverse any edge at any given time, we have that the node corresponding to x can have at most two red children.

Let $U(j)$ be the up-tree along which a packet q is routed in the optimal schedule. We know that in iteration j packet q was considered for scheduling on $U(j)$, but was rejected because of a set S' which includes packet x, scheduled on $U(i)$. This means that the schedule of x on $U(i)$ conflicts with the schedule of q on $U(j)$. We now distinguish between two cases. The first one is when $i = j$, and the second one is when $i \neq j$.

If $i = j$, then by the ordering in which we consider the packets in iteration i, the packet x 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 x along $U(i)$ conflicts with the routing of packet q along $U(i)$, one of the following two claims hold: either packets q and x conflict on the up-tree, in which case q must contain the last rootward edge of the packet x, or they conflict during the leafward portion of their routing, which in this case ($i = j$) can only happen if they have the same first leafward edge. It follows that according to the optimal solution, packet q either crosses e_1 at t_1, or crosses e_2 at t_2.

We now consider the case where $i \neq j$. Since packet x was already scheduled on $U(i)$, when packet q was considered for $U(j)$, we have that $i < j$. 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 x along $U(i)$ can conflict only during the leafward portion. However, for two leafward portions to overlap, one must contain the first leafward edge of the other. If the schedulings conflict then both packets attempt to traverse this edge at the same time. 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 up-trees from latest to earliest, and $i < j$, it follows that packet x 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 x along $U(i)$. That is, packet q, when scheduled on $U(j)$, traverses e_2 at time t_2. ■

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

Proof: The children of every blue or green node correspond to the packets in the set S' that were dropped from the solution when packet x was added to the solution. By the definition of the algorithm, the weight of the packets in S' is less than half the weight of the packet x. ■

Claim 3 *For any (green or blue) node $\langle x \rangle$, the total weight of the green and blue nodes in the subtree rooted at $\langle x \rangle$ is at most $2w(x)$.*

Proof: We prove the claim by induction on the height of node $\langle x \rangle$. If $\langle x \rangle$ is a leaf (or has no blue children) the claim is obvious. By Claim 2, the total weight of the blue children of $\langle x \rangle$ is at most $(1/2)w(x)$. Applying the induction hypothesis to the subtrees rooted at the blue children of $\langle x \rangle$ we have that the total weight of the blue (and green) nodes in the subtrees rooted in all the blue children of $\langle x \rangle$ is at most $2 \cdot (1/2)w(x)$. Therefore the total weight of the blue and green nodes in the subtree rooted at $\langle x \rangle$ is at most $w(x) + 2 \cdot (1/2)w(x) = 2w(x)$. ■

Lemma 1 *The total weight of all the nodes in the subtree rooted at a green or blue node* $\langle x \rangle$ *is at most* $10 \cdot w(x)$.

Proof: By Claim 3, the total weight of the green and blue nodes in the tree rooted at $\langle x \rangle$ is at most $2w(x)$. Any red node is a child of a green or blue node. By Claim 1 the total weight of the red children of a green or blue node $\langle p \rangle$ is at most $4w(p)$. It follows that the total weight of the red nodes in the tree rooted at $\langle x \rangle$ is at most $8w(x)$. The weight of all the nodes in the tree rooted at $\langle x \rangle$ is therefore at most $10w(x)$. ■

Proof of Theorem 1: The theorem now follows from Lemma 1. The weight of the optimal solution is upper bounded by the total weight of the nodes in the blame graph. The total weight of the solution of **WT** is equal to the weight of the green nodes. The blame graph is a forest, where the root of every tree in the forest is a green node, and every green node is the root of a tree. Thus using Lemma 1, we have that the weight of the optimal solution is at most 10 times the weight of the solution of **WT**. ■

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: We build almost the same blame graph as for the weighted case. The difference is in the introduction of red nodes. When we add red nodes, we add only one red node per packet q in $OPT(I) - ALG(I)$, choosing arbitrarily one of the packets in S' as its parent.

We make a number of observations about the blame graph in this case. First, there are no blue nodes, since any packet that is scheduled is never discarded. This is because the weight of any packet is never larger than the weight of a scheduled packet with which it conflicts. Second, the weight of any red node is equal to the weight of its parent, since all packets have the same weight, and we add a single red node for every packet in $OPT(I) - ALG(I)$. The bounds relating the weight of the optimal solution to the weight of the blame graph, and the weight of **WT**'s solution to the weight of the green nodes, remain the same.

Now observe that the blame graph consists of a forest, where each tree consists of a green node that possibly has red children. We have already seen that each green node can have at most two red children. Therefore each tree consists of a green root and at most two red children. Since the weight of all nodes is the same, it follows that the weight of the solution of **WT** is at least a third of the weight of the optimal solution. ■

We also note that we can slightly improve the algorithm by optimizing the factor used in the replacement decision. In particular, we obtain an approximation ratio of $5 + \sqrt{24} \approx 9.89$ by using a factor of $1 + \sqrt{24}/4 \approx 2.22$ in the replacement decision.

2.2 Buffered schedules

We now turn to the question of how much buffering can help on the tree. We demonstrate that it is limited, and thus algorithm **WT** can also serve as a buffered approximation algorithm. We show that for any set of packets I_B that can be transmitted in its entirety using a buffered schedule on a tree, there is a subset I_{BL} of I_B, which can be transmitted in its entirety using a bufferless schedule, such that the total weight of the packets of I_{BL} is not much smaller than the total weight of the packets of I_B. Let n be the number of nodes in the tree. For a set of packets I, let $\sigma(I)$ be the maximum slack in I, let $R(I) = \min(|I|, \sigma(I) + 1)$, and let $T(I) = \min(n, |I|, \sigma(I) + 1)$. We prove the following theorem.

