PACKET ROUTING: COMPLEXITY AND ALGORITHMS

BRITTA PEIS, MARTIN SKUTELLA, AND ANDREAS WIESE

Abstract. Store-and-forward packet routing belongs to the most fundamental tasks in network optimization. Limited bandwith requires that some packets cannot move to their destination directly but need to wait in intermediate nodes on their path or take detours. It is desirable to find schedules that ensure fast delivery of the packets, in particular for time critial applications. In this paper we investigate the packet routing problem theoretically. We prove that the problem cannot be approximated with a factor of $6/5-\epsilon$ for all $\epsilon > 0$ unless $P = NP$. We show that restricting the graph topology to planar graphs or even directed trees still does not allow a PTAS.

We present three individual algorithms for packet routing on trees: For undirected trees we give a factor 2 approximation. For directed trees we show that the path coloring problem can be solved optimally in polynomial time. Based on this, we show how to construct a schedule with length $C + D - 1$ where C and D denote the trivial lower bounds congestion and dilation. If the lengths of the paths of the packets are pairwise different, we can compute a schedule of length D on directed trees which is optimal. For all cases we show that our analysis is tight. Finally, we show that it is NP-hard to approximate the packet routing problem with an absolute error of k for any $k \geq 0$.

1. Introduction

In this paper we study the packet routing problem. Given a set of packets in a network originating at possibly different start vertices we want to transfer them to their respective destination vertices. The goal is to minimize the overall makespan, that is the time when the last packet arrives at its destination. We consider the offline version of the problem where all information about the network and the packets, in particular the start- and destination vertices are given in advance. In our routing model, we assume store-and-forward routing. This means that every node can store arbitrarily many packets but each link (a directed or undirected edge) can be used by only one packet at a time. We study the case where the paths of the packets are fixed in advance as well as the case where their computation is part of the problem. Moreover, we distinguish between different types of the underlying graphs, e.g., directed graphs, undirected graphs, planar graphs or trees.

This has important applications in all settings where packets need to be transfered through a network. In computer networks like local area networks (LANs) or the internet many data packets need to be routed. The priority of the packets in the schedule is not immediately clear and inappropriate routing rules can lead to ineffective schedules. Delay due to packet latency is not desirable. In particular, time-critial applications like voice over IP (VoIP) need their packets to be delivered within a certain time frame in order to work accurately. Therefore we are interested in schedules that guarantee the packets to arrive at their respective destinations as early as possible. Finding such effective schedules in distributed systems is a challenging task.

1.1. **Packet Routing Problem.**

The packet routing problem is defined as follows: Let $G = (V, E)$ be a directed or undirected graph. A packet $M_i = (s_i, t_i)$ is a tupel consisting of a start vertex $s_i \in V$ and a destination vertex $t_i \in V$. Let $\mathcal{M} = \{M_1, M_2, M_3, ..., M_N\}$ be a set of packets.

Then $(G, \mathcal{M})$ is an instance of the *packet routing problem with variable paths.* The problem has to parts: First, for each packet M_i we need to find a path $P_i = (s_i = v_0, v_1, ...v_{\ell-1}, v_\ell = t_i)$ from s_i to t_i such that $\{v_i, v_{i+1}\} \in E$ if G is undirected or $(v_i, v_{i+1}) \in E$ if G is directed for all i with $0 \le i \le \ell - 1$. Assuming that it takes one timestep to send a packet along an edge we need to find a routing schedule for the packets such that

- each message M_i follows its path P_i from s_i to t_i and
- each edge is used by at most one packet at a time

We assume that time is discrete and that all packets take their steps simultaneously. The objective is to minimize the makespan, i.e., the time when the last packet has reached its destination vertex. For each packet M_i we define D_i to be the length of the shortest path from s_i to t_i, assuming that all edges have unit length. Moreover, the *dilation* D is defined by $D := \max_i D_i$. It holds that D is a lower bound for the length of an optimal schedule.

Since there are algorithms known to determine paths for routing the packets (see [24, 16] or simply take shortest paths) we will also consider the packet *routing problem with fixed paths.* An instance of this problem is a tupel $(G, \mathcal{M}, \mathcal{P})$ such that G is a (directed or undirected) graph, $\mathcal{M}$ is a set of packets and $\mathcal{P}$ is a set of predefined paths. Since the paths of the packets are given in advance they do not need to be computed here. The aim is to find a schedule with the properties described above such that the makespan is minimized. For each packet M_i we define D_i to be the length of the path P_i, again assuming that all edges have unit length. Like above we define the *dilation* D by $D := \max_i D_i$. For each edge e we define C_e to be the number of paths that use e. Then we define the *congestion* C by $C := \max_e C_e$. It holds that C and D are lower bounds for the length of an optimal schedule.

1.2. **Related Work.**

Packet routing and related problems have been widely studied in the literature. Di Ianni showed that the delay routing problem [15] is NP-hard. The proof implies that the packet routing problem is NP-hard as well. Leung et al. [19, chapter 37] studied packet routing on different graph classes, including in- and out-trees. In particular, they showed that for those trees the farthest-destination-first (FDF)-algorithm works optimally. In [3] Busch et al. studied the direct routing problem, that is the problem of finding a routing schedule such that a packet is never delayed once it has left its start vertex. They gave complexity results and algorithms for finding direct schedules.

Mansour and Patt-Shamir [20] studied greedy scheduling algorithms (algorithms that always forward a packet if they can) in the setting where the paths of all packets are shortest paths. They proved that in this setting every packet M_i reaches its destination after at most $D_i + N - 1$ steps where D_i is the length of the path of M_i and N is the number of packets in the network. Thus, giving priority to the packets according to the lengths of their paths yields an optimal algorithm if we assume that the path-lengths are pairwise different.

In [17] Leighton et al. showed that there is always a routing schedule that finishes in $O(C + D)$ steps. In [18] Leighton et al. presented an algorithm that finds such a schedule in polynomial time. However, this algorithm is not suitable for practical applications since the hidden constants are very large. There are also some local algorithms for this problem, needing $O(C) + (\log^* N)^{O(\log^* N)} D + \mathit{poly}(\log N)^6$ [23] and $O\left(C + D + \log^{1+\epsilon} N\right)$ [21] steps with high probability (recall that N denotes the number of packets in the network). For the case that all paths are shortest paths, Meyer auf der Heide et al. [2] presented a randomized online routing protocol which needs only $O(C + D + \log N)$ steps with high probability. Using the algorithm by Leighton et. al as a subroutine Srinivasan and Teo [24] presented an algorithm that solves the packet routing problem with variable paths with a constant approximation factor. This algorithm was recently improved by Koch et al. [16] for the more general message routing problem (where each message consists of several packets).

The packet routing problem is closely related to the multi-commodity flow over time problem [12, 7, 8, 13, 14]. In particular, Hall et al. [12] showed that this problem is NP-hard, even in the very restricted case of series-parallel networks. We obtain the the packet routing problem with variable paths if we additionally require unit edge capacities, unit transit times, and integral flow values. If there is only one start and one destination vertex then the packet routing problem is equivalent to the quickest flow problem. It can be solved optimally in polynomial time, e.g., using the Ford-Fulkerson algorithm for the maximum flow over time problem [5, 9, 10] together with a binary search framework.

Adler et al. [1] studied the problem of scheduling as many packets as possible through a given network in a certain time frame. They gave approximation algorithms and NP-hardness results.

1.3. **Our Contributions.** We show that the special case of the packet routing problem is NP-hard to approximate with a factor of $6/5 - \epsilon$ for $\epsilon > 0$. For the packet routing problem restricted to planar graphs and directed trees we show that it is also NP-hard to approximate within factors $7/6 - \epsilon$ and $8/7 - \epsilon$, respectively. These results hold no matter if the paths of the packets are variable or given in advance. All results hold on directed and undirected graphs. We also show that it is NP-hard to approximate the packet routing problem with fixed paths with an absolute error of k for any $k \geq 0$.

For the packet routing problem on directed trees, we show that the FDF-algorithm can perform arbitrarily bad: the obtained schedule can be longer by an arbitrarily large factor than the optimal schedule. However, we give a factor 2-approximation algorithm for undirected trees that uses the FDF-algorithm as a subroutine. For directed trees, we show how to solve the path coloring problem optimally. Having computed such a coloring, we show how to construct a direct schedule whose length is bounded by $C + D - 1$. Then we present an algorithm for directed trees which constructs an optimal direct schedule of length D in the setting that the lengths of the paths of the packets are pairwise different. For all our algorithms we show that the analysis of the respective approximation ratios is tight.

1.4. **Outline of the paper.** In Section 2 we present our complexity results for the packet routing problem on different graph classes. In Section 3 we present algorithms for the packet routing problem on different graph classes. Finally in Section 4 we summarize our results.

2. Complexity results

In this section we show the NP-hardness of the packet routing problem on several graph classes. For some settings we can also prove that it is NP-hard to approximate the problem within certain approximation factors. We distinguish between fixed and variable paths.

2.1. **General Graphs.** We study the complexity of the packet routing problem without imposing any conditions on the graphs.

Theorem 1. *It is NP-hard to approximate the packet routing problem with fixed paths with an approximation factor of* $6/5 - \epsilon$ *for all* $\epsilon > 0$.

Proof. In this proof we employ a technique which was used in [25] for showing that the general acyclic job shop problem is NP-hard to approximate within an approximation factor better than 5/4. We reduce from 3-BOUNDED-3-SAT. In this problem we are given a 3-SAT formula in which each variable occurs at most three times. The question is whether there is an assignment for the variables such that the formula is satisfied. It was shown in [11] that 3-BOUNDED-3-SAT is NP-hard. We assume that we are given a 3-BOUNDED-3-SAT formula with a set of clauses $\{C_1, C_2, ..., C_m\}$. We construct an instance of the packet routing problem corresponding to this formula. We will show that the formula is satisfiable if and only if the length of an optimal schedule is at most 5.

W.l.o.g. we assume that each variable occurs at most two times as a positive literal and at most two times as a negative literal in the formula. If a variable occurs three times positive then we just set it true and thus satisfy all the clauses containing this formula. If the variable occurs three times negative then we set the variable false. We also assume that each clause contains two or three variables.

In the sequel we will explain how we construct our graph and the packets with their start and destination vertices from this formula. Figure 2.1 shows a part of our construction for the formula $(x \vee y \vee z)$. For each variable x we introduce the vertices $v_{x,1}, v_{x,2}, ..., v_{x,11}$. For each clause C there is a vertex v_C and a vertex v'_C . If C contains only two variables there is an additional vertex v''_C. We add all the edges to the graph which are necessary for the packets according to their predefinded paths (we define the paths below).

Now we introduce the packets. For each variable x we have four packets labeled $x_1, \bar{x}_1, x_2, \bar{x}_2$. We will call them the *variable packets* later on. These packets encode whether x is set to true or false in the variable assignment which corresponds to the schedule. The intuition is that a variable x is set to true if x_1 and x_2 are schedules at time $t = 0$ and $\bar{x}_1$ and $\bar{x}_2$ are schedules at time $t = 1$. If $\bar{x}_1$ and $\bar{x}_2$ are schedules at time $t = 0$ and x_1 and x_2 are scheduled at time $t = 1$ then x is set to false. The predefined paths for the variable packets ensure that other schedules for the variable packets would cause the overall schedule to be longer than 5. For each occurence of the variable x in a clause we have a packet. We denote by $c_{x,i}$ the packet for the i-th positive occurence of the variable x and by $c_{\bar{x},i}$ the packet for the i-th negative occurence of the variable x. We will call those packets the *clause*

Packet	Path	
Packets for variables x		
x_1	$v_{x,1}, v_{x,3}, v_{x,5}, v_{x,7}, v_{x,10}$	For each variable x
x_2	$v_{x,2}, v_{x,4}, v_{x,5}, v_{x,6}, v_{x,8}$	For each variable x
$\bar{x}_1$	$v_{x,1}, v_{x,3}, v_{x,5}, v_{x,6}, v_{x,9}$	For each variable x
$\bar{x}_2$	$v_{x,2}, v_{x,4}, v_{x,5}, v_{x,7}, v_{x,11}$	For each variable x
Packets for clauses C with three variables		
$c_{x,1}$	$v_C, v'_C, v_{x,7}, v_{x,10}$	where C contains the first positive occurence of x
$c_{x,2}$	$v_C, v'_C, v_{x,6}, v_{x,8}$	where C contains the second positive occurence of x
$c_{\bar{x},1}$	$v_C, v'_C, v_{x,6}, v_{x,9}$	where C contains the first negative occurence of x
$c_{\bar{x},2}$	$v_C, v'_C, v_{x,7}, v_{x,11}$	where C contains the second negative occurence of x
Packets for clauses C with two variables		
$c_{x,1}$	$v_C, v'_C, v''_C, v_{x,7}, v_{x,10}$	where C contains the first positive occurence of x
$c_{x,2}$	$v_C, v'_C, v''_C, v_{x,6}, v_{x,8}$	where C contains the second positive occurence of x
$c_{\bar{x},1}$	$v_C, v'_C, v''_C, v_{x,6}, v_{x,9}$	where C contains the first negative occurence of x
$c_{\bar{x},2}$	$v_C, v'_C, v''_C, v_{x,7}, v_{x,11}$	where C contains the second negative occurence of x

TABLE 1. The predefined paths of the packets.

packets. In a satisfying variable assignment there must be at least one variable x (or $\bar{x}$ respectively) in each clause C which satisfies C. The intuition is that the packet $c_{x,j}$ (or $c_{\bar{x},j}$ respectively) corresponding to this variable x (or $\bar{x}$ respectively) is scheduled last (at time $t = 2$). In Table 1 we define the predefined paths of the packets.

