Algorithmica
DOI 10.1007/s00453-016-0177-0

Online Packet-Routing in Grids with Bounded Buffers

Guy Even[1] · **Moti Medina**[2]

Received: 5 October 2014 / Accepted: 16 June 2016

Abstract We present deterministic and randomized algorithms for the problem of online packet routing in grids in the competitive network throughput model (Aiello et al. in SODA, pp 771–780 2003). In this model the network has nodes with bounded buffers and bounded link capacities. The goal in this model is to maximize the throughput, i.e., the number of delivered packets. Our deterministic algorithm is the first online algorithm with an $O\left(\log^{O(1)}(n)\right)$ competitive ratio for uni-directional grids (where n denotes the size of the network). The deterministic online algorithm is centralized and handles packets with deadlines. This algorithm is applicable to various ranges of values of buffer sizes and communication link capacities. In particular, it holds for buffer size and communication link capacity in the range $[3 \ldots \log n]$. Our randomized algorithm achieves an expected competitive ratio of $O(\log n)$ for the uni-directional line. This algorithm is applicable to a wide range of buffer sizes and communication link capacities. In particular, it holds also for unit size buffers and unit capacity links. This algorithm improves the best previous $O(\log^2 n)$-competitive ratio of Azar and Zachut (ESA, pp 484–495, 2005).

Keywords Online algorithms · Packet routing · Bounded buffers · Admission control · Grid networks

Preliminary versions of this manuscript appeared in the proceedings of ICALP 2010 [13] and SPAA 2011 [14].

✉ Moti Medina
moti.medina@gmail.com

Guy Even
guy@eng.tau.ac.il

1 School of Electrical Engineering, Tel-Aviv University, 69978 Tel Aviv, Israel

2 MPI for Informatics, 66123 Saarbrücken, Germany

Published online: 11 July 2016

1 Introduction

Large scale communication networks partition messages into packets so that high bandwidth links can support multiple sessions simultaneously. Packet routing is used by the Internet as well as telephony networks and cellular networks. Thus, the development of algorithms that can route packets between different pairs of nodes is a fundamental problem in networks. In a typical setting, requests for routing packets arrive over time, thus calling for the development of online packet routing algorithms. The holy grail of packet routing is to develop online distributed algorithms whose performance is competitive with respect to multiple criteria, such as: throughput (i.e., deliver as many packets as possible), delay (i.e., guarantee arrival of packets on time), stability (e.g., constant rate, avoid buffer overflow) , fairness (i.e., fair sharing of resources among users), etc. From a theoretical point of view, there is still a huge gap between known lower bounds and upper bounds for packet routing even in the simple setting of directed paths and centralized algorithms.

We study the "Competitive Network Throughput Model" introduced by [4] for dynamic routing on networks with bounded buffers. The goal is to route packets (i.e., constant length formatted data) in a network of n nodes. Nodes in this model are switches with local memories called buffers. An incoming packet is either forwarded to a neighbor switch, stored in the buffer, or erased. The resources of a packet network are specified by two parameters: c—the capacity of links and B—the size of buffers. The capacity of a link is an upper bound on the number of packets that can be transmitted in one time step along the link. The buffer size is the maximum number of packets that can be stored in a node.

1.1 Previous Work

Algorithms for dynamic routing on networks with bounded buffers have been studied both in theory and in practice. The networks we study are uni-directional grids of d dimensions. Such 2-dimensional grids with or without buffers serve as crossbars in networks (see [5,6,25] for many references from the networking community). Thus, even centralized algorithms for this task are of interest since they can be used to control a crossbar.

Online Algorithms for Uni-directional Lines Our work on uni-directional line networks is based on a sequence of papers starting with [4]. In [4], a lower bound of $\Omega(\sqrt{n})$ was proved for the greedy algorithm on uni-directional lines if the buffer size B is at least two. For the case $B = 1$ (in a slightly different model), an $\Omega(n)$ lower bound for any deterministic algorithm was proved by [3,7]. Both [7] and [3] developed, among other things, online randomized centralized algorithms for uni-directional lines with $B > 1$. In [3] an $O(\log^3 n)$-competitive randomized centralized algorithm was presented for buffer size B at least 2. For the case $B \geq 2$, [3] proved that nearest-to-go is $\tilde{O}(\sqrt{n})$-competitive. For the case $B = 1$, [3] presented a randomized $\tilde{O}(\sqrt{n})$-competitive distributed algorithm. (This algorithm also applies to rooted trees when the packet destinations are the root.) In [7], an $O(\log^2 n)$-competitive randomized algorithm was presented for the case $B \geq 2$. (This algorithm also applies to rings and trees).