Theorem 3 *For any nonempty set of packets I_B which can be wholly scheduled on the tree by a buffered schedule, there is a subset $I_{BL} \subseteq I_B$, which can be wholly scheduled by a bufferless schedule and satisfies the following condition:*

- *if $T(I_B) = 1$ then $w(I_{BL}) = w(I_B)$.*
- *if $T(I_B) > 1$ then $w(I_{BL}) = \Omega(1/\log T(I_B))w(I_B)$.*

Proof: If $T(I) = 1$ we either have a tree of size 1, a single packet, or packets all with slack 0. In the first two cases $I_B = I_{BL}$ since either there are no edges in the tree or there is a single packet. In the latter case the fact that all packets have slack 0 means that the buffered schedule does not use buffers. Therefore the same schedule is also a bufferless schedule.

We now consider the case where $T(I) > 1$. The claim for this case follows from the following two lemmas that we prove below.

Lemma 2 *For any nonempty set of packets I_B which can be wholly scheduled on the tree by a buffered schedule, there is a subset $I_{BL} \subseteq I_B$ such that I_{BL} can be wholly scheduled by a bufferless schedule and $w(I_{BL}) = \Omega(1/\log n)w(I_B)$ (assuming $n > 1$).*

Lemma 3 *For any nonempty set of packets I_B which can be wholly scheduled on the tree by a buffered schedule, there is a subset $I_{BL} \subseteq I_B$ such that I_{BL} can be wholly scheduled by a bufferless schedule and $w(I_{BL}) = \Omega(1/\log R(I_B))w(I_B)$ (assuming $R(I_B) > 1$).*

■

Proof of Lemma 2: To prove the lemma we partition the set of packets in I_B into $L = O(\log n)$ classes based on the paths of the packets. For each packet $p \in I_B$ let $\pi_p = (v_1, \ldots, v_k)$ be the sequence of nodes into which packet p arrives while traveling in the network (i.e., the full path of the packet is $(s = v_0, v_1, \ldots, v_k = t)$.) We call π_p the in-path of p. The assignment of the packets to classes is done in such a way that the set of packets assigned to each class satisfies the following properties: the packets in this class can be further partitioned into sub-classes such that if two packets belong to different sub-classes then their paths do not intersect; and for each sub-class $\mathcal{C}$ there is a node v in the tree such that $v \in \pi_p$ for each $p \in \mathcal{C}$.

Using such partition, we can choose the class that has the largest total weight of packets, incurring a penalty of a factor of L. Then we consider the chosen packets by their sub-classes. Since any two packets belonging to two different sub-classes cannot conflict, we can consider each sub-class separately, determining the heaviest subset that can be scheduled without buffers. If we are guaranteed that the heaviest subset has a weight within a constant fraction of the weight of the original set, we can conclude that there is a bufferless schedule for a subset of I_B, with weight that is at least an $O(\log n)$ fraction of the weight of I_B. We now formally prove these claims.

We first prove that there is always a partition as defined above. Partitions similar to the one used in the following lemma appeared already in the literature in similar contexts (see for example, [2, 11])

Lemma 4 *Any set of packets on a tree of n nodes can be partitioned into $L = O(\log n)$ classes such that the packets of any one class can be further partitioned into a number of sub-classes with the following properties:*

1. *Two packets that belong to different sub-classes have (directed) paths that do not intersect.*
2. *For each sub-class there is a node in the tree which is included in all the in-paths of the packets in the sub-class.*

Proof: We first partition the nodes of the trees into classes. The partition is based on the fact that for any tree of n nodes, there is a node v, such that if v is removed, the tree becomes a set of sub-trees, each with at most $(2/3)n$ nodes (cf. [17]). We first pick one such node, v, in the original tree of n nodes, and assign it to class 1. Then we apply the same procedure recursively on all the trees created by the removal of v from the tree. All the nodes picked up in the second phase are assigned to class 2. We continue recursively, assigning the nodes to classes of increasing level (the base of the recursion is the case of a tree of a single node which is picked). We thus assign all the nodes to $L = O(\log n)$ classes.

Now we create L classes for the packets. For each packet p we consider its in-path as defined above. We assign packet p to the class that corresponds to the node in π_p having the lowest-numbered class.

Observe now that for any packet p in a class ℓ, there is exactly one node of class ℓ in π_p. This is because any path that connects two nodes of class ℓ passes through a node of a lower-numbered class. Now partition the packets in any class ℓ into sub-classes according to the single number-ℓ node in which they pass. Thus property number 2 of the claim is satisfied. As to property number 1 consider two packets p and q that belong to different sub-classes of the same class ℓ. Assume towards a contradiction that their (directed) paths (considering the whole paths, including the nodes $v_0 = s$) intersect in edges. Then the union of the two paths creates a (undirected) path between two distinct nodes of the same level ℓ. This means that this path includes a node of a level lower than ℓ. Observe now that this node cannot be the source node ($s = v_0$) of both packets, and therefore at least one of the packets would have been classified into a class lower than ℓ. This is a contradiction. ■

We now consider the packets of a single sub-class.

Lemma 5 *Let v be a node on a tree, and let J_B be a set of packets all having node v on their in-path. If there is a buffered schedule for all the packets in J_B, then there is a subset $J_{BL} \subseteq J_B$, and a bufferless schedule for all the packets in J_{BL}, such that $w(J_{BL}) \geq w(J_B)/2$.*

Proof: We first build an intermediate buffered schedule for all the packets in J_B. Consider the original buffered schedule. For any packet $p \in J_B$, let p_{in} be the time at which it crosses into v, and p_{out} be the time at which it leaves v. If the packet never leaves v (i.e., v is its destination), then we define p_{out} to be ∞. Now consider the following buffered schedule for J_B: any packet p is buffered only in v (traveling in a bufferless fashion until it reaches v, and after it leaves v), crossing into v at p_{in} and crossing out of v at p_{out} (unless $p_{out} = \infty$). We further modify this schedule so that a packet does not wait in v "unnecessarily" (i.e., if a packet is in v, it will be sent out of v at the earliest possible time). To that end, awe still make any packet p arrive at v at p_{in}, but att any time t, and for each outgoing edge e, we send out along e the packet p with the earliest p_{out}, among the packets currently at v (and having e on their path). Observe that no two such packets can have the same p_{out}, and each packet p leaves v no later than p_{out}.