Now we want to prove that the length of an optimal schedule is at most 5 if and only if the formula is satisfiable. For the first part we assume that our formula is satisfiable. We consider a variable assignment such that the formula is satisfied. We construct a schedule as follows. For each variable x which is set true in the satisfying variable assignment we schedule the packets x_1 and x_2 at time 0 and the packets $\bar{x}_1$ and $\bar{x}_2$ are scheduled at time 1. For each false variable y we schedule the packets y_1 and y_2 at time 1 and the packets $\bar{y}_1$ and $\bar{y}_2$ at time 0. Since our assignment of the variables satisfies the formula, for each clause C there has to be one literal x which satisfies the clause. We schedule the packets corresponding to the literals in C in such a way that the packet corresponding to the literal x leaves v_C at time $t = 2$. We schedule the two other packets of C in an arbitrary order at times $t = 0$ and $t = 1$.

Occuring collisions (i.e., situations where two or more packets need to use the same edge at a time) are resolved arbitrarily. We want to show that the length of this schedule is 5. Let x_i be a variable packet which leaves its start vertex at time $t = 0$. It can be delayed at most once in our schedule, namely by a clause packet which leaves its start vertex at time $t = 1$. Thus x_i reaches its destination vertex after at most 5 timesteps. Now let x_i be a variable packet which left its start vertex at time $t = 1$. This implies that x was set to false in the variable assignment. Thus, there can be no clause packet $c_{x,i}$ which left its start vertex at time $t = 2$. Therefore, x_i will not be delayed on its way and will reach its destination vertex after 5 timesteps. For "negative" variable packets $\bar{x}_i$ we can apply the same reasoning. Now let $c_{x,i}$ be a clause packet of a clause with three literals. If $c_{x,i}$ was scheduled

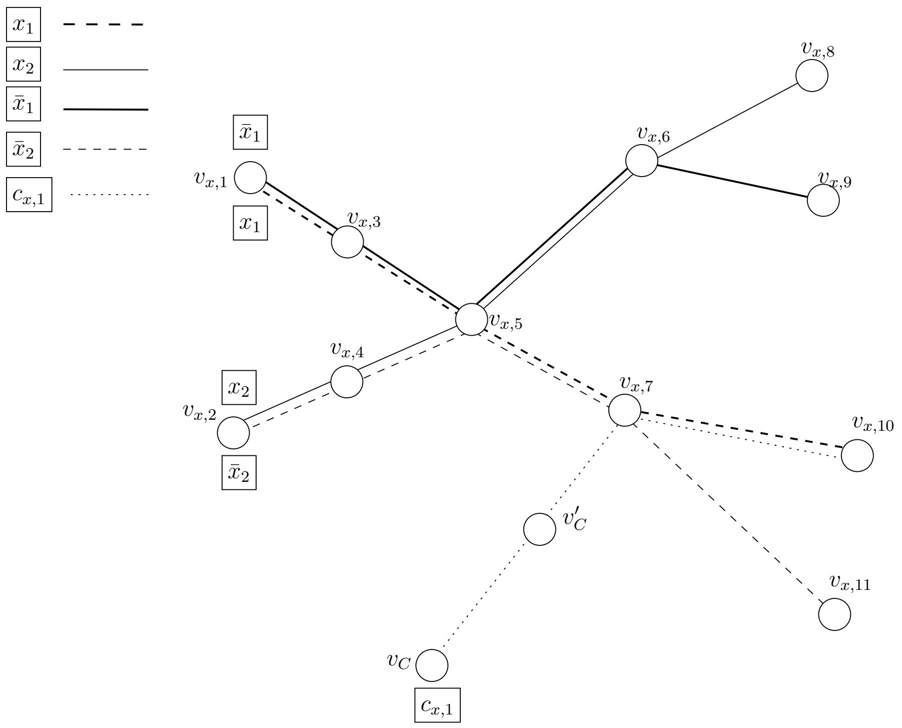

FIGURE 2.1. A part of the graph for the clause $(x \vee y \vee z)$. The different lines represent the ways of the packets through the network. The parts corresponding to the variables y and z and their packets are omitted for brevity.

at time $t = 0$ it will not be delayed at all and will reach its destination after 3 timesteps. If $c_{x,i}$ was scheduled at time $t = 1$ it can be delayed by at most one variable packet (by at most one timestep) and will therefore reach its destination after at most 5 timesteps. Finally, if $c_{x,i}$ was scheduled at time $t = 2$ we know that x was set true in the variable assignment. This implies that the packets x_1 and x_2 were scheduled at time $t = 0$ and will not be able to delay $c_{x,i}$. Thus, $c_{x,i}$ will reach its destination vertex at time $t = 5$. We can apply the same reasoning to clause packets $c_{\bar{x},i}$ for negated literals. For clause packets corresponding to clauses with only two literals we can apply a similar reasoning: The two packets behave like the last two packets which are scheduled in a vertex v_C where C is a clause with three literals.

Now we assume that there is a schedule of length 5 or less. We want to show that the formula is satisfiable. In order to do this, we show how we can construct a satisfying variable assignment from the schedule. W.l.o.g. we assume that the schedule never delays a packet if it is the only packet that wants to use an edge at a time. Consider a variable x and its packets $x_1, x_2, \bar{x}_1$, and $\bar{x}_2$. Since x_1 and $\bar{x}_1$ have the same start vertex, either x_1 is scheduled at time 0 and $\bar{x}_1$ is scheduled at time 1 or vice versa. The same holds for x_2 and $\bar{x}_2$. Since the length of the schedule is 5, we see that those of the packets which are scheduled at time 1 are not delayed later in the schedule. From this it follows that x_1 and x_2 are scheduled at the same time. If they are scheduled at time 0 we set the variable x to true, otherwise we set x to false. We do this with all variables. Now we want to show that this leads to a satisfying assignment for the formula.

Let C be a clause. We discuss the case where C contains only positive literals. The other cases can be proven similarly. Let $C = (x \vee y \vee z)$. Consider the three packets $c_{x,i}$, $c_{y,j}$, $c_{z,k}$ for C. They all originate at the same vertex v_C. We consider the packet of the three which was scheduled last (i.e., at time $t = 2$). W.l.o.g. let $c_{x,i}$ be this packet. Since the length of our schedule is 5 we conclude that $c_{x,i}$ was never delayed after time $t = 2$. Thus, the packet x_i must have been scheduled at time $t = 0$. Otherwise both x_i and $c_{x,i}$ would reach the vertex $v_{x,6}$ (or $v_{x,7}$ respectively) at time $t = 4$ and the total makespan of the schedule would be 6. This is a contradiction.

This implies that we set the variable x to true in our variable assignment. Therefore, the clause C is satisfied. Doing this reasoning for all clauses shows that the formula is satisfied. □

As corollaries we obtain the following results for the packet routing problem with fixed paths on directed graphs and for the packet routing problem with variable paths.

Corollary 2. *It is NP-hard to approximate the packet routing problem with fixed paths on directed graphs with an approximation factor of* $6/5 - \epsilon$ *for all* $\epsilon > 0$.

Proof. It is straight forward to see that we can orientate the edges of the underlying graph in the construction of Theorem 1 such that the packets always move in the direction of the edges (see Figure 2.2).

□

Corollary 3. *It is NP-hard to approximate the packet routing problem with variable paths on directed graphs within an approximation factor of* $6/5 - \epsilon$ *for all* $\epsilon > 0$.

Proof. Consider the construction used in Theorem 1. It turns out that if the graph is directed the path for each packet is unique. Therefore, allowing the packets to choose their paths does not change anything.

Now we show that all paths are unique. Let x be a variable. We orientate the edges in the direction in which the packets move. See Figure 2.2 for a sketch. Denote by G_x the subgraph induced by the vertices $v_{x,1}, ..., v_{x,11}$. There are no edges oriented out of G_x. Since G_x is a tree, it holds that once a packet has reached G_x the remainder of its path is unique. So the path of each variable packet is unique. Now let C be a clause and let p_C be a clause packet corresponding to C. In the schedule the packet p_C passes the vertices v_C, v'_C (and v''_C if C contains only two variables). After this p_C has to move to a vertex v_y in a subgraph G_y of a variable y. Since there is no directed way out of G_y and there is exactly one variable y such that the destination vertex of p_C is contained in G_y the path of of p_C is unique. Therefore, the paths of all clause packets are unique. □

Corollary 4. *It is NP-hard to approximate the packet routing problem with variable paths on undirected graphs with an approximation factor of* $6/5 - \epsilon$ *for all* $\epsilon > 0$.

Proof. It turns out that in the construction of Theorem 1 the packets cannot take advantage of being allowed to use paths different than the ones specified in the reduction.

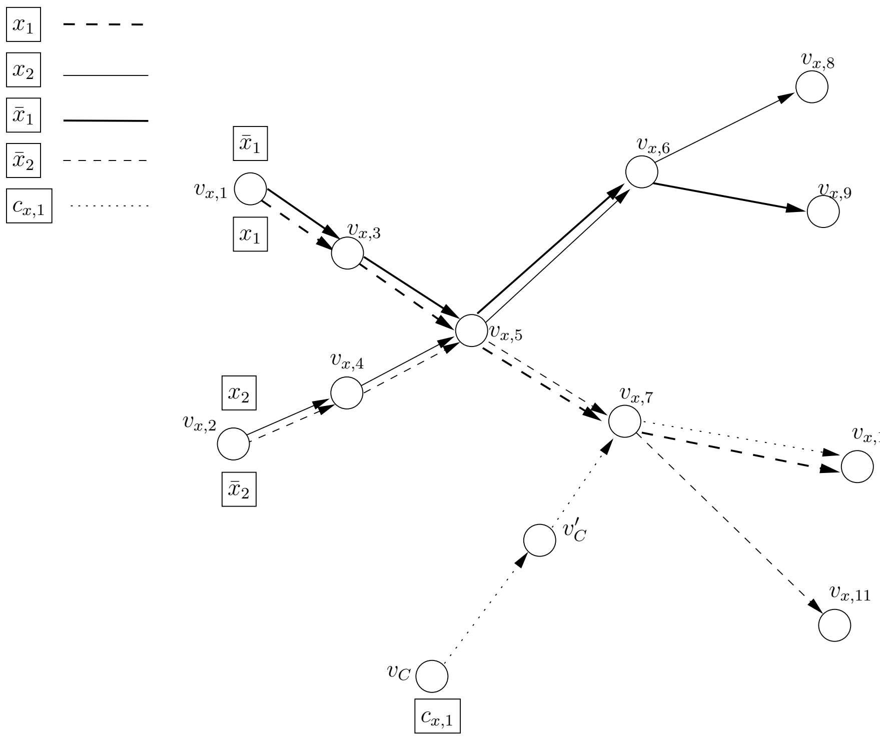

FIGURE 2.2. A part of the directed graph for the clause $(x \vee y \vee z)$.

We use the same reduction as defined in Theorem 1. For each packet we set its start vertex and destination vertex as defined in Table 1. Since we study the packet routing problem with variable paths the latter can be chosen arbitrarily by the schedule. We want to prove that the formula is satisfiable if and only if the optimal schedule has length 5. Assume that the formula is satisfiable. In the proof of Theorem 1 it was shown that there exists a schedule of length 5 which uses the paths specified in Table 1. Now assume that there is a schedule of length 5. For all variable packets and all clause packets it holds that all paths to their respective destination vertices different from the ones defined in Table 1 need more than 5 edges. Therefore, the packets have to move according to the paths defined in the table. The remainder of the proof is exactly as in Theorem 1. □

If there is only one start and one destination vertex and the paths of the packets are variable, the packet routing problem is solvable in polynomial time, e.g., using the Ford-Fulkerson algorithm for the maximum flow over time problem [5, 9, 10] together with a binary search framework. However, if the paths of the packets are fixed the problem becomes NP-hard.