Table 1 Comparison of our results to previous online algorithms for packet routing

Paper	d	Comp. ratio	Det.\Rand.	Range of B, c	Remarks
Angelov et al. [3]	2	$\tilde{\Theta}(n^{2/3})$	Det.	$B > 1$	Distributed, nearest-to-go, 1-bend routing
Angelov et al. [3]	1	$\tilde{O}(\sqrt{n})$	Det.	$B > 1$	Distributed, nearest-to-go
Angelov et al. [3]	1	$\tilde{O}(\sqrt{n})$	Rand.	$B = 1$	Shared randomness, distributed
Angelov et al. [3]	1	$O(\log^3 n)$	Rand.	$B > 1$	Centralized
Azar and Zachut [7]	1	$O(\log^2 n)$	Rand.	$B > 1$	Centralized, FIFO buffers
Theorem 4	1	$O(\log^5 n)$	Det.	$[3, \log n]$	Centralized, preemptive, deadlines
Theorem 10	d	$O(\log^{d+4} n)$	Det.	$[3, \log n]$	Centralized, preemptive, deadlines
Theorem 13	d	$O(\log n)$	Det.	$B, c > \log n$, $B/c = n^{O(1)}$	Centralized, deadlines
Theorem 29, 31	1	$O(\log n)$	Rand.	$B \in [1, \log n]$, $c \geq 1$	Centralized
Theorem 30	1	$O(\log n)$	Rand.	$\log n \leq B/c \leq n^{O(1)}$	Centralized

The networks are uni-directional d-dimensional directed grids. Unless written otherwise link capacities are unit. In the special case of $B = 0$ and $c \geq 3$, the algorithm stated in Theorem 11 is $O(\log^{d+2} n)$-competitive. The remark "deadlines" means that the algorithm handles requests with deadlines

Online Algorithms for Uni-directional Grids Angelov et al. [3] showed that the competitive ratio of greedy algorithms in uni-directional 2-dimensional grids is $\Omega(\sqrt{n})$ and that nearest-to-go policy achieves a competitive ratio of $\tilde{\Theta}(n^{2/3})$.

Other Related Results Kleinberg and Tardos [18] studied the disjoint path problem in undirected planar graphs (see [18] for a formal description of the family of graphs for which their results hold). They presented constant approximation randomized algorithm for this problem as well as an online algorithm with logarithmic competitive ratio. Note that our results apply to high-dimensional grids that do not satisfy the planarity requirement in [18].

Leighton et al. [19] and subsequent works [20,23,24] deal with a different model for packet routing. In this model, there are unbounded input queues and bounded intermediate buffers. In addition, each packet comes with a path along which it is sent. The latency of each packet is $O(C + D)$, where C denotes the maximum congestion and D denotes the length of a longest path.

Offline algorithms for trees and meshes were studied in [5] . They obtained a logarithmic approximation ratio for unbounded buffers and a constant approximation ratio for bufferless networks. Offline packet routing for uni-directional lines was studied in [16,22].

1.2 Our Results

In this paper, we unify results from [13,14] with slightly improved constants. The following results are presented for online packet routing in d-dimensional uni-directional grids (for $d = O(1)$). See Table 1 for a comparison of our results to previous results.

Deterministic Online algorithm We present a centralized *deterministic* online algorithm for packet routing in uni-directional grids with n nodes. Our algorithm achieves a polylogarithmic competitive ratio for a wide combination of parameters described below. (The buffer size is denoted by B and the link capacities are denoted by c.) The deterministic packet-routing algorithm handles requests with deadlines, allows preemptions (i.e., packets may be dropped before they reach their destination), and employs adaptive routing (i.e., part of the route is computed while the packet is traveling to its destination).

(i) For $B, c \in [3 \ldots \log n]$, the competitive ratio of the algorithm is $O(\log^{d+4} n)$ for uni-directional grids of dimension d.
(ii) For $B = 0$ and $c \geq 3$, the competitive ratio of the algorithm is $O(\log^{d+2} n)$ for uni-directional grids of d dimensions. In the trivial case of a uni-directional line (i.e., $d = 1$), our algorithm is degenerated to the nearest-to-go policy [4] and is optimal.
(iii) For $B, c \geq \log n$ and $B/c = n^{O(1)}$ the algorithm reduces to online integral path packing [2,10]. The competitive ratio of the algorithm is $O(\log n)$ for uni-directional grids, independent of the dimension d. In this algorithm, packets are either rejected or routed but not preempted.

In the rest of the paper, we address the algorithm for uni-directional grids as the 'deterministic' algorithm.