To see that the above schedule is a feasible schedule for J_B we use the fact that the network is a tree. Note that any two packets, p and p', that cross into v on the same edge e, must have $p_{in} \neq p'_{in}$, since the original schedule is feasible. Since the network is a tree, the paths of p and p' intersect only in one continuous path that includes e. Therefore in the schedule we defined above p and p' do not interfere while traveling towards v. As any two packets that leave v on the same edge do that at different time steps, by similar arguments it follows that they do not interfere with each other while traveling away from v. Clearly two packets that do not cross into v on the same edge do not interfere while traveling towards v, and two packets that leave v on different edges do not interfere with each other while traveling away from v (since the network is a tree). Observe that since each packet p arrives at v at time p_{in}, and travels towards v in a bufferless fashion, then it leaves its source not earlier that its release time. Similarly, since every packet p leaves v no later than p_{out}, and proceeds in a bufferless fashion, then it arrives to its destination by its deadline.

We now prove the existence of a subset $J_{BL} \subseteq J_B$ and modify the above schedule for the packets in J_{BL}, so that the schedule is wholly bufferless. We show below that we have such subset with total weight that satisfies $w(J_{BL}) \geq w(J_B)/2$. Each packet $p \in J_{BL}$ will cross into v at time t such that $p_{in} \leq t \leq p_{out} - 1$, and will leave v at time $t + 1$ (if it has to leave v). For each edge of v at most one packet will cross into v and at most one packet will cross out of v, at any time. Since the schedule will be bufferless this guarantees that the schedule is feasible in that each packet leaves its source no earlier than its release time, and arrives by its deadline, and no two packets interfere.

To do the above we consider the situation at hand in a somewhat different way. Let δ be the degree of v. Consider a switch with δ input ports and δ output ports. An equivalent scenario to the one we have is the scenario where each packet $p \in I_B$ arrives to the input port of the switch at time p_{in}, and is placed at that time at an appropriate buffer for its output port. The packet is inserted into the buffer according to its time p_{out}, such the these times are increasing going from the head of the buffer to its tail. At each time unit the packet at the head of each buffer is extracted and sent on the output port. This type of switch is called "Output Queuing" (OQ) in the literature (cf. [24, 6]). Note that in each time unit several packets may be placed on the same output buffer. We now consider another type of switch that is closely related to the bufferless setting that we need. This type of switch is called "Combined Input Output Queuing" (CIOQ). In such a switch there are buffers at both the input ports and the output ports. Arriving packets are placed in the input buffer of the port they arrive on, and packets leave from the buffers of the output ports. Packets can be transferred from the input buffers to the output buffers in the following way only. Packets are transferred in cycles. In each cycle packets are transferred between buffers according to a *matching* between the input buffers and the output buffers (i.e., in each cycle, at most a single packet is extracted from each input buffer, and at most a single packet is received in each output buffer.) We now use a result by Chuang et al. [6]. They proved the following result on the relation between OQ switches and CIOQ switches.

Theorem 4 ([6] Theorem 4) *Consider a sequence of packets that travel through an OQ switch that adheres to a Push-In First-Out policy.*[1]*. Replace the OQ switch by a CIOQ switch that has two cycles in each time unit. If the same sequence of packets arrive into the CIOQ switch at the*

[1]A policy for an OQ is called PIFO if it operates by extracting in each time step the packet at the head of each output queue, while the insertion of the packets into the queue is arbitrary at the time of the insertion. The relative order of two packets in a queue cannot however be changed after their insertion.

same times as they arrive into the OQ switch, then there is a feasible schedule for the CIOQ switch such that it delivers the packets over its output ports at the same times as they are delivered by the OQ switch. ■

We note that the schedule that we defined can indeed be a schedule of a OQ with a PIFO policy, and therefore the above theorem applies. We now consider the schedule guaranteed by the above theorem for a CIOQ. For each time unit we consider the two cycles, the two matchings, and the two sets of packets transferred in them. We take the heavier of the two sets (in term of the weight of the packets). Thus, for each time t we have a matching M_t and a set of packets P_t that are transferred between input buffers and output buffers at time t. We now define the set J_{BL} to be $\cup_t P_t$. We set the (bufferless) schedule for each packet $p \in J_{BL}$ as follows: if $p \in P_t$ then p crosses into v at time t and leaves v at time $t+1$. The departure time of p from its source is set so that it travels in a bufferless fashion and arrives at v at time t. Since according to the schedule guaranteed by the above theorem, each packet p of J_B arrives at the CIOQ switch at time p_{in} and leaves it by time p_{out}, we have that for all packets in P_t, $p_{in} \leq t \leq p_{out} - 1$. This is because the packet must be transferred from its input buffer to its output buffer not earlier than its arrival time to the switch, and before it leaves the switch. Therefore each packet $p \in J_{BL}$ arrives at v at time t such that $p_{in} \leq t \leq p_{out} - 1$. Since we take at each time t the heavier of the two sets of packets we lose at most half of the total weight of the packets of J_B, and thus $w(J_{BL}) \geq w(J_B)/2$, as required. ■

The claim of Lemma 2 now follows by combining Lemma 4 and Lemma 5. ■

Proof of Lemma 3: We will show below that algorithm **WT**, applied to the set I_B selects and schedules a subset of packets as required. We prove this using similar techniques to those used for the analysis in Section 2.1, as well as techniques developed for comparing bufferless schedules to buffered schedules in [1]. We build a blame graph similar to the one used in the previous section.

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 is somewhat 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 bufferless schedule. When comparing to a buffered schedule, we will consider all the up-trees on which the packet is eligible. Thus, for any packet q in the buffered schedule that does not have a green or a blue node, and for each up-tree along which it is eligible, the blame graph has a collection of red nodes.