Corollary 5. *It is NP-hard to approximate the packet routing problem with fixed paths on directed or undirected graphs with one start or one destination with an approximation factor of $7/6 - \epsilon$ for all $\epsilon > 0$. If there is a single start and a single destination vertex it is NP-hard to approximate the problem within a factor of $8/7 - \epsilon$ for all $\epsilon > 0$.*

Proof. We take the construction of Theorem 1 and add a super start-vertex v_s. We connect v_s with each start vertex v of the graph (one edge per packet that starts in v). For each packet we set v_s to be its start vertex and define the rest of its path according to Table 1. Thus, the lengths of all paths are increased by one. This proves the first claim. We can do the same for the case of only one destination vertex by introducing a super-destination vertex and obtain the same hardness result for this setting. Doing both (adding a super-start vertex and a super-destination vertex) shows that it is NP-hard to approximate the packet routing problem with fixed paths on undirected graphs with one start and one destination vertex with a factor of $8/7 - \epsilon$ for all $\epsilon > 0$. In all these constructions we can orientate the edges such that all the packets always move in the direction of the edges (see also Corollary 2). □

2.2. **Planar Graphs.** The part of the graph shown in Figure 2.2 is in fact a tree. However, the whole construction is in general not even planar. E.g., consider the formula $(x \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z})$. Figure 2.3 shows a sketch of the graph constructed according to the construction in Theorem 1. We want to show that it contains a $K_{3,3}$-minor which implies that it is not planar. We define $A := \{v'_1, v'_2, v'_3\}$ and let $B := \{v_{x,6}, v_{y,7}, v_{z,6}\}$. We can see that there are edge-disjoint paths between all pairs of vertices (v_a, v_b) with $v_a \in A$ and $v_b \in B$. Therefore, the graph contains a $K_{3,3}$-minor and is not planar.

Even though the construction in Theorem 1 is not planar, it can be easily adjusted to obtain a planar graph. However, this is to the cost of a slightly worse bound for the approximation ratio.

Theorem 6. *On planar graphs it is NP-hard to approximate the packet routing problem with fixed paths with an approximation factor of $7/6 - \epsilon$ for any $\epsilon > 0$.*

Proof. We change the construction used in Theorem 1 slightly in order to obtain a planar graph. We introduce the vertices v_L (L for Left) and v_R (R for right). We change the predefined paths of the packets such that all packets pass either v_L or v_R. This extends the lengths of the path by one edge in comparison with the construction in Theorem 1. Figure 2.4 shows a sketch of this construction.

Now we describe the construction in detail. In addition to the vertices defined in the proof of Theorem 1 we introduce the vertices v_L, v_R and vertices $v'_{1,x}$ and $v'_{2,x}$ for each variable x. The predefined paths of the packets are shown in Table 2. From the sketch in Figure 2.4 it is easy to see that the graph is planar.

Similar to Theorem 1 it holds that the length of an optimal schedule for this network is 6 if and only if the formula is satisfiable. The proof works along the lines of the proof of Theorem 1. Assume that the formula is satisfiable. We schedule the packets with the same technique as in the proof of Theorem 1. However, here we must be more careful with the schedule of the packets of a clause C which are not needed in order to satisfy C. There might be a variable x such that two of those packets want to use the edge $\{v_L, v_{x,6}\}$ or $\{v_R, v_{x,7}\}$ at time $t = 3$ (this happens when both packets startet at time $t = 1$). We show that there is always a schedule in which this does not happen. We consider a graph $G' = (V', E')$. The set V' contains one vertex for each literal in a clause which is not needed in order to satisfy its respective clause. Two vertices are connected if and only if their respective literals they are in the same clause or the paths of their respective packets share an edge. Note that this implies that the degree of a vertex is at most

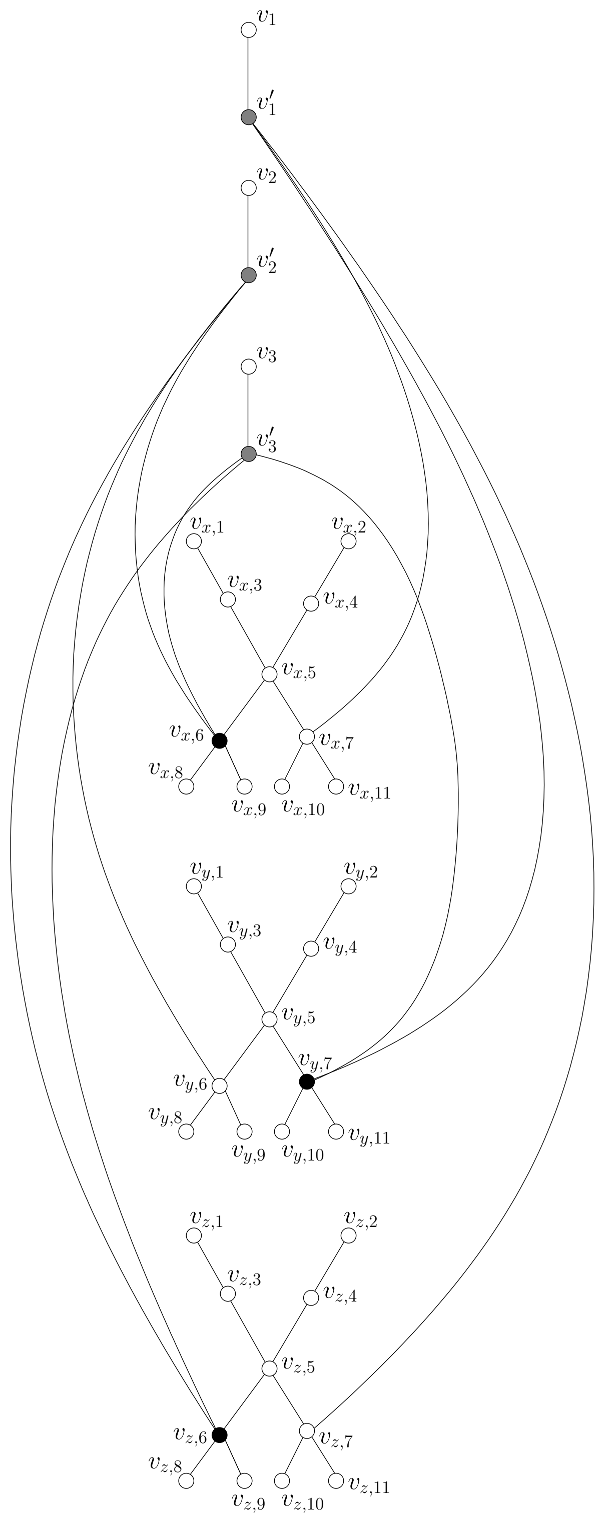

FIGURE 2.3. In general the graphs constructed in the proof of Theorem 1. The gray and black vertices form the sets A and B for the $K_{3,3}$-minor.

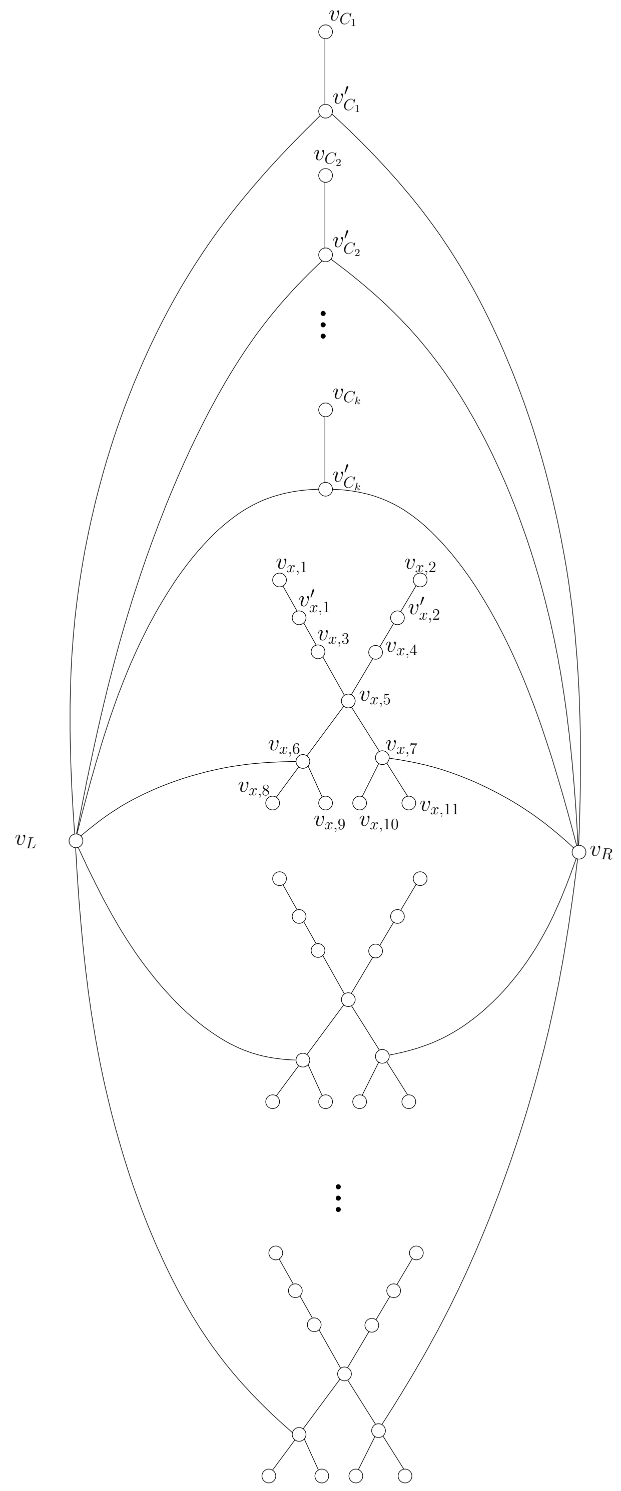

FIGURE 2.4. Sketch of a planar embedding of the construction in Theorem 6.

Packet	Path	
Packets for variablex x		
x_1	$v_{x,1}, v'_{x,1}, v_{x,3}, v_{x,5}, v_{x,7}, v_{x,10}$	For each variable x
x_2	$v_{x,2}, v'_{x,2}, v_{x,4}, v_{x,5}, v_{x,6}, v_{x,8}$	For each variable x
$\bar{x}_1$	$v_{x,1}, v'_{x,1}, v_{x,3}, v_{x,5}, v_{x,6}, v_{x,9}$	For each variable x
$\bar{x}_2$	$v_{x,2}, v'_{x,2}, v_{x,4}, v_{x,5}, v_{x,7}, v_{x,11}$	For each variable x
Packets for clauses C with three variables		
$c_{x,1}$	$v_C, v'_C, v_L, v_{x,7}, v_{x,10}$	where C contains the first positive occurence of x
$c_{x,2}$	$v_C, v'_C, v_R, v_{x,6}, v_{x,8}$	where C contains the second positive occurence of x
$c_{\bar{x},1}$	$v_C, v'_C, v_R, v_{x,6}, v_{x,9}$	where C contains the first negative occurence of x
$c_{\bar{x},2}$	$v_C, v'_C, v_L, v_{x,7}, v_{x,11}$	where C contains the second negative occurence of x
Packets for clauses C with two variables		
$c_{x,1}$	$v_C, v'_C, v, v_L, v_{x,7}, v_{x,11}$	where C contains the first positive occurence of x
$c_{x,2}$	$v_C, v'_C, v''_C, v_R, v_{x,6}, v_{x,11}$	where C contains the second positive occurence of x
$c_{\bar{x},1}$	$v_C, v'_C, v''_C, v_R, v_{x,6}, v_{x,11}$	where C contains the first negative occurence of x
$c_{\bar{x},2}$	$v_C, v'_C, v''_C, v_L, v_{x,7}, v_{x,11}$	where C contains the second negative occurence of x

TABLE 2. The predefined paths of the packets.

two. Let $V'' \subseteq V'$ be the set of vertices corresponding to packets in clauses with only two literals. It holds that the degree of vertices in V'' is at most one.

We want to show that G' is bipartite. Let c be a cycle in G'. Since the degree of each vertex in G' is at most two the cycle c must be a connected component of G'. Moreover, it cannot contain vertices in V''. For each clause C which contributes vertices to c both vertices must be in c. This implies that c is has even length and thus G' is bipartite. Therefore, we can color its vertices with two colors, say 0 and 1. Then we schedule the corresponding packets according to the color of the vertices at times $t = 0$ and $t = 1$. This ensures that there is no collision of two packets when using edges $\{v_L, v_{x,6}\}$ or $\{v_R, v_{x,7}\}$ for a variable x. Thus, we can guarantee that the length of the overall schedule is at most 6.