A Randomized Algorithm for the One Dimensional Case We present a centralized online *randomized* packet routing algorithm for maximizing throughput in uni-directional lines.[1] Our algorithm is *nonpreemptive*; rejection is determined upon arrival of a packet. Our algorithm is centralized and randomized and achieves an $O(\log n)$-competitive ratio. In addition to handling the case that $B = 1$ and $c = 1$, our algorithm improves over previous algorithms as follows:

(i) The competitive ratio is $O(\log n)$ compared to the best previous competitive ratio of $O(\log^2 n)$ by Azar and Zachut [7].
(ii) Our algorithm works also for buffers of size $B = 1$ (with no restriction on the link capacities).
(iii) We consider also the parameter c of the capacity of the links ([3,7] considered only the case $c = 1$).
(iv) The $O(\log n)$ competitive ratio applies for the following combination of parameters: (1) $B \in [1, \log n]$ and $c \geq 1$, or (2) $\log n \leq B/c \leq n^{O(1)}$.

In the rest of the paper, we address the algorithm for uni-directional lines as the 'randomized' algorithm.

[1] We remark that the randomized algorithm can be generalized to d-dimensional grids to obtain competitive ratios that are $(O(\log n))^d$. In light of similar competitive ratios with the deterministic algorithm, we omit the description and analysis of the randomized algorithm for d-dimensional grids.

1.3 Techniques

Reduction of Packet-Routing to Circuit Switching Packet routing is reduced to a circuit switching problem [2,18] by applying a *space-time transformation* [1,6,7,22]. We extend the space-time transformation of [7] so that it also supports deadlines.

The reduction of packet routing to circuit switching relies on the ability to bound the path lengths without losing too much throughput. In [7] a bound on the path lengths that incurs only a constant fraction loss of throughput is proven for routing in a uni-directional line. We extend the lemma of [7] to d-dimensional grids and to general values of buffer sizes B and link capacities c.

This implies that online packet-routing is reduced to the well studied problem of online packing of paths [2,10]. Algorithms for online packing of paths either reject a request or assign a path to a request (i.e., perform call admission). The edge capacities of the space-time graph are B and c. If the capacities are large, i.e., $B, c \geq \log n$, then the online path packing algorithm by Awerbuch et al. [2] achieves a $\log n$ competitive ratio, where n is number of vertices of the (original) graph, as required. In the case where the capacities are small, i.e., $B, c < \log n$, the algorithm by [2] does not apply, hence we coalesce groups of nodes by *tiling* [9,18]. This induces a new graph, called a *sketch graph* in which the capacities are (again) large. We apply the online path packing algorithm over the sketch graph, but are left with the problem of translating paths over the sketch graph to paths over the space-time graph. We refer to this translation as *detailed routing*. We use the framework of Buchbinder and Naor [10,11] for *online path packing* because it helps us point out the tradeoffs between the path lengths, the competitive ratios, and the overloading of edges.

Detailed Routing The path packing algorithm computes a path over the sketch graph, and the algorithm must translate this sketch path to a detailed path over the space-time graph. The detailed path traverses the same tiles that are traversed by the sketch path and bends whenever the sketch path bends. Detailed routing has been addressed before in undirected graphs [9,18] as well as in space-time graphs of the uni-directional line [22].

Detailed routing is not always successful; indeed, we need to bound the fraction of the requests that are lost during detailed routing. In the deterministic algorithm, the detailed routing technique partitions each path in the sketch graph into three parts, and reserves only a unit of capacity for each part. This is the reason why the algorithm requires $B, c \geq 3$. In some parts of the detailed routing, we reduce the problem of detailed routing to *online interval packing*. This reduction uses an online procedure for packing intervals on a line (which is, in fact, a nearest-to-go routing policy). We apply an online distributed simulation of the optimal interval packing algorithm [17]. The correctness of this simulation is based on the ability of the packet-routing algorithm to preempt (i.e., drop) packets.

Classify and Select Requests are categorized as *near* or *far*, and the algorithm randomly chooses to deal with one category of requests. The categorization is based on the tiles. A request that can be routed within a tile is considered near; otherwise it is a far request.

Randomization is also employed to choose a random subset of the requests so as to further weaken the adversary. We use *random phase shifts* that determine the quadrants within tiles from which paths may start.

Random Sparsification Requests that are assigned sketch paths by the online path packing algorithm are randomly sparsified. This *random sparsification* has two roles: (1) Reduction of loads of sketch graph edges incurred by the path packing algorithm to a small constant fraction with high probability. (2) Solving the problem that the source nodes of requests may be densely packed in an area A. The capacity of the edges that enable routing paths out of A is proportional to the "perimeter" of A, while the number of source nodes in A is proportional to the "area" of A. In a d dimensional grid, the area of a subregion can be as large as the perimeter of the subregion to the power d. By applying random sparsification, the number of remaining paths whose source node is in a quadrant of a tile roughly equals the perimeter of the quadrant.

Comparison to [7]. There are three main differences between this paper and [7].