We now precisely define the red nodes. Consider a packet $q \in I_B$ for which there is no green or blue node in the blame graph. Let k be the number of up-trees on which q is eligible, and $U(i)$ be any such up-tree. Since, when applying **WT**, q is not routed on $U(i)$ it follows that when iteration i of **WT** occurred, there was a set S' of scheduled packets that conflicted with scheduling q on $U(i)$, such that $w(S') \geq w(q)/2$. For each packet $x \in S'$ we define a red node $\langle q, x\rangle$ and an edge from $\langle q, x\rangle$ to $\langle x\rangle$. Note that we add such nodes for each of the k up-trees $U(i)$ on which packet q is eligible.

We assign weights to the nodes in the blame graph as follows. Green and blue nodes $\langle p\rangle$ have weight $w(p)$ as in Section 2.1. The weight of red nodes is slightly different. Let q be a packet for which there are red nodes, and let k be the number of up-trees on which it is eligible. For each such up-tree $U(i)$, let S_i be the set (S') that prevented **WT** to schedule q on $U(i)$. Let $x \in S_i$ and $\langle q, x\rangle$ be a red node. The weight of $\langle x, q\rangle$ is set to $\frac{1}{k} \cdot w(q) \cdot \frac{w(x)}{w(S_i)}$ (i.e., we equally partition the weight of q between all the up-trees on which it is eligible, and within each up-tree we assign weight to the red nodes as in the previous section). Observe that the total weight of all the red nodes $\langle q, x\rangle$ for a given q is $w(q)$, and that the weight of a red node $\langle q, x\rangle$ is at most $2w(x)/k$.

Let $H(n)$ denote the harmonic sum $\sum_{i=1}^{n} \frac{1}{i}$. We prove the following claim.

Claim 4 *The total weight of the red children of any (green or blue) node $\langle p\rangle$ is at most $8H(R(I_B))w(p)$.*

Proof: Let $U(i)$ be the up-tree along which packet p was scheduled by **WT** (packet p may have later been discarded from the solution). Let $S_k \subseteq I_B$ be the set of packets q of slack at most $k-1$ such that $\langle q, p\rangle$ is a red child of $\langle p\rangle$. Consider a packet q in S_k. It follows that there exists an up-tree $U(j)$, on which packet q is eligible, such that the scheduling of q along $U(j)$ (in a bufferless schedule) conflicts with the scheduling of p along $U(i)$. Moreover, following the reasoning of Claim 1, we can show that a conflict between these two schedulings 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 e_1 when scheduled along $U(i)$. For any q in S_k, the aforementioned conflict with p implies that there is an eligible bufferless schedule of q such that either q crosses e_1 at time t or q crosses e_2 at time $t+1$ (both events occur in this schedule if the path of q includes both e_1 and e_2). Since q has slack at most $k-1$, and uses at least one of e_1 and e_2 at the above times, it follows that in the buffered schedule of I_B, q must either traverse e_1 at a time in the interval $[t-k+1, t+k-1]$ or e_2 at a time in the interval $[t-k+2, t+k]$ (or both, if it uses both edges). Since only one packet can cross any edge at any given time in any schedule, it follows that there are at most $4k$ packets in S_k ($2k$ for the last rootward edge and $2k$ for the first leafward edge).

A packet with slack exactly $k-1$ is eligible on exactly k distinct up-trees. Therefore, the weight of any red node $\langle q, p\rangle$ (of a packet q), where q has slack $k-1$, is at most $2w(p)/k$. Define S_0 to be the empty set; the packets in $S_k - S_{k-1}$ are the packets with slack $k-1$. The total weight of the red children of $\langle p\rangle$ is therefore at most

$$\sum_{k=1}^{\sigma(I)+1} (|S_k| - |S_{k-1}|)\frac{2w(p)}{k} \le \sum_{k=1}^{\sigma(I)+1} 4 \cdot \frac{2w(p)}{k} = 8w(p) \sum_{k=1}^{\sigma(I)+1} \frac{1}{k} = 8H(\sigma(I)+1)w(p).$$

At the same time, since the total number of the packets in the sets S_k is at most $|I_B|$, we can also write:

$$\sum_{k=1}^{\sigma(I)+1} (|S_k| - |S_{k-1}|)\frac{2w(p)}{k} \le \sum_{k=1}^{\lceil \frac{|I_B|}{4} \rceil} 4 \cdot \frac{2w(p)}{k} = 8w(p) \sum_{k=1}^{\lceil \frac{|I_B|}{4} \rceil} \frac{1}{k} = 8H(\lceil \frac{|I_B|}{4} \rceil)w(p).$$

■

Since the definition of the green and blue nodes has not changed, Claim 3 still holds, and thus we can prove the following.

Lemma 6 *For any (green or blue) node $\langle x\rangle$ in a blame graph, the total weight of the nodes in the subtree rooted at $\langle x\rangle$ is $(2 + 16H(R(I_B))) \cdot w(x)$.*

Proof: We repeat the proof of Lemma 1, replacing Claim 1 by Claim 4. The total weight of the green and blue nodes in the tree rooted at $\langle x\rangle$ is $2(x)$. Any red node is a child of a blue or green node. Using Claim 4 we therefore have that the total weight of all the nodes in the tree rooted at $\langle x \rangle$ is $2w(x) + 2w(x) \cdot (8H(R(I_B))) = (2 + 16H(R(I_B))) \cdot w(x)$. ■

Lemma 3 now follows from Lemma 6, using the same arguments as in the proof of Theorem 1. ■

We can conclude that algorithm **WT** also serves as an approximation algorithm for the *buffered* version of the problem. The following theorem follows from Theorem 3 using the substitution $OPT(I)$ for I_B.

Theorem 5 *For any weighted buffered problem instance I for the tree, we have*

$$w(ALG(I)) \geq w(OPT(I))/O(\log T(OPT(I))).$$

■

We note that we can replace the notion of "slack" by the alternative notion of "effective slack". Given a schedule for a set of packets, the effective slack of a packet is defined to be the time the packet spends waiting in the buffers after leaving its source and before arriving at its destination. Thus, the difference between the arrival time at the destination and the departure time at the source equals the sum of the effective slack of the packet and the distance traversed by the packet. In other words, we can change the specification of a packet to have its release time at the time it leaves its source, and its deadline to be the time it arrives at its destination. Thus, we may be able to reduce the maximum slack without reducing the total weight of the packets being scheduled, obtaining a possibly better bound.