The proof for constructing a satisfying variable assignment out of a schedule of length 6 works exactly as in the proof of Theorem 1. □

Corollary 7. *On undirected planar graphs it is NP-hard to approximate the packet routing problem with variable paths with an approximation factor of* $7/6 - \epsilon$ *for any* $\epsilon > 0$.

Proof. Like in Corollary 4 it turns out that the packets cannot take advantage of being able to choose a path different from the one predefined in Table 2. This can be proven similarly as in the proof of Corollary 4. □

2.3. **Trees.**

We can show that the packet routing problem is still NP-hard to approximate if we restrict the considered graphs to directed trees.

Theorem 8. *It is NP-hard to approximate the packet routing problem on directed trees with an approximation ratio of* $8/7 - \epsilon$ *for any* $\epsilon > 0$.

Proof. We reduce from 3-BOUNDED-3-SAT. Our construction is a slightly changed version of the construction in Theorem 1. Since the graph is a tree the bound on the approximation ratio that we can prove is slightly worse.

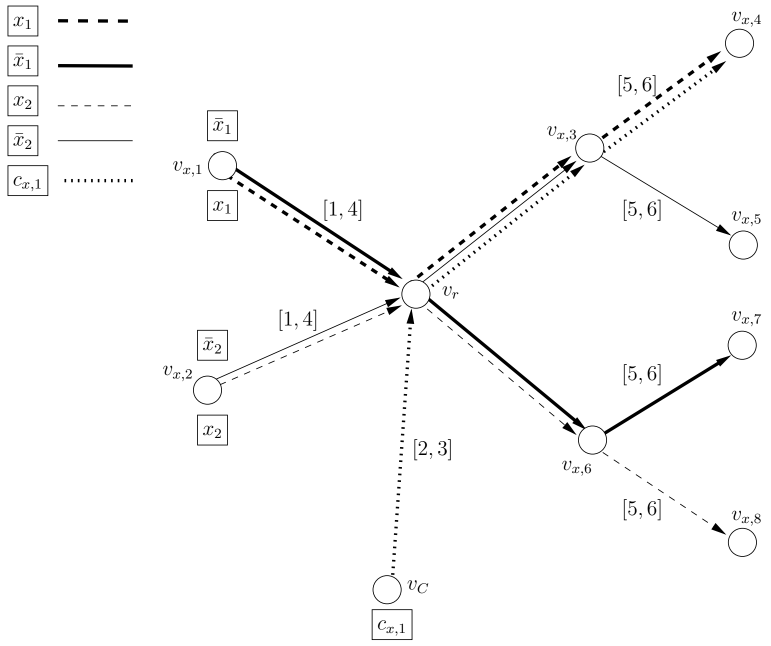

FIGURE 2.5. A part of the graph for the clause $C = (x \vee y \vee z)$ (construction for Theorem 8). The parts corresponding to the variables y and z and their packets are omitted for brevity. The time intervals at the edges denote the time when the respective edge is blocked. Note that the first timestep is $t = 0$.

Packet	Path	
Variable packets for variable x		
x_1	$v_{x,1}, v_r, v_{x,3}, v_{x,4}$	For each variable x
x_2	$v_{x,2}, v_r, v_{x,6}, v_{x,8}$	For each variable x
$\bar{x}_1$	$v_{x,1}, v_r, v_{x,6}, v_{x,7}$	For each variable x
$\bar{x}_2$	$v_{x,2}, v_r, v_{x,3}, v_{x,5}$	For each variable x
Clause packets for clause C		
$c_{x,1}$	$v_C, v_r, v_{x,3}, v_{x,4}$	where C contains the first positive occurence of x
$c_{x,2}$	$v_C, v_r, v_{x,6}, v_{x,8}$	where C contains the second positive occurence of x
$c_{\bar{x},1}$	$v_C, v_r, v_{x,6}, v_{x,7}$	where C contains the first negative occurence of x
$c_{\bar{x},2}$	$v_C, v_r, v_{x,3}, v_{x,5}$	where C contains the second negative occurence of x

TABLE 3. The predefined paths of the packets.

Assume we are given a 3-BOUNDED-3-SAT formula ϕ. We construct an instance of the packet routing problem with fixed paths such that the underlying graph is a directed tree. In particular, the packets always move in the direction of the edges. We show that the formula is satisfiable if and only if the optimal schedule has length 7. Like in the proof of Theorem 1, we assume that each clause contains at least two variables and that each variable occurs at most two times positive and at most two times negated in ϕ. The construction is sketched in Figure 2.5. The main difference in comparison with Theorem 1 is that now there is a central vertex v_r that all packets pass. For each variable x in the formula, there are the four variable packets x_1, x_2, $\bar{x}_1$ and $\bar{x}_2$. They can be scheduled at time $t = 0$ or $t = 4$. The intuition is that if x_1 and x_2 are scheduled at time $t = 0$ then we set the variable x true, if x_1 and x_2 are scheduled at time $t = 4$ then we set the variable x false. Note that now it could happen that this assignment is not well-defined, i.e., the packets x_1 and $\bar{x}_2$ go first or the packets x_2 and $\bar{x}_1$ go first. Like in the proof of Theorem 1 our construction ensures that this leads to a schedule which is strictly longer than 7 timesteps. Moreover, for each literal in a clause C there is one packet. Packets corresponding to literals in the same clause share the first edge on their path. Given a satisfying variable assignment for ϕ there is at least one literal in C which satisfies the clause. The packet which corresponds to this literal is scheduled last.

Now we describe the construction in detail. For each variable x we introduce vertices $v_{x,1}, v_{x,2}, ..., v_{x,8}$. We also introduce the four *variable packets* x_1, x_2, $\bar{x}_1$, and $\bar{x}_2$. In the formula, we fix an order of the clauses. For each clause C there is a vertex v_C. For each literal in C there is one *clause packet*. We write $c_{x,i}$ for the packet corresponding to the i-th positive occurence of x and $c_{\bar{x},i}$ for the packet corresponding to the i-th negative occurence of x. The predefined paths for the packets are shown in Table 3 and sketched in Figure 2.5. Note that the underlying graph is a directed tree.

In order to make the construction work as desired we want to block some edges at certain times. This can be done by introducing *blocking packets*. If we want to block an edge e at time $t = \bar{t}$ we introduce a blocking packet $b_{e,\bar{t}}$ which needs to use e at time $t = \bar{t}$ or otherwise would not reach its destination before time $t = 7$. Note that this does not destroy the tree structure of the underlying graph. In order to clearify matters we do not explicitely define the paths of the blocking packets. Instead, we state which edges are blocked at what time. Let x be a variable. The edges $(v_{x,1}, v_r)$ and $(v_{x,2}, v_r)$ are blocked during the time interval $[1, 4]$ (note that $t = 0$ is the first timestep). The edges $(v_{x,3}, v_{x,4})$, $(v_{x,3}, v_{x,5})$, $(v_{x,6}, v_{x,7})$, and $(v_{x,6}, v_{x,8})$ are blocked during the time interval $[5, 6]$. For each clause C with three literals the edge (v_C, v_r) is blocked during the time interval $[2, 3]$. For each clause C' with two literals the edge $(v_{C'}, v_r)$ is blocked during the time interval $[1, 3]$.

Now we want to prove that ϕ is satisfiable if and only if the optimal schedule has length 7. First assume that ϕ is satisfiable. We consider a variable assignment that satisfies the formula and schedule the packets as follows: Everytime there is a packet M that is the only packet that needs to use the next edge on its path we move M. Let x be a variable. If x is set true in our variable assignment then we schedule the packets x_1 and x_2 at time $t = 0$ and the packets $\bar{x}_1$ and $\bar{x}_2$ at time $t = 4$. If x is set false then we schedule x_1 and x_2 at time $t = 4$ and $\bar{x}_1$ and $\bar{x}_2$ at time $t = 0$.

Since our variable assignment satisfies the formula in each clause C there must be at least one literal y (or $\bar{y}$ respectively) which satisfies the clause. We schedule the packet corresponding to y (or $\bar{y}$ respectively) to leave v_C at time $t = 3$. We schedule the other two packets to leave v_C in arbitrary order at times $t = 0$ and $t = 1$ (if C contains only two literals the other packet is scheduled at time $t = 0$). Blocking packets are never delayed by other packets. If there is a variable packet and a clause packet competing for using an edge at the same time we give priority to the variable packet (this is only for simplification of the proof). Other ties are broken arbitrarily.

Now we show that all packets arrive at their destination vertices after at most 7 timesteps. First assume that x_1 started at time $t = 0$. Since the edge $(v_{x,2}, v_r)$ is blocked during the time interval $[1, 4]$ the packet $\bar{x}_2$ cannot reach v_r before $t = 5$. Thus, x_1 reaches $v_{x,6}$ at time $t = 2$ and $v_{x,8}$ at time $t = 3$. For the case that the packets x_2, $\bar{x}_1$, or $\bar{x}_2$ startet at time $t = 0$ the proof works similarly.

Now assume that $\bar{x}_2$ startet at time $t = 4$. The reasoning above shows that $\bar{x}_2$ cannot collide with x_1. Since variable packets have a higher priority than clause packets we conclude that $\bar{x}_2$ reaches $v_{x,6}$ at time $t = 6$ and $v_{x,8}$ at time $t = 7$. For the case that the packets x_1, x_2, or $\bar{x}_1$ startet at time $t = 4$ the proof works similarly.

Next we discuss the clause packets. First assume that the packet $c_{x,1}$ was scheduled at time $t = 0$ or $t = 1$. In both cases it reaches $v_{x,6}$ at time $t = 4$ at the latest and $v_{x,8}$ at time $t = 5$ at the latest. (Note that either the packet x_1 or the packet $\bar{x}_2$ uses the edge $(v_r, v_{x,6})$ at time $t = 1$ and thus it does not make a difference for our reasoning if $c_{x,1}$ was scheduled at time $t = 0$ or $t = 1$). For the other clause packets which were scheduled at time $t = 0$ or $t = 1$ the proofs works similarly.

Now assume that the clause packet $c_{\bar{x},2}$ was scheduled at time $t = 3$. This implies that in the satisfying variable assignment x was set false. Then $c_{\bar{x},2}$ reaches v_r at time $t = 4$. From the above reasoning it follows that it cannot collide with any other packet at v_r. Moreover, it reaches $v_{x,6}$ at time $t = 5$. The edge $(v_{x,6}, v_{x,8})$ is blocked in the time interval $[5, 6]$. The packet $\bar{x}_2$ is the only packet that $c_{\bar{x},2}$ competes with for using the edge $(v_{x,6}, v_{x,8})$. But since x was set false in our variable assignment $\bar{x}_2$ startet at time $t = 0$ and the above reasoning shows that $\bar{x}_2$ reached $v_{x,8}$ at time $t = 3$. Thus, $c_{\bar{x},2}$ reaches $v_{x,8}$ at time $t = 7$. This shows that all packets reach their respective destination vertices after at most 7 timesteps.

Now assume that there is a schedule S of length 7. We want to show that there is a variable assignment which satisfies ϕ. Let x be a variable. If the packets x_1 and x_2 were both scheduled at time $t = 0$ then we set x to true, if the packet $\bar{x}_1$ and $\bar{x}_2$ were both scheduled at time $t = 0$ we set x to false. We will show in the sequel that this assignment is well-defined and that it satisfies ϕ. First we show that in S either x_1 and x_2 were both scheduled at time $t = 0$ or $\bar{x}_1$ and $\bar{x}_2$ were both scheduled at time $t = 0$. Assume on the contrary that x_1 and $\bar{x}_2$ were scheduled at time $t = 0$. Since the edges $(v_{x,1}, v_r)$ and $(v_{x,2}, v_r)$ are blocked in the time interval $[1, 4]$ and S has length 7 this implies that $\bar{x}_1$ and x_2 were scheduled at time $t = 4$. However, this implies that both packets arrive at v_r at time $t = 5$. Since both need to use the edge $(v_r, v_{x,3})$ next this contradicts that S has length 7. The case that $\bar{x}_1$ and x_2 were scheduled at time $t = 0$ can be proven similarly. Now we prove that our variable assignment satisfies ϕ. Consider a clause C with three literals (the case that C contains only two literals can be proven similarly). Since the edge (v_C, v_r) is blocked during the time interval $[2, 3]$ there must be at least one packet corresponding to C which was scheduled at time $t = 3$ or later. Let $c_{x,1}$ be this packet (the cases for the other packets can be proven similarly). We want to show that the packet x_1 was scheduled at time $t = 0$ and thus x is set true in our variable assignment. Assume on the contrary that x_1 was scheduled at time $t = 4$ or later. Then it can arrive at $v_{x,6}$ at time $t = 6$ the earliest. Since $c_{x,1}$ was scheduled at time $t = 3$ it can arrive at the vertex $v_{x,6}$ at time $t = 5$ the earliest. The edge $(v_{x,6}, v_{x,8})$ is blocked in the time interval $[5, 6]$ and thus $c_{x,1}$ cannot have reached $v_{x,8}$ by time $t = 6$. Thus, in order to reach $v_{x,8}$ at time $t = 7$ the packets $c_{x,1}$ and x_1 must use the edge $(v_{x,6}, v_{x,8})$ at time $t = 6$. This is a contradiction. Thus, x_1 was scheduled at time $t = 0$ and we set x true. Doing this reasoning for all clauses C shows that our variable assignment satisfies ϕ. □

Note that the above result implies that it is also NP-hard to approximate the packet routing problem on undirected trees with a performance ratio of $8/7 - \epsilon$ for all $\epsilon > 0$.