Node Model The first difference is the node model for Store-and-Forward Networks. (see "Appendix 6" for a detailed comparison.) We believe that our model is simpler and more realistic. The linear lower bounds [3,7] on the competitive ratio for the case of unit buffer sizes do not hold for our model. In fact, our randomized algorithm is $O(\log n)$-competitive even if buffer capacities are unit.

Integral Solution First, Sparsify Later The second difference is in the algorithmic design. The online algorithm in [7] computes a fractional solution and rounds it to an integral solution. In this paper, the online algorithm computes a (non-feasible) integral solution which is randomly sparsified to obtain feasibility. The sparsification technique helps in dealing with the logarithmic ratio between the number of request sources in a quadrant of a tile and the cut along which these requests must be routed. For more details on this sparsification see Sect. 7.4 and in particular Sect. 7.4.3.

Tiling and Detailed Routing We employ tiling to "increase" the capacities of the input graph to $\Omega(\log n)$. These high capacities enable the use of known online path packing algorithms [2,10] that are $O(\log n)$ competitive. Since the tiling produces a "low resolution" sketch graph, the outcome of the online path packing procedure is sketch paths that need to be translated to the higher resolution graph, i.e., the input graph. This translation is called "detailed routing". We designed the detailed routing procedure in our algorithm so that it is modular. This modulatory enables us to show its correctness and its effect over the competitiveness of the algorithm. Moreover, detailed routing is adaptive and computed in a distributed on-the-fly fashion. This part of the online algorithm is different in the deterministic and randomized algorithm, e.g., in the deterministic algorithm some of the packets are dropped during detailed routing. For more details see Sects. 5.2 , and 7.4.2.

1.4 Organization

The formal definition of the problem is stated in Sect. 2. In Sect. 3, the reduction of packet-routing to path packing is presented. In Sect. 4, we outline the steps of

the deterministic algorithm. In Sect. 5, we elaborate on each step of the deterministic algorithm with respect to uni-directional lines and prove that the algorithm is $O(\log^5 n)$-competitive, where n is the number of nodes. In Sect. 6 we present a generalization of the deterministic algorithm to the d-dimensional case and extensions to special cases, such as: bufferless grids, and grids with large buffers and large link capacities. In Sect. 7 we design and analyze a randomized algorithm for uni-directional lines. Our randomized algorithm achieves a competitive ratio of $O(\log n)$.

2 Problem Definition

2.1 Store-and-Forward Packet Routing Networks

We consider a synchronous store-and-forward packet routing network [3,4,7].

Each packet is specified by a 4-tuple $r_i = (a_i, b_i, t_i, d_i)$, where $a_i \in V$ is the source node of the packet, $b_i \in V$ is the destination node, $t_i \in \mathbb{N}$ is the time step in which the packet is input to a_i, and d_i is the deadline. Since we consider an online setting, no information is known about a packet r_i before time t_i. Deadlines mean that the algorithm is only credited for delivering packet r_i to its destination b_i before time d_i.

The network is a directed graph $G = (V, E)$. Each edge has a capacity c that specifies the number of packets that can be transmitted along the edge in one time step. Each node has a local buffer of size B that can store at most B packets. Each node has a local input through which multiple packets may be input in each time step. The network operates in a synchronous fashion with a delay of one time step for communication. This means that a single time step is needed for a packet to traverse a single link.

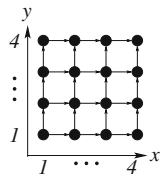
In each time step, a node v considers the packets arriving via the local input, the packets arriving from incoming edges, and the packets stored in the buffer. Packets destined to node v (i.e., $b_i = v$) are removed from the network (this is considered a success provided that the deadline has not passed, and no further routing of the packet is required). As for the other packets, the node determines which packets are sent along outgoing edges (i.e., forwarded) and which packets are stored in the buffer. The remaining packets are *deleted*.

The literature contains two different models of node functionality. We use the model used by [6,22]. The reader is referred to "Appendix 6" for a comparison between two different models of node functionality; this comparison is mostly of interest for the case $B = 1$.

We use the following terminology. A packet is *rejected* if it is locally input to a node and the node deletes it. A packet that is locally input but not rejected is called an *injected* packet. A packet is *preempted* or *dropped* if it was injected and deleted before it reached its destination.

The task of *admission control* is to determine which packets are injected and which are rejected. An algorithm that drops packets is a *preemptive algorithm*; an algorithm that does not drop packets is called a *non-preemptive algorithm*.

Fig. 1 A 4×4 grid network



2.2 Grid Networks

A two dimensional $\ell_1 \times \ell_2$ uni-directional grid network is a directed graph $G = (V, E)$ defined as follows (see Fig. 1). The set of vertices is $V \triangleq [\ell_