Note also that in [1] a problem instance I is presented 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 $\Omega(\log T(I))$. This instance is of the form that $\log T(I) = \Theta(\log n) = \Theta(\log \sigma(I)) = \Theta(\log |I|)$. Thus, the result of Theorem 3 is asymptotically optimal in terms of the measures that we consider here.

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 $O(d \cdot 2^d)$ in each of the results.

In the following we consider dimension order routing on the two dimensional mesh. We partition the set of packets into two groups: one consisting of packets that either travel rightward and then upward or travel leftward and then downward, and the other group consisting of the remaining packets. (A packet that travels along a row only or a column only is placed in the two groups.) We run algorithm **WM** for the two groups separately and pick the better of the two solutions. We thus incur a penalty of at most a factor of 2. In our description and analysis of **WM** below, we assume without loss of generality that every packet either travels rightward and then upward or travels leftward and then downward. A packet of the first type is called an *lr-packet*, and a packet of the second type is called an *rl-packet*.

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 are no longer 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 here 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. For an lr-packet it is an lr-mesh, and for an rl-packet 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 $d(e) + t(e)$ (resp., $d^*(e) + t(e)$), 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., $d^*(e) + t(e)$ and rl-mesh) and the one with the largest value of $d(e) + t(e)$ is the *latest* lr-mesh (resp., $d^*(e) + t(e)$ and 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 or rl-meshes (depending on the direction in which they travel vertically). As in the tree algorithm, we do this in such a way that a packet that traverses at time t its first vertical edge e' will be associated with the same lr-mesh (or rl-mesh), regardless of whether it travels horizontally beforehand.

In the following we describe algorithm **WM**. It calls two algorithms, lr-**WM**, and rl-**WM**, that separately schedule lr-packets and rl-packets, respectively. We give a formal description of lr-**WM**. Algorithm rl-**WM** is analogous with the appropriate changes to deal with rl-packets (i.e, exchanging any mentions of "right" and "left").

Algorithm **WM**:

$S_1 =$lr-**WM**.
$S_2 =$rl-**WM**.
Return $S_1 \cup S_2$.

Algorithm lr-**WM**:

Let S be the empty set.
Let N be the set of all lr-packets.
Let $L(1)\dots L(k)$ be the set of all possible lr-meshes in order from latest to earliest.
For $i = 1$ to k:
  Let $E(i)$ be the set of packets in $N - S$ eligible for routing on $L(i)$.
  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' be the set of packets in S that conflict with p if p were routed on $L(i)$.
    If $w(S') < w(p)/2$
      Remove all packets in S' from S and add p to S.
      Assign the packet p to $L(i)$.
      Remove the packet p from N.
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 **WT** considers only one class of up-trees, **WM** considers two different classes of mesh subgraphs, the lr-meshes and the rl-meshes. We can still use the analysis framework used in Section 2 for each one of lr-**WM** and lr-**WM**. 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 6 *For any weighted problem instance I for the mesh, the total weight of the packets routed by algorithm* **WM** *is $\Omega(w(OPT(I)))$.*

Since lr-packets and rl-packets do not interfere with each other and form a partition of all packets, we can prove the above theorem by proving the following theorem for lr-packets and lr-**WM**, and rl-packets and rl-**WM**, separately.

Lemma 7 *For any weighted problem instance I of lr-packets (resp. rl-packets) for the mesh, the total weight of the packets routed by algorithm lr-***WM** *(resp. rl-***WM***) is $\Omega(w(OPT(I)))$.*

Proof: We prove the theorem for lr-packets and lr-**WM**. The proof for rl-packets and rl-**WM** is analogous.

As in the proof in Section 2 we create a *blame graph.* The blame graph for the present proof will also have green, blue and red nodes, which are the same as those used in the blame graph of Section 2. Weights are assigned to the nodes in the blame graph in the same way as is done in Section 2.

We now prove a claim analogous to Claim 1.

Claim 5 *The total weight of the red children of any (blue of green) node $\langle x\rangle$ is at most $4w(x)$.*

Proof: The proof is almost identical to the proof of Claim 1, with the appropriate changes required to deal with rl-meshes instead of up-trees.

As shown in the discussion in Section 2, the weight of any red node is at most twice the weight of its parent. Thus we only need to show that every node has at most two red children.

Consider a blue or green node $\langle x \rangle$ with a red child, say $\langle q, x \rangle$. Let $L(i)$ be the lr-mesh along which packet x was scheduled by lr-**WM** (when the blue or green node was added to the blame graph).

Let e_1 be the last horizontal edge traversed by x, and e_1 the first vertical edge traversed by x, and let t_1 and t_2 be the times at which packet x traverses e_1 and e_2, respectively, when routed along $L(i)$. We will show that if a schedule for packet q conflicts with the schedule of x along $L(i)$, then either q traverses e_1 at t_1, or q traverses e_2 at t_2. Any red node represents a packet whose schedule according to the optimal solution conflicts with the schedule selected by **WM** for its parent. Since in the optimal solution only one packet can traverse any edge at any given time, we have that the node corresponding to x can have at most two red children.

Let $L(j)$ be the lr-mesh along which a packet q is routed in the optimal schedule. We know that in iteration j packet q was considered for scheduling on $L(j)$, but was rejected because of a set S' which includes packet x, scheduled on $L(i)$. This means that the schedule of x on $L(i)$ conflicts with the schedule of q on $L(j)$. We now distinguish between two cases. The first one is when $i = j$, and the second one is when $i \neq j$.

If $i = j$, then by the ordering in which we consider the packets in iteration i, the packet x must have a rightmost edge that is not to the right of the rightmost edge of the packet q. Since the routing of packet x along $L(i)$ conflicts with the routing of packet q along $L(i)$, one of the following two claims hold: either packets q and x conflict on the horizontal portion of their path, in which case q must contain the rightmost edge of the packet x, or they conflict on the vertical portion of their path, which in this case ($i = j$) can only happen if they have the same first horizontal edge. It follows that according to the optimal solution, packet q either crosses e_1 at t_1, or crosses e_2 at t_2.