2.4. Absolute approximation.

All NP-hardness results that we presented so far constructed reductions with a gap of one time unit between yes- and no-instances of 3-BOUNDED-3-SAT. This raises the question whether it is NP-hard to approximate the packet routing problem with an *absolute* error. We answer this question in the affirmative.

Theorem 9. *It is NP-hard to approximate the packet routing problem with fixed paths on directed planar graphs with an absolut error of k.*

Proof. We give a reduction from 3-BOUNDED-3-SAT. Let ϕ be a 3-BOUNDED-3-SAT formula. We show by induction that for each k there is an integer α_k and an instance I_k of the packet routing problem with fixed paths on directed planar graphs with the properties that

- $OPT\,(I_k) = \alpha_k$ if ϕ is satisfiable and
- $OPT\,(I_k) = \alpha_k + k + 1$ if ϕ is not satisfiable

In order to clearify matters, we construct I_k such that each packet has its own start and its own destination vertex. For I_0 we take the construction which was used in the proof of Theorem 6 and introduce two additional vertices for each packet such that the start and destination vertices are unique. This increases the makespan by two. This implies that $\alpha_0 = 8$. Let n_0 be the number of packets that are used in this construction.

For the inductive step we assume that the claim is true for all $i \leq k$. We want to construct a packet routing instance I_{k+1} with the above properties. A sketch of the construction is given in Figure 2.6. Let n_k be the number of packets used in I_k. We denote by α_{k+1} is the length of the optimal makespan for I_{k+1} assuming that ϕ is satisfiable. In order to meet the properties stated above we want that an optimal schedule needs at least $\alpha_{k+1} + (k + 1) + 1$ steps if ϕ is not satisfiable.

We will use n_k copies F_i of I_k and denote them as *formula gadgets.* The intuition is the following: If ϕ is satisfiable then all packets in all formula gadgets arrive at their destination vertices at time $t = \alpha_k$. If ϕ is not satisfiable then in each gadget there is at least one packet that reaches its destination vertex at time $t = \alpha_k + k + 1$. Thus, there is an absolut difference of $k + 1$ between yes- and no-instances. We want to increase this difference to $k + 2$.

In case that ϕ is not satisfiable we want to control which packet from a formula gadget is late (i.e., reaches its destination vertex at time $t = \alpha_k + k + 1$). In order to achieve this we introduce another type of gadget which we call the *sorting gadget.* We place one sorting gadget behind each formula gadget. A sorting gadget works as follows:

Its input consists of all packets from a formula gadget and an additional *sorting packet*. If all packets from the formula gadget left it at time $t = \alpha_k$ then inside the sorting gadget no packets is delayed. If there is a non-empty set of packets P which left the formula gadget at time $t = \alpha_k + k + 1$ then in the sorting gadget either one packet in P is delayed once or the sorting packet is delayed once. If a packet in P is delayed inside the sorting gadget then we can already guarantee that the overall makespan has to be at least $\alpha_{k+1} + k + 2$. So for the remainder of the proof we assume that if ϕ is not satisfiable it will be the sorting packet that will be delayed inside the sorting gadget.

Finally we place an additional copy $\bar{F}$ of I_k. We will call this the *final gadget.* The input of the final gadget consists of all sorting packets. The intuition is that if ϕ is satisfiable no sorting packet will be delayed inside the sorting gadgets. Moreover, all sorting packets will need only α_k additional steps inside the final gadget.

If ϕ is not satisfiable each sorting packet will be delayed once in the sorting gadget and one of them will need $\alpha_k + k + 1$ steps inside the final gadget. This causes an absolute difference of $k + 2$ between yes- and no-instances.

Now we describe the construction in detail. Let $F_1, ..., F_{n_k}$ be the n_k copies of I_k (the formula gadgets). We know that ϕ is satisfiable if and only if in an optimal schedule all packets leave the formula gadgets after at most α_k timesteps. We call the packets used in formula gadgets the *formula packets.*

For a formula packet m_j denote by s_j its start vertex and by u_j its first vertex behind in the formula gadget (i.e., the first vertex outside the gadget), see Figure 2.6. Behind each formula gadget F_j we place a sorting gadget S_j. The input of the sorting gadget S_j consists of all packets leaving F_j and an additional sorting packet q_j. Figure 2.7 depicts a sketch of a sorting gadget. For a packet p_j denote by w_j the first vertex outside of the sorting gadget. Inside the sorting gadget the formula packets move on horizontal lines consisting of $2n_k$ vertices without interfering with each other. The sorting packet first moves $\alpha_k + k + 4$ edges (this can be understood as an artifical delay). Then it intersects the paths of each formula packet in one edge. After having left the sorting gadget the formula packets move on a path of additional $\alpha_k + k + 4$ vertices (there is one path for each packet, so the packets do not interfere with each other). The paths of the sorting packets are then connected to the final gadget $\bar{F}$ which is another copy of I_k. The entire construction is shown in Figure 2.6. We set $\alpha_{k+1} := 2n_k + 2 \cdot \alpha_k + k + 2$.

Now we want to prove that if ϕ is satisfiable then there is a schedule whose length is at most α_{k+1}. We construct a schedule with that length. In this schedule we never delay a packet if not necessary, i.e., if it is the only packet that needs to use its next edge. Since ϕ is satisfiable there is a schedule such that after α_k timesteps all formula packets p_j have reached their respective vertex u_j. Inside the sorting gadgets no packet encounters any delay. Therefore, the sorting packet reaches its vertex w_j after $(\alpha_k + k + 2) + (2n_k + 1) = 2n_k + \alpha_k + k + 3$ time steps. After another timestep it reaches its last vertex before entering $\bar{F}$. Since ϕ is satisfiable there is a schedule such that after another α_k timesteps, the sorting packet reaches its destination vertex. Thus, after $2n_k + 2\alpha_k + k + 4 = \alpha_{k+1}$ timesteps all sorting packets have reached their destination vertices. A formula packet p_j reaches w_j after $\alpha_k + 2n_k$ steps. Since after this it has to move another $\alpha_k + k + 4$ edges p_j has reaches its destination vertex t_j after at most $2n_k + 2 \cdot \alpha_k + k + 2 = \alpha_{k+1}$ timesteps. Thus, the length of the overall makespan is at most α_{k+1}.

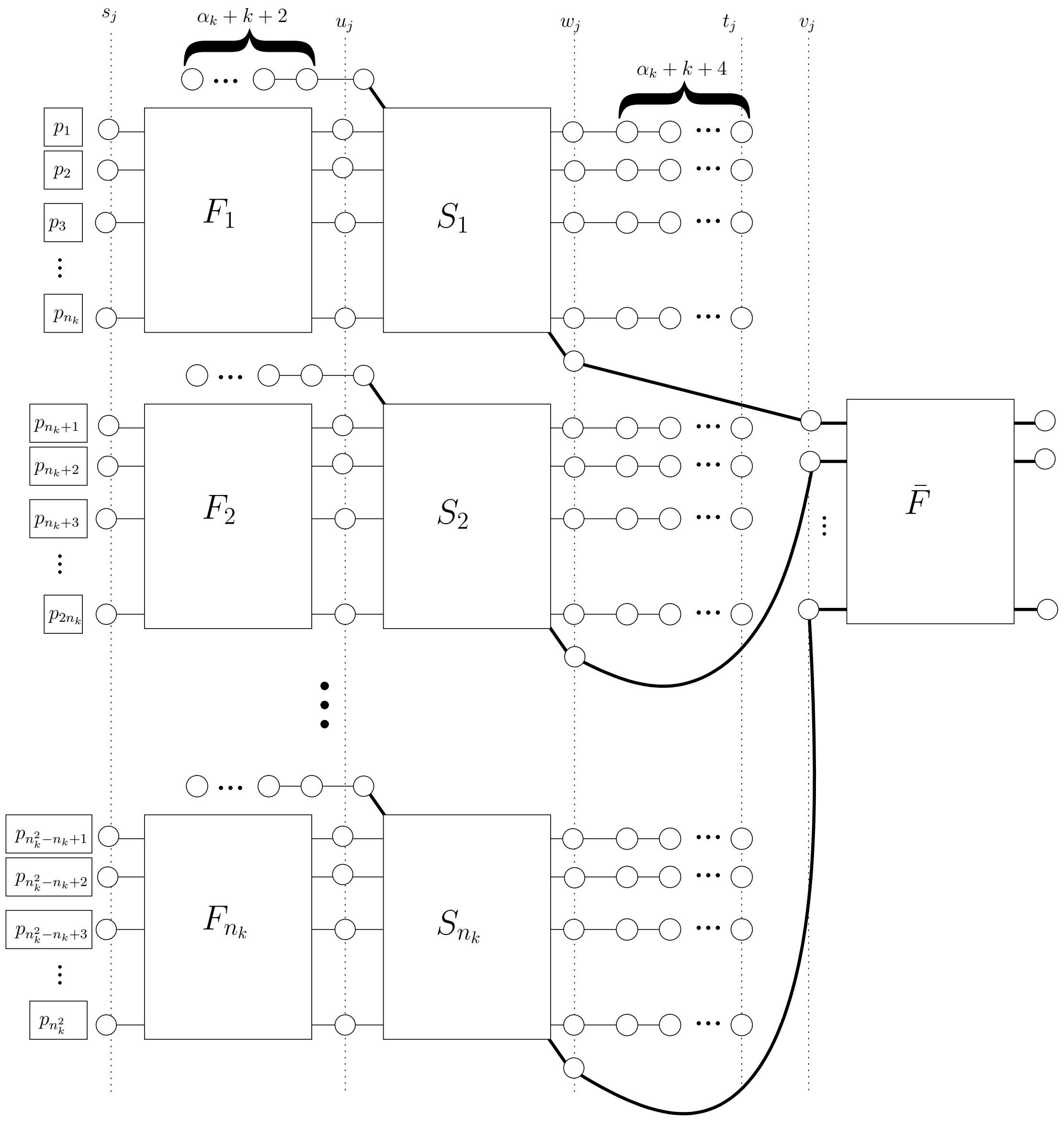

FIGURE 2.6. Construction for Theorem 9

Next we want to show that if ϕ is not satisfiable then the optimal makespan is at least $\alpha_{k+1} + k + 2$. If ϕ is not satisfiable, then in each copy F_j there is at least one formula packet p_j which reaches the vertex u_j after at least $\alpha_k + k + 1$ steps. Now consider the sorting gadget S_j. By construction inside S_j either the sorting packet q_j or p_j is delayed. If p_j is delayed then it needs at least another $(2n_k + 1) + \alpha_k + k + 4$ steps to reach its destination after having reached u_j. This gives $2\alpha_k + 2n_k + 2k + 6 = \alpha_{k+1} + k + 2$ steps in total. So now assume that in each formula gadget F_j the the sorting packet q_j is delayed . This implies that q_j reaches

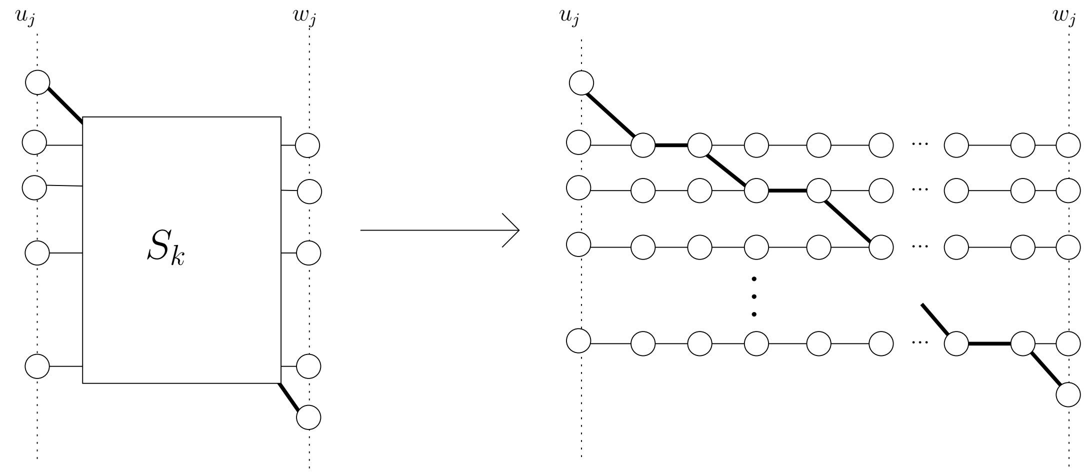

FIGURE 2.7. Sorting gadget

w_j after at least $(\alpha_k + k + 2) + (2n_k + 1) + 1$ steps. Then q_j needs an additional step in order to reach their respective vertex v_j. This reasoning applies to all sorting packets q_j. Since ϕ is not satisfiable there must be at least one sorting vertex which needs another $\alpha_i + i + 1$ steps inside $\bar{F}$. This gives $2\alpha_k + 2n_k + 2k + 6 = \alpha_k + k + 2$ steps in total. Thus, if ϕ is not satisfiable, the length of an optimal schedule is at least $\alpha_k + k + 2$. The total number of packets used is $(n_k)^2 + n_k =: n_{k+1}$. The size of this construction is clearly bounded by a polynomial in the size of I_k and therefore it is bounded by a polynomial in the size of ϕ.