We now consider the case where $i \neq j$. Since packet x was already scheduled on $L(i)$, when packet q was considered for $L(j)$, we have that $i < j$. Since the lr-meshes $L(j)$ and $L(i)$ are different, it follows that the routing of q along $L(j)$ and the routing of x along $L(i)$ can conflict only in the vertical portion of their paths. However, for two vertical portions to overlap, one must contain the first vertical edge of the other. If the schedulings conflict then both packets attempt to traverse this edge at the same time. If those two packets traverse that edge at the same time, then the packet that starts its vertical journey at an earlier time must contain the first vertical edge of the other packet. Since we process the lr-meshes from latest to earliest, and $i < j$, it follows that packet x starts its vertical journey at a later time than packet q, and thus the routing of packet q along $L(j)$ contains the first vertical edge of the routing of x along $L(i)$. That is, packet q, when scheduled on $L(j)$, traverses e_2 at time t_2. ■

Claims 2 and 3 hold for the case of the mesh with no change. Using these claims, and Claim 5 above, Lemma 1 holds for the mesh as well. The proof of Theorem 7 now follows by the same arguments as those used in the proof of Theorem 1. ■

We also have a 6-approximation algorithm for dimension order routing in the case where all packets have the same weight. This follows from the statement of Theorem 2, applied to the mesh, where the penalty in the approximation ratio (of a factor of 2) is due to the partitioning of the packets on the mesh into two disjoint sets based on the orientation of the paths. Furthermore, we can place an upper bound on the approximation ratio of **WM** with respect to buffered schedules using an analysis similar to the one given in Lemma 3 in Section 2.2. Recall that for any set of packets I, $R(I) = \min(|I|, \sigma(I) + 1)$; let $OPT(I)$ be the set of packets scheduled in an optimal buffered solution; and let $ALG(I)$ be the set of packets scheduled by **WM**.

Theorem 7 *For any weighted buffered problem instance I for the mesh,* $w(ALG(I)) \geq w(OPT(I)/O(\log R(OPT(I$

Proof: To prove the theorem we use a proof analogous to the proof of Lemma 3. The only difference is the proof of Claim 4, where we make the analogous changes to those done to prove Claim 5 on the basis of the proof of Claim 1. ■

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 $t(i) - i$ is independent of 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 problem is identical to the problem of finding a maximum weighted independent set in interval graphs and can be solved 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 8 *For any I, $w(ALG(I)) \geq w(OPT(I))/2$.*

Proof: Let $OPT_\ell(I)$ denote the subset of packets in $OPT(I) - ALG(I)$ that are scheduled along scan line ℓ by the optimal solution. Note that any packet in $OPT_\ell(I)$ was presented as input to the algorithm that computed the maximum weight set N_ℓ. Suppose $w(ALG(I)) < w(OPT(I))/2$, then $w(\cup_\ell OPT_\ell(I)) > w(ALG(I))$. Then for some ℓ, $w(OPT_\ell(I)) > w(N_\ell)$, 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, we show that for any set of packets I_B that can be transmitted in its entirety using a buffered schedule on the array, there is a subset I_{BL} of I_B, which can be transmitted in its entirety using a bufferless schedule, such that the total weight of the packets of I_{BL} is not much smaller than the total weight of the packets of I_B. As in [1], we express the comparison in terms of the parameter $\Lambda(I_B) = \min\{\sigma(I_B) + 1, \delta(I_B), |I_B|\}$, where $\sigma(I_B)$ is the maximum slack in I_B and $\delta(I_B)$ is the maximum span in I_B. In the following we assume that $\Lambda(I_B) > 1$. For $\Lambda(I_B) = 1$ it is straightforward to see that the set I_{BL} can be the set I_B itself (see also Section 2.2).

Theorem 9 *For any nonempty set of packets I_B which can be wholly scheduled on the array by a buffered schedule, there is a subset $I_{BL} \subseteq I_B$, which can be wholly scheduled by a bufferless schedule and satisfies that*

$$w(I_B) \leq 4(\lceil \log(\Lambda(I_B)) \rceil + 1) \cdot w(I_{BL}).$$

Proof: We will show that $w(I_B) \leq 4(\lceil \log x \rceil + 1) \cdot w(I_{BL})$ for each $x \in \{\sigma(I_B) + 1, \delta(I_B), |I_B|\}$. We start with $x = \sigma(I_B) + 1$. In this case, we will show that the set I_{BL} can be the set obtained by applying **WA** to the set I_B. More precisely, we will show that $w(I_B) \leq 4\lceil \log(\sigma(I_B)+1) \rceil w(ALG(I_B))$. We divide the set I_B into $\lceil \log(\sigma(I_B) + 1) \rceil$ sets such that the ith set I_i, $i \geq 0$, consists of the packets in I_B that have slack at least $2^i - 1$ and less than $2^{i+1} - 1$. We now prove that, for every i, $w(I_i - ALG(I_B))$ is at most $4w(ALG(I_B))$.

Consider any scan line ℓ. Let X_ℓ be the set of packets in $I_i - ALG(I_B)$ that are individually eligible to be scheduled on ℓ. Since the slack of each packet in X_ℓ is less than $2^{i+1} - 1$, 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 $I_i - ALG(I_B)$ contributes to at least 2^i scan lines, we obtain the following inequality by adding Equation 1 over all scan lines.

$$w(I_i - ALG(I_B)) \leq 4w(ALG(I_B)). \tag{2}$$

Equation 2 now yields

$$\begin{aligned} & w(I_B - ALG(I)_B) \\ \leq & \sum_{i=0}^{\lceil \log(\sigma(I_B)+1) \rceil - 1} w(I_B - ALG(I_B)) \\ \leq & \; 4\lceil \log(\sigma(I_B) + 1) \rceil w(ALG(I_B)). \end{aligned}$$

We next show the desired claim for $x = \delta(I)$. Again, we will show that the set I_{BL} can be the set obtained by applying **WA** to the set I_B. Divide the set I_B into $\lceil \log(\delta(I_B)) \rceil$ sets such that the ith set I_i consists of the packets in I_B that have span at least 2^i and less than 2^{i+1}. We now prove that $w((I_i - ALG(I_B)))$ is at most $4w(ALG(I_B))$.

Consider any scan line ℓ. Let X_ℓ be the set of packets in $I_i - ALG(I_B)$ 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 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 $I_i - ALG(I_B)$ 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: $w(I_i - ALG(I_B)) \leq 4w(ALG(I_B))$. We now prove the desired claim as follows.

$$\begin{aligned} w(I_B) & \leq \sum_{i=1}^{\lceil \log(\delta(I_B)) \rceil} w(I_i - ALG(I_B)) + w(ALG(I_B)) \\ & \leq (4\lceil \log \delta(I) \rceil + 1) w(ALG(I_B)). \end{aligned}$$

Finally, consider the case $x = |I_B|$. We first observe that all of the packets that have slack at least $|I_B| + 1$ can always be routed irrespective of how the other packets are routed. This is because

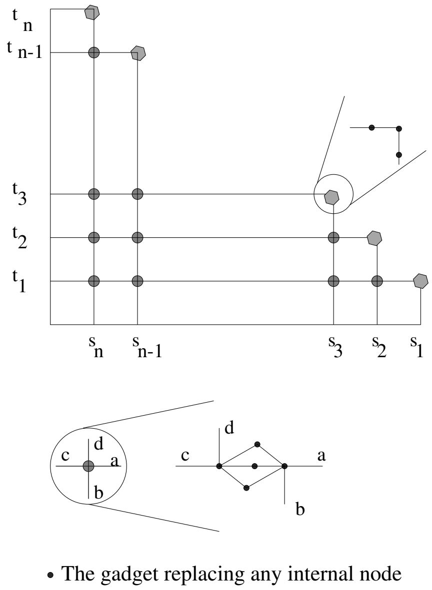

Figure 1: Graph used in the proof of Theorem 10

for any packet with slack at least $|I_B| + 1$, there is at least one scan line along which it can be routed and no other packet will be routed since there are only $|I_B|$ packets. Therefore, it remains to consider only those packets in I_B that have slack at most $|I_B|$. Let I'_B denote the subset of packets in I_B that have slack at most $|I_B|$. By our earlier argument concerning slack, it follows that $w(I'_B)$ is at most $4\lceil \log(|I_B| + 1) \rceil w(ALG(I'_B))$. By letting $I_{BL} = ALG(I'_B) \cup (I_B - I'_B)$, the desired claim follows. ∎

5 Hardness of off-line approximations

In this section, we establish lower bounds on the approximability of the time-constrained routing problem. All of the results described involve instances in which none of the packets have any slack, and thus apply both to the buffered and bufferless case. Furthermore, the lower bounds apply to the special case where all packets have the same weight.

We first provide strong evidence that there does not exist a polynomial-time $k^{1-\epsilon}$-approximation, for any $\epsilon > 0$, for the time-constrained routing problem with k packets on arbitrary topologies.

Theorem 10 *For any $\epsilon > 0$, there is no $k^{1-\epsilon}$-approximate algorithm for the time-constrained problem, unless NP = ZPP, where k is the number of packets in the instance of the routing problem.*

Proof: We give a reduction from the independent set problem which is known to be $n^{1-\epsilon}$-hard to approximate, unless NP = ZPP, where n is the number of nodes in the given graph [10]. Our starting point is input $G = (V, E)$ to the independent set problem. We use a *mesh-like* network for the time-constrained routing problem, and specify packets and routes through a reduction from G; see Figure 1. Our construction here is similar to the one used in [9] 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).

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. ■

We observe that the reduction in Theorem 10 uses the same network, illustrated in Figure 1, for every instance of the independent set problem. Thus, the inapproximability result holds even if the routing algorithm for the time-constrained routing problem is allowed to pre-process the given network "for free". That is, Theorem 10 also holds in a model where each network has a specific algorithm, and the input is only the set of packets and their paths.

In the instances generated in the proof of Theorem 10 above, all paths are shortest paths. We next 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 the routing is along dimension order paths.

Theorem 11 *For any $\epsilon > 0$, there is no $n^{1-\epsilon}$-approximate algorithm for the time-constrained problem even when the underlying graph is an n-node mesh, unless NP = ZPP.*

Proof: The mesh-like graph constructed for Theorem 10 loses its mesh structure once we replace the nodes with the gadgets. In order to establish the hardness result for the mesh, we suitably modify the gadgets, as shown in Figure 2. The rest of the proof is similar to the proof of Theorem 10.

As before, we are given an instance of the maximum independent set problem; that is, we are given a graph G with n nodes. We consider a $4n \times 4n$ mesh, with rows numbered 0 through $4n-1$ from the bottom to the top and columns labeled 0 through $4n-1$ from the right to the left. The entire