The sketch in Figure 2.6 shows that the underlying graph for I_{k+1} is planar if the underlying graph for I_k was planar. This proofs that it is NP-hard to approximate the packet routing problem with fixed paths on directed planar graphs with an absolute error of k for any $k \geq 0$. □

3. ALGORITHMS

In this section we study algorithms for the packet routing problem on directed and undirected trees.

3.1. **Farthest-Destination-First-Algorithm.**

As a general concept we first introduce the farthest destination first algorithm (FDF). Let $I = (G, \mathcal{M}, \mathcal{P})$ be an instance of the packet routing problem with fixed paths. The FDF-algorithm computes a schedule according to the following rule:

Let e be an edge. If there is only one packet M that needs to use e at a time t, we transfer M along e. If there are several packets $M_1, ..., M_k$ which are ready to use the edge e at time t we transfer the packet M_i along e which still has the longest distance to go to its destination among all packet $M_1, ..., M_k$. Ties are broken arbitrarily. Denote by $FDF(I)$ the longest schedule for the packet routing instance I that the FDF-algorithm could possibly compute.

Let $T = (V, A)$ be a directed tree. We call T an *out-tree* if the indegree of each vertex is at most one. We call T an *in-tree* if the outdegree of each vertex is at most one.

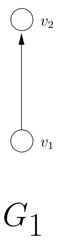

FIGURE 3.1. The graph G_1

Theorem 10. *On in-trees and out-trees the FDF-Algorithm works optimally.*

Proof. This was shown by Leung [19]. □

Throughout the paper we will use the notation $|S|$ for the length of a schedule S. For a packet routing instance I with fixed or variable paths let $OPT(I)$ denote a schedule with minimum makespan.

In terms of approximation factor the FDF-algorithm can perform arbitrarily bad, even if we consider only directed trees.

Theorem 11. *For every $k \geq 1$ there is a a directed tree T_k and a packet routing instance $I_k = (T_k, \mathcal{M}_k)$ such that*

$$|FDF(I_k)| \geq k \cdot |OPT(I_k)|$$

Proof. In order to prove the claim, we inductively construct packet routing instances $(G_1, \mathcal{M}_1), ..., (G_k, \mathcal{M}_k)$. Let $m \geq 1$ be an integer and let $i = 1$. The graph G_1 (see Figure 3.1) consists of two vertices $\{v_1, v_2\}$. The set of packets $\mathcal{M}_1$ has m packets which all start in v_1 and whose destination vertex is v_2. We define all packets to be *upper packets* (in the inductive step we will see the reason for this definition). It is easy to see that for each upper packet M_1 there is a schedule with total makespan $m =: \beta_1$ such that M_1 reaches its destination vertex after $1 =: \alpha_1$ timesteps. Moreover, the lengths of the paths of all packets are identical. Since the FDF-algorithm breaks ties arbitrarily, it might compute a schedule in which M_1 reaches its destination v_2 after $m =: \gamma_1$ timesteps.

Now let $i = 2$. For $J_2 = (G_2, \mathcal{M}_2)$ we take m copies of $J_1 = (G_1, \mathcal{M}_1)$. From each copy $J_{1,\ell}$ we choose one packet M_ℓ and extend its path by two more vertices $\{v_3, v_4\}$. We call these packets M_ℓ the *upper packets* and all other packets the *lower packets*. For all lower packets we extend their path by two more vertices (so for each of these packet we introduce two new vertices). The whole construction is a directed tree and the congestion on the edge (v_3, v_4) is exactly m. See Figure 3.2 for a sketch of G_2.

We can easily see that an optimal schedule has length $2 + m =: \beta_2$ (give priority to the upper packets and schedule the lower packets in arbitrary order). We also see that for each upper packet M_ℓ there is an optimal schedule such that M_ℓ arrives at its destination after $3 =: \alpha_2$ timesteps. Since the paths of all packets have the same length, the FDF-algorithm schedules them in arbitrary order. In the worst possible schedule, the upper packets are scheduled with lowest priority. For each upper packet M_ℓ the FDF-algorithm might compute a schedule in which M_ℓ reaches its destination after $m + 1 + m =: \gamma_2$ steps.

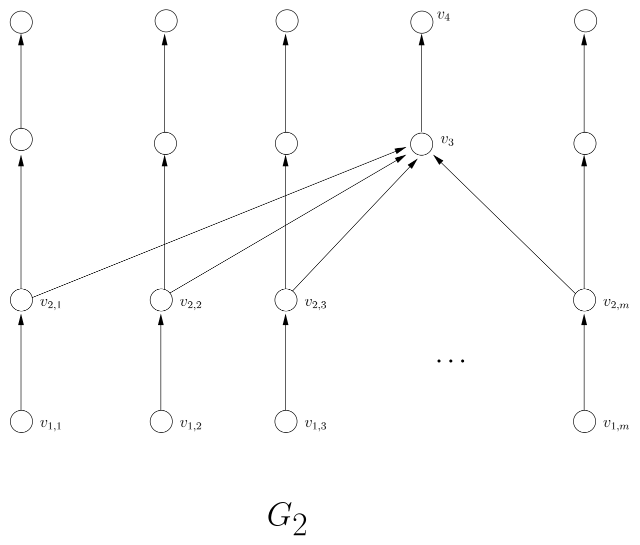

FIGURE 3.2. The graph G_2

We continue inductively: For the construction of J_i we take m copies of J_{i-1}. From each copy $J_{i-1,\ell}$ we choose one upper packet M_ℓ and extend its path by the vertices v_{2i-1} and v_{2i} (so all these packets now share one edge). These packets form the upper packets of J_i. All other packets are lower packets. The paths of all lower packets are extended by two new vertices (so we add two new vertices for each packet). Thus, the lengths of the paths of all packets are identical (since by the induction hypothesis they were identical in the previous iteration).

From the induction hypothesis we know that for each upper packet M_ℓ there is a schedule for $J_{i-1,\ell}$ of length β_{i-1} such that M_ℓ arrives at its destination after α_{i-1} timesteps. Also, there is another schedule for $J_{i-1,\ell}$ in which M_ℓ reaches its destination after γ_{i-1} steps. Thus, for each upper packet M_ℓ in J_i there is a schedule for J_i of length $\max\{\alpha_{i-1}+1+m, \beta_{i-1}+2\} =: \beta_i$ in which $M_{i,\ell}$ reaches its destination after $\alpha_{i-1}+2 =: \alpha_i$ steps. Also, there is an FDF-schedule for J_i in which M_ℓ reaches its destination after $\gamma_{i-1}+1+m =: \gamma_i$ steps.

Some basic calculus shows that $\alpha_i = 2i-1$, $\beta_i = 2i-2+m$ and $\gamma_i = i \cdot m + i - 1$. Recall that $FDF(J_i)$ is the longest possible schedule produced by the FDF-algorithm for J_i. Then it holds that

$$FDF(J_i) \geq \frac{\gamma_i}{\beta_i} \cdot OPT(J_i) = \frac{i \cdot m + i - 1}{m + 2i - 1} \cdot OPT(J_i)$$

The claim follows by choosing m and i sufficiently large. □

3.2. **Undirected Trees.**

Looking at Theorem 11 the FDF-algorithm does not look very promising when good approximation factors are desired. However, we can use it as a subroutine for computing a 2-approximation for the packet routing problem on undirected trees. Since on trees there is a unique simple path between two vertices, we do not distinguish between the settings with fixed and variable paths. Instead we assume that the packets use the unique simple path between their start and destination vertex.

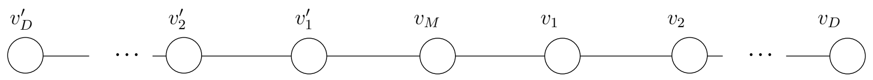

FIGURE 3.3. Graph for showing that the analysis in Theorem 12 is tight.

The algorithm works as follows: Let T be a tree and let $I = (T, \mathcal{M})$ be a packet routing instance. Let v_r be an arbitrary vertex. We define v_r to be the root of the tree and orientate all edges towards v_r. We observe that the orientation of the edges splits the path of each packet M_i into two parts: in the first part M_i moves according to the direction of the edges. In the second part of the path M_i moves in the opposite direction of the edges. Let v_i be the vertex which divides these two parts. We split the routing problem into two subproblems: First we move each packet M_i from s_i to v_i. In the second part we move each packet M_i from v_i to t_i. We observe that the first part is a packet routing problem on an in-tree and can therefore be solved optimally using the FDF-algorithm (see [19]). Similarly, the second part is a packet routing instance on an out-tree which can also be solved optimally using the FDF-algorithm (see [19]). In the overall schedule for I, first we run the optimal (FDF-)schedule for the first part. Then we run the optimal (FDF-)schedule for the second part. Denote by $TREE\,(I)$ the resulting schedule for the instance I.

Theorem 12. *For the schedule $TREE\,(I)$ it holds that $|TREE\,(I)| \leq 2 \cdot |OPT\,(I)|$.*

Proof. Since the length of an optimal schedule for each of the two subproblem forms a lower bound for the size of an optimal schedule for the original problem, we achieve an approximation ratio of two. □

Remark. It can be shown that the time needed to compute $TREE\,(I)$ is bounded by $O\left(n^2\right)$. For details see [22].

Note that the runtime of $O\left(n^2\right)$ is optimal since the size of an optimal schedule in the computer memory can be up to $\Theta\left(n^2\right)$. When implementing the algorithm one would not let packets wait until all other packets have finished the first part of the schedule. We would rather always move a packet when the next edge on its path is free and prioritize the packets according to the algorithm. But even then there is an example that shows that our analysis is tight.

Let D be an arbitrary positive integer (in the packet routing instance this will be the value of the dilation). Consider the graph shown in Figure 3.3. We introduce $2 \cdot D$ packets as follows: We define $M_1 := (v_M, v_D)$ and $M'_1 := (v_M, v'_D)$. For each $k \in \{2, ..., D\}$ we define $M_k := (v_1, v_M)$ and $M'_k := (v'_1, v_M)$. The optimal makespan for this packet routing instance is D, obtained by giving priority to the packets M_1 and M'_1 and scheduling the other packets in arbitrary order. In order to analyze the schedule obtained by our algorithm we need to consider all possible choices for v_r. If $v_r = v_M$ or $v_r = v_k$ with $1 \leq k \leq D$ then the packets M'_2 to M'_k have a higher priority than M'_1 and therefore M'_1 will need $2 \cdot D - 1$ steps to reach v'_D. Analogously, if $v_r = v'_k$ with $1 \leq k \leq D$ then the packets M_2 to M_k have a higher priority than M_1 and therefore M_1 will need $2 \cdot D - 1$ steps to reach v'_D. Thus, the competitive ratio of $\mathcal{A}$ is at best $\frac{2 \cdot D - 1}{D}$. Since we can choose D arbitrarily large this shows that the proven competitive ratio of 2 is tight.