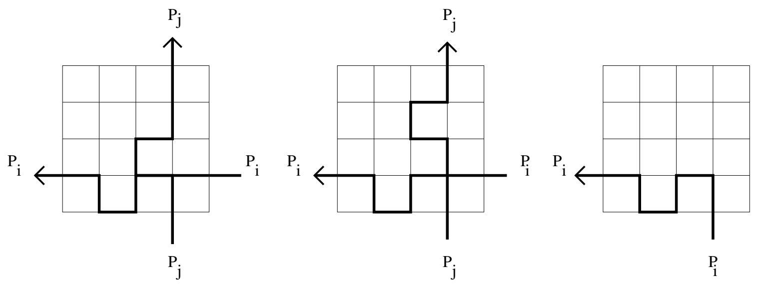

Figure 2: Gadgets used in the proof of Theorem 11

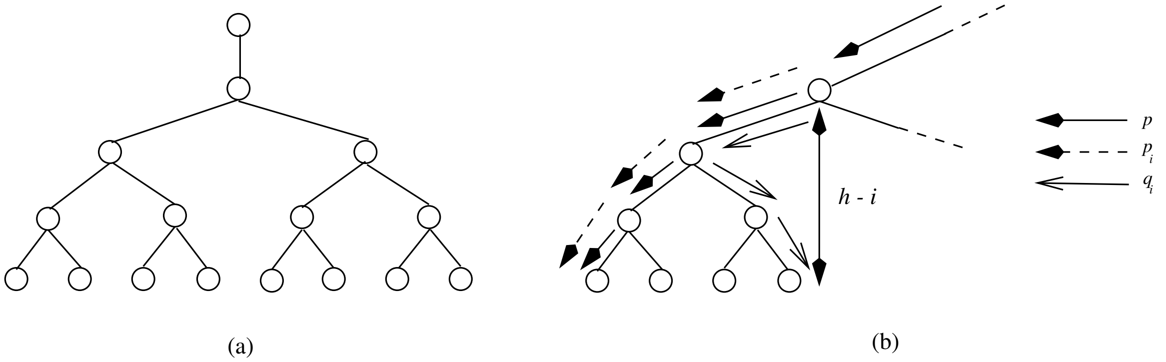

Figure 3: (a) Lower bound tree for $h = 4$; (b) Time step i: p is a packet currently scheduled by the on-line algorithm, p_i and q_i are new packets injected at time step i.

mesh is divided into $n \times n$ *blocks* of 4×4 submeshes. All of the source nodes s_1 through s_n are located on row 0. More specifically, the coordinates of s_i are given by $(0, 4i - 3)$. Similarly, all of the destination nodes t_1 through t_n are on column 0. The coordinates of t_i are given by $(4i - 3, 0)$.

We now specify the paths of the time-constrained routing problem. The path π_i from source s_i to destination t_i represents vertex i of G and its "interaction" with other paths represents the adjacency of i in G. The path passes through $i - 1$ blocks vertically, then takes the form of the diagonal gadget, and then passes through $n - i$ blocks horizontally. While going through a block vertically (resp., horizontally), it may interact with a path π_j for $j < i$ (resp., $j > i$). In this case, the path will take the form as described in Figure 2 depending on whether i and j are adjacent in G. It is easy to verify that each path has length $6n$. We set the release time r_i of the ith packet to be $6(i - 1)$ and the deadline to be $r_i + 6n$ (i.e., no slack is given).

As in the proof of Theorem 10, it is easy to see that for any $i < j$, packets starting at s_i and s_j at their respective release times arrive simultaneously at the vertex where the paths π_i and π_j intersect. It now follows from the construction of the gadgets the pair of packets collide if and only if i and j are adjacent in G. The claimed result now follows from the hardness of the independent set problem. ■

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 show here that the analogous result is not possible in the case of trees, even when all packets have the same weight. We derive this lower bound by giving a family of tree networks, parameterized by the parameter $h \geq 1$, and showing that for any deterministic on-line algorithm, there is a sequence of packets on these networks such that the on-line algorithm will schedule at most a single packet, while an off-line algorithm can 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 preemptive on-line algorithms, i.e., algorithms that are allowed to drop a packet that has already been scheduled.

Theorem 12 *The competitive ratio of any on-line algorithm for time-constrained routing on an n-node tree is $\Omega(\log n)$.*

Proof: We establish the claim by first defining the tree used in the argument and then defining the set of packets that form the input provided to the on-line algorithm.

The lower bound tree. For any $n = 2^h$, $h \geq 1$, we build a tree T_n of height h. For any nonnegative integer i, let $T(i)$ denote a complete binary tree of height i. The tree T_n employed in our lower bound consists of a root r with a single child that forms the root of the complete binary tree $T(h-1)$. Thus, for each i in the range $0 \leq i < h$, every node of T_n 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 $0 \leq i < h$, we introduce two new packets at a node at height $h - i + 1$. Each packet is destined to a leaf node and has deadline time $h + 1$. 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 optimal solution 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 descending 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 $i > 0$, 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_i, 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.

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 12. ∎

Acknowledgments

We would like to thank Thomas Erlebach for pointing out an error in the analysis of the dimension order routing algorithm for meshes in an earlier version of this paper. We also thank Boaz Patt-Shamir for pointing out to us the paper of Chuang et al. [6].

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] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén, Competitive non-preemtive call control. *5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 312-320.

[3] M. Andrews, A. Fernandez, M. Harchol-Balter, F.T. Leighton, and 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*, 294-302.

[4] M. Andrews and L. Zhang (1999): Packet routing with arbitrary end-to-end delay requirements. *31st ACM Symp. on Theory of Computing*, 557-565.

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

[6] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar, Matching Output Queueing with a Combined Input Output Queued Switch. Journal on Selected Areas in Communication, Vol. 17, No. 6, pp. 1030–1039, 1999. Also in proceedings of INFOCOM 99.

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

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

[9] 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. *31st ACM Symp. on Theory of Computing*, pages 19–28.

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

[11] M. R. Henzingerm, and S. Leonardi, Scheduling Multicasts on Unit-Capacity Trees and Meshes, *10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 438–447.

[12] 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.

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

[14] F.T. Leighton (1992): Methods for message routing in parallel machines (invited survey). *24th ACM Symp. on Theory of Computing*, 77-96.

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

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

[17] J. Van Leeuwen, Editor. Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity. The MIT Press, 1990.

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

[19] K.-S. Lui and S. Zaks (1999): Scheduling in synchronous networks and the greedy algorithm. *Theoretical Comp. Sci.*, 220:157-183.

[20] M.W. Mutka (1994): Using rate monotonic scheduling technology to support real-time communications in wormhole networks. *Wkshp. on Parallel and Distributed and Parallel Real-Time Systsems*, 194-199.

[21] 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.

[22] 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.

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

[24] B. Prabhakar, and N. McKeown, On the speedup required for combined input and output queued switching. Stanford University Computer Systems Technical Report CSL-TR-97-738.

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

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

[27] 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.*

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

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