3.3. **Directed Trees.** For directed trees we obtain an even stronger result: we can construct a direct schedule of length at most $C+D-1$ in polynomial time. Again, since on directed trees the paths for the packets are unique we do not distinguish between the settings of fixed and variable paths.

The algorithm works as follows: First we find a coloring for the paths of the packets such that two paths that share an edge have different colors. We will show that the number of colors needed is exactly C. We assign each packet the color of its path. Then we assign each edge a time-dependent color. The idea behind this is that we transfer a packet P with color c_P along an edge $e=(u,v)$ exactly when e has the color c_P. We assign the coloring such that for two consecutive edges $e=(u,v)$ and $e'=(v,w)$ it holds that at time t the edge e' has always the color that e had at time $t-1$. This ensures that once a packet starts moving, it will never stop until it reaches its destination.

3.3.1. *Path Coloring.* Now we describe the algorithm in detail. First we want to find a coloring for the paths such that two paths with the same color do not share an edge. We do this in two phases: in the first phase we consider each vertex v together with its adjacent vertices (this subgraph forms a star). For each of these subgraphs (together with the paths of the packets in this subgraph) we solve the path coloring problem optimally (here we use the fact that our tree is directed). Then we combine all these part-solutions and obtain a solution for the global path-coloring problem.

Phase one: Let v be a vertex and denote by T_v the subgraph induces by v and its adjacent vertices. We want to find a coloring for the paths that use edges in T_v. We reduce this problem to the edge-coloring problem on bipartite multigraphs (note that it is crucial that the edges in T_v are directed):

Let U be the set of vertices which have outgoing edges to v, i.e., $U=\{u|\,(u,v)\in E\}$. Similarly, let W be the set of vertices having ingoing edges from v, i.e., $W=\{w|\,(v,w)\in E\}$. We construct an undirected graph B_v as follows: the set $U\cup W$ forms the set of vertices in B_v. For each path P that goes from a vertex $u\in U$ through v to a vertex $w\in W$ we introduce an edge $e_P:=\{u,w\}$ in B_v. For all paths P that start in a vertex $u\in U$ and end in v we introduce a new vertex $w_P\in W$ and an edge $e_P:=\{u,w_P\}$ in B_v. Similarly for paths P that start in v and end in a vertex $w\in W$ we add a vertex v_P and introduce an edge $e_P:=\{v_P,w\}$ in B_v. Thus, for the maximum degree $\Delta(B_v)$ of a node in B_v it holds that $\Delta(B_v)\le C$. Also, it holds that two edges e_P and $e_{P'}$ share an end-vertex if and only if their corresponding paths P and P' share an edge in T_v. Thus, a valid edge-coloring for B_v implies a valid path coloring for T_v and vice versa. Moreover, from the construction it follows that B_v is bipartite. We compute a minimum edge coloring for B_v (e.g., see [4]). The number of colors needed equals the maximum node degree $\Delta(B_v)$ [4].

Phase two: Now we combine the found solutions for the graphs T_v one by one to obtain a global solution (a similar construction is described in [6, Lemma 2]). We start with an arbitrary vertex v and the path coloring of T_v. Now let v' be a vertex adjacent to v and consider the graph $T_{v'}$. We permute the colors of the paths in $T_{v'}$ such that the paths that use the edge (v,v') (or (v',v), respectively) have the same colors in T_v and $T_{v'}$. We iterate over the vertices by always adding a vertex that is adjacent to one of the vertices that have been considered already. Eventually, for each edge $e=(u,v)$ all paths that use e have the same color in T_u and T_v. Since T is a tree in each iteration we can find a valid permutation of the colors of the paths by using a simple greedy strategy. Since for each graph T_v we find path colorings with at most C colors, the resulting path coloring for T has C colors as well.

3.3.2. *Time-Dependent Coloring.* Now we construct a time-dependent coloring $c : E \times \mathbb{N} \to \{1, 2, ..., C\}$ with C colors for the edges of T. It has the *consecutive property:* for two consecutive edges $e = (u, v)$ and $e' = (v, w)$ it holds that $c(e, i) = c(e', i + 1)$. Since our graph is a directed tree such a coloring can be found with a greedy method: Start with an arbitrary edge e and define its coloring $c(e, i) := i \bmod C$ for all $i \in \mathbb{N}$. Then inductively assign the colors to the remaining edges such that the consecutive property holds.

3.3.3. *Routing Schedule.* Finally we describe the schedule algorithm. To each packet we assign the color of its path. Now let M be a packet on a vertex u that needs to use the edge $e = (u, v)$ next. Let c_P be the color of M. We move M along e in the next timestep t with $c(e, t) = c_P$. Denote by $DTREE(I)$ the resulting schedule for an instance I.

Theorem 13. *Let T be a directed tree and let $I = (T, \mathcal{M})$ be a packet routing instance. It holds that $|DTREE(I)| \leq C + D - 1$. A packet is never delayed once it has left its start vertex (direct routing). Moreover, $DTREE(I)$ can be computed in $O\left(n^2 \log C\right)$.*

Proof. Since no two packets with the same color share an edge there can be at most one packet that uses an edge e at a time t. Each packet M waits in its origin vertex for at most $C - 1$ timesteps. Due to the consecutive property once it left its start vertex it moves to its destination without being delayed any further. Thus, the length of the overall makespan is bounded by $C - 1 + D$.

For the runtime analysis let $n := \max\{|\mathcal{M}|, |V|\}$. The edge coloring problem on bipartite multigraphs can be solved optimally in $O(m \log \Delta)$ where m denotes the number of edges in the graph and $\triangle$ the maximum node degree, see [4]. Thus, computing the optimal path coloring for one graph T_v can be done in $O(n \log C)$ and for all graphs T_v in $O\left(n^2 \log C\right)$.

Then we need to combine the colorings for the graphs T_v to a global path coloring. We say a path P *touches* a vertex v if P it goes through v, starts in v or ends in v. We pick an arbitrary vertex v and and color all paths which touch v in the colors that they have in T_v. After this initialization we iterate by taking vertices v' which are adjacent to already considered vertices. When we iterate we need to pick a color for each uncolored path P which touches v'. For doing this we need to try at most $C \leq n$ colors to find a color for P in a valid color permutation for $T_{v'}$. Since we have at most n paths to color and the order of the vertices v' is obtained by a depth-first-search, the second phase can be done in $O\left(n^2\right)$. This gives a total runtime of $O\left(n^2 \log C\right)$. □

Note that the bound $C + D - 1$ is the best bound we can give in terms of C and D since there are packet routing instances which need this many steps. E.g., consider a path of length D with vertices $v_1, ..., v_D$ and C packets all having as start vertex v_1 and as destination vertex v_D.

In the construction used in the proof of Theorem 11 the lengths of the paths for all packets are all equal and the FDF-algorithm may find a very bad schedule. However, when we require the lengths of the paths of the packets to be pairwise

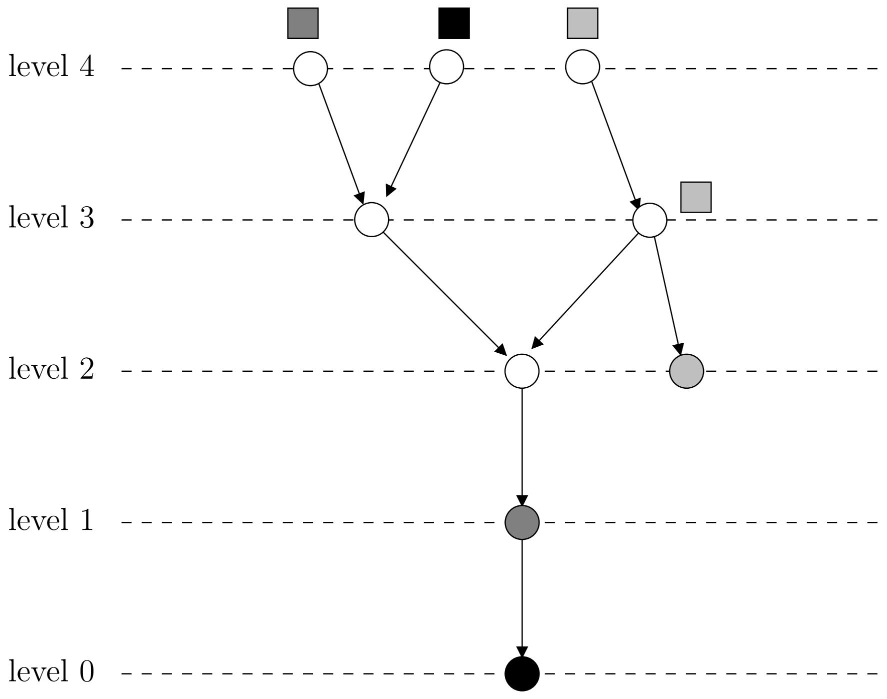

FIGURE 3.4. The partitioning of a directed tree into five levels. The destination vertices of the packets are filled with the color of the respective packets. Note that for the definition of the level of a vertex we cannot just take the maximum distance to a leave since then, e.g., the light gray vertex would be in level 0 rather than in level 2.

different, we can find a direct schedule of length D on directed trees in polynomial time.

Let $T = (V, A)$ be a directed tree and let $I = (T, \mathcal{M})$ be a packet routing instance such that the lengths of the paths of the packets are pairwise different. Note that we can assume that all leaves in T are either start or destination vertices of packets. First we partition the packets into different levels (see Figure 3.4). In order to measure the level distance between two packets we introduce the notion of *directed distance.*

Definition 14 (directed distance)**.** Let T' be the underlying undirected tree of T. Let u and v be two vertices and let $P = (u_0 = u, u_1, u_2, ..., u_k = v)$ be the unique simple path from u to v in T'. We define

$$w(u_i, u_j) := \begin{cases} +1 & \text{if } (u_i, u_j) \in A \\ -1 & \text{if } (u_j, u_i) \in A \end{cases}$$

The *directed distance* $d(u, v)$ between u and v is defined by

$$d(u, v) := \sum_{i=0}^{k} w(u_i, u_j)$$

For each vertex v we define its *level* by $l(v) := \max_{v' \in V} d(v, v')$. For a sketch of the levels of a directed graph, see Figure 3.4. The directed distance between two vertices equals in fact the difference of their levels: $d(u, v) = l(u) - l(v)$.

For a packet $M = (s, t)$ we define its *start level* by $sl(M) := l(s)$ and its *destination level* by $dl(M) := l(t)$. Now we sort the packets by their destination levels. We define $\mathcal{M}_i := \{M | dl(M) = i\}$. Since the lengths of the paths of the packets are pairwise different, for each two packets $M \in \mathcal{M}_i$ and $M' \in \mathcal{M}_i$ it holds that $sl(M) \neq sl(M')$.

If for all pairs of packets M and M' we could guarantee that $sl(M) \neq sl(M')$ then we could immediately obtain a direct schedule by greedily always moving all packets. Clearly, this schedule would be optimal since each packet is sent directly to its destination vertex without being delayed at all. However, in general different packets may have the same start level. Nevertheless, we can compute a one-one map $vl : \mathcal{M} \to \mathbb{N}$ such that

- $vl(M) \neq vl(M')$ if $M \neq M'$,
- $vl(M) \geq sl(M)$ and
- $vl(M) - dl(M) \leq D$ for all packets M.

The intuition behind this is that we assign each packet M a new virtual start level $vl(M)$ such that $vl(M)$ is not smaller than $sl(M)$ (intuitively speaking: the new start level is "higher up" than the original one). Assuming the map vl as the level assignment of the packets, we can construct a (virtual) direct routing schedule by greedily moving all packets at all timesteps. The length of this schedule is at most D, since $vl(M) - dl(M) \leq D$ for all packets M.

When computing a routing schedule for I we can simulate the virtual schedule described above. First we compute vl. Then we schedule the packets as follows: A packet $M = (s, t)$ waits in s for $vl(M) - sl(M)$ timesteps (from the first property of vl this is non-negative). Then it moves to t without being delayed any further. Denote by $DTPD(I)$ ("directed tree path-lengths pairwise different") the resulting schedule.

In order to compute vl we inductively define maps $l_i : \mathcal{M} \to \mathbb{N}$. We begin with $l_0(M) := d(s, t)$ for all packets $M = (s, t)$. Assume the maps $l_0, ..., l_k$ are computed already. When computing l_{k+1} for all packets $M \in \bigcup_{i \leq k} \mathcal{M}_i$ we fix their values from l_k. The values for all other packets are increased by at least one. Algorithm 3.1 shows the computation in detail. See Figure 3.5 for a sketch of the inductive steps.

Theorem 15. *Let T be a directed tree and $I = (T, \mathcal{M})$ be a packet routing instance such that the lengths of the paths of the packets are pairwise different. Then the schedule $DTPD(I)$ is optimal with $|DTPD(I)| = D$. The computation of $DTPD(I)$ can be done in $O(n^2)$.*

Proof. First we need to show that vl is one-one and for all packets M it holds that $vl(M) \geq sl(M)$ and $vl(M) \leq D + dl(M)$. We give a proof by induction.

The map l_0 is one-one since the lenghts of the packets are pairwise different. Moreover, it holds that $l_0(M) \leq D + 0$. For induction we assume that $l_k(M) \leq D + k$ and l_k is one-one. Since we defined $l_{k+1}(M_1) := D + k + 1$ it follows that $l_{k+1}(M_1) > l_{k+1}(M)$ for all packets $M \in \bigcup_{i \leq k} \mathcal{M}_i$. From the definition of $l_{k+1}(M')$ for packets $M' \in \bigcup_{i > k} \mathcal{M}_i$ and the induction hypothesis it follows that l_{k+1} is one-one. Moreover, for all packets M it holds that $l_{k+1}(M) \leq l_{k+1}(M_1)$ which implies that $l_{k+1}(M) \leq D + k + 1$.

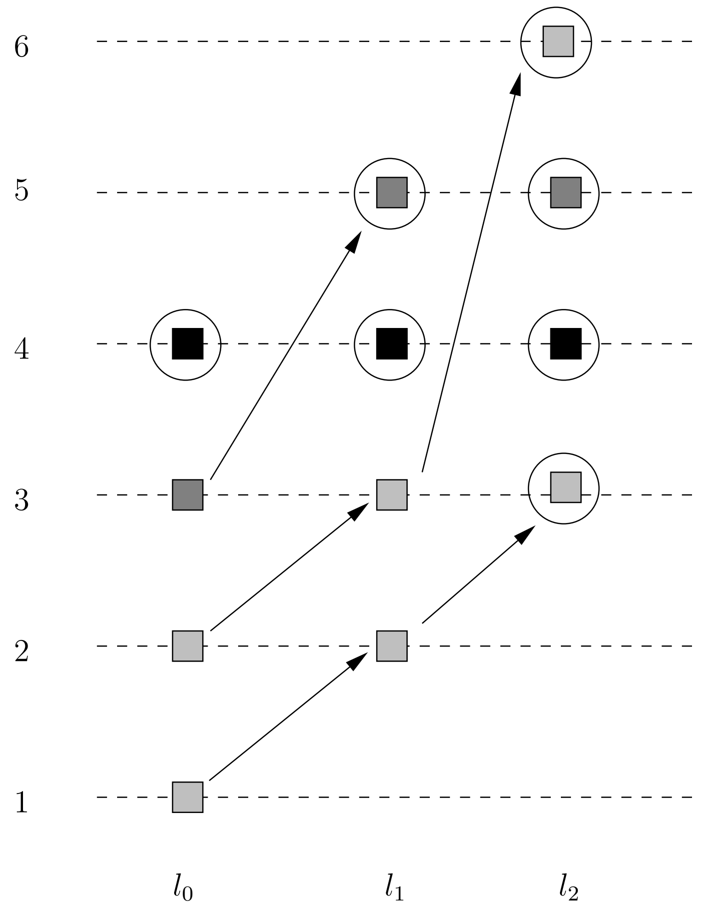

FIGURE 3.5. The inductive steps for the construction of h'. The circles denote packets whose values are not redefined after the respective iteration.

Let M be a packet. We want to prove that $vl\,(M) \leq D + dl(M)$. For l_0 it clearly holds that $l_0\,(M) \leq D + dl(M)$. Now assume that $l_k\,(M) \leq D + dl(M)$. For the packet M_1 we have that $l_{k+1}\,(M_1) = D + k + 1 \leq D + dl\,(M_1)$. If $M \in \bigcup_{i>k} \mathcal{M}_i$ we know that $l_{k+1}\,(M) < l_{k+1}\,(M_1) = D+k+1 \leq D+dl(M)$. If $M \in \bigcup_{i\leq k} \mathcal{M}_i$ it holds that $l_k\,(M) = l_{k+1}\,(M)$ and thus the claim follows from the induction hypothesis.

Now we want to show that $vl\,(M) \geq sl\,(M)$. Let M be a packet with $dl(M) = k$. Then in the first k iterations of the computation of vl the respective value for M was strictly increased. Formally, $l_0(M) < l_1(M) < ... < l_{k-1}(M) < l_k(M) = l_{k+1}(M) = l_{k+2}(M) = ... = l_{MAX}(M) = vl(M)$. We conclude that $vl(M) \geq l_0(M) + k = dist\,(s,t) + dl(M) = sl(M)$.

Now we show that $DTPD(I)$ is a valid schedule. Let $M_i = (s_i, t_i)$ be a packet. Denote by $v_{i,\tau}$ the vertex where M_i is located at time τ. Throughout this proof we assume that $\tau \geq 0$. We call M_i *active at time* τ if $vl\,(M_i) - sl\,(M_i) \leq \tau$ and $vl\,(M_i) - \tau > dl\,(M_i)$. Intuitively speaking, a packet is active at time τ if it moves at time τ. Note that if a packet M_i is active then $l\,(v_{i,\tau}) = vl\,(M_i) - \tau$.

Algorithm 3.1: Computation of the map vl

```
foreach $M = (s,t) \in \mathcal{M}$ do
    $l_0(M) := dist\,(s,t)$;
end
$K$:=max$\{dl(M) | M \in \mathcal{M}\}$;
for $k$=1 to $K$ do
    let $\mathcal{M}_S := \bigcup_{i \leq k} \mathcal{M}_i$;
    let $\mathcal{M}_L := \bigcup_{i > k} \mathcal{M}_i$;
    foreach $M \in \mathcal{M}_S$ do
        $l_k(M) := l_{k-1}(M)$;
    end
    let $M_1 \in M_L$ be the packet with $l_k(M_1) \geq l_k(M)$ for all packets $M \in \mathcal{M}_L$;
    $l_k(M_1) := D + k + 1$;
    foreach $M \in \mathcal{M}_L$ do
        let $p$ be the minimum value such that $p > l_k(M)$ and $\exists M' \in \mathcal{M}_L$ with
        $l_k(M') = p$;
        $l_k(M) := p$;
    end
end
```

We claim that for two active packets M_i and M_j it holds that $l(v_{i,\tau}) \neq l(v_{j,\tau})$. Assume on the contrary that there are two active packets M_i and M_j and a value for τ with $l(v_{i,\tau}) = l(v_{j,\tau})$ in $DTPD(I)$. Also assume that τ is the first time that this happens in $DTPD(I)$. Since both packets are active and thus $vl(M_i) - \tau = l(v_{i,\tau}) = l(v_{j,\tau}) = vl(M_j) - \tau$ we conclude that $vl(M_i) = vl(M_j)$. But this is a contradiction since vl is one-one. Thus, in $DTPD(I)$ no two packets are ever located on the same vertex and, therefore, $DTPD(I)$ is a direct schedule.

Now we prove that $|DTPD(I)| = D$. From the definition of the algorithm and since $DTPD(I)$ is a direct schedule it follows that $l(v_{i,\tau}) = \max\{\min\{vl(M_i) - \tau, sl(M_i)\}, dl(M_i)\}$ for all packets M_i and all τ. We already showed that $vl(M_i) \leq D + dl(M_i)$. Note that this implies that $vl(M_i) - D \leq D + dl(M_i) - D = dl(M_i) \leq sl(M_i)$ and thus $l(v_{i,D}) = \max\{\min\{vl(M_i) - D, sl(M_i)\}, dl(M_i)\} = \max\{vl(M_i) - D, dl(M_i)\}$.

Putting things together we conclude that

$$\begin{aligned} l(v_{i,D}) &= \max\{vl(M_i) - D, dl(M_i)\} \\ &\leq \max\{(D + dl(t_i)) - D, dl(M_i)\} \\ &= dl(M_i) \end{aligned}$$

This shows that after at most D steps M_i has reached t_i. Since for the computation of $DTPD(I)$ we only need to state how long each packet has to wait in its start vertex the computation of vl dominates the runtime of the algorithm. The map vl can be computed in $O(n^2)$ and thus the computation of $DTPD(I)$ can be done in $O(n^2)$. □

4. Conclusion

We investigated the packet routing problem with fixed and variable paths. A variant, the delay routing problem has already been shown to be NP-hard [15]. We proved that the packet routing problem is also NP-hard (for fixed and variable paths), and that there can be no polynomial time approximation schemes (PTAS) for it unless $P = NP$. This holds even if we restrict the considered graph class to directed trees.

All NP-hardness results rely on constructing a gap of one timestep between yes- and no-instances of the 3-SAT variant that we reduce from. This raises the question whether there is an approximation algorithm with an absolut error. We answered this question in the negative. We proved that there is no approximation algorithm with an absolute error of k for any $k \geq 0$ for the packet routing problem unless $P = NP$. This holds even for planar graphs. It would be interesting to investigate whether on trees an absolute approximation is possible.

We showed that the FDF-algorithm can perform arbitrarily bad even on directed trees due to bad tie-breaking decisions. However, we showed how it can be used in a beneficial manner to obtain a 2-approximation for packet routing in undirected trees. For directed trees we can do even better. We presented an algorithm that finds a direct routing schedule of length at most $C + D - 1$. In cases where C is known to be significantly smaller than D (or vice versa) this gives a provably better approximation than 2. For direct routing on undirected trees the best known result is a schedule of length $2C + D - 2$ yielding a 3-approximation [3]. Our 2-approximation for general trees is the first step towards approximation algorithms that do not rely on C and D as lower bounds. This is very desirable since our bound of $C + D - 1$ is the best one can guarantee in terms of C and D.

For the setting that the lengths of the paths of the packets are pairwise different, we presented an optimal algorithm for directed trees which computes a schedule of length D. It is interesting to find more classes of packet routing instances which allow schedules of that length and algorithms to compute these schedules.

Acknowledgements. We would like to thank Joachim Reichel, Ronald Koch and Daniel Plümpe for helpful discussions.

References

[1] M. Adler, S. Khanna, R. Rajaraman, and A. Rosénbusch. Time-constrained scheduling of weighted packets on trees and meshes. In *Proceedings of the 11th annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, 1999.

[2] F. Meyer auf der Heide and B. Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31, 1999.

[3] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. Spirakiscac. Direct routing: Algorithms and complexity. In *Proceedings of the 12th Annual European Symposium on Algorithms*, volume 3221 of *LNCS*, pages 134–145, 2004.

[4] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21:5–12, 2001.

[5] W. J. Cook, L. Lovász, and J. Vygen, editors. *Research Trends in Combinatorial Optimization*. Springer, 2009.

[6] T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255:33–50, 2001.

[7] L. Fleischer and M. Skutell. Quickest flows over time. *SIAM Journal on Computing*, 36:1600–1630, 2007.

[8] L. Fleischer and M. Skutella. Minimum cost flows over time without intermediate storage. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 66–75, Baltimore, MD, 2003.

[9] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[10] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[11] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. Freeman NY, 1979.

[12] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 2719:397–409, 2003.

[13] A. Hall, K. Langkau, and M. Skutella. An FPTAS for quickest multicommodity flows with inflow-dependent transit times. *Algorithmica*, 47:299–321, 2007.

[14] B. Hoppe and É. Tardos. The quickest transshipment problem. In *Proceedings of the 6th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 512–521, 1995.

[15] M. Di Ianni. Efficient delay routing. In *2nd International EURO-PAR Conference*, volume 1123 of *LNCS*, 1996.

[16] R. Koch, B. Peis, M. Skutella, and A. Wiese. Real-time message routing and scheduling. Technical Report 006-2009, Technische Universität Berlin, February 2009.

[17] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-scheduling in $O(congestion + dilation)$ steps. *Combinatorica*, 14:167–186, 1994.

[18] F. T. Leighton, B. M. Maggs, and A. W. Richa. Fast algorithms for finding $O(congestion + dilation)$ packet routing schedules. *Combinatorica*, 19:375–401, 1999.

[19] J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. 2004.

[20] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14, 1993.

[21] R. Ostrovsky and Y. Rabani. Universal $O(congestion + dilation + \log^{1+\epsilon} N)$ local control packet switching algorithms. In *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pages 644–653, 1997.

[22] B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. Technical Report 003-2009, Technische Universität Berlin, February 2009.

[23] Y. Rabani and É. Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th annual ACM Symposium on Theory of Computing*, pages 366–375. ACM, 1996.

[24] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30, 2001.

[25] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov, and D. B. Shmoys. Short shop schedules. *Operations Research*, 45:288–294, 1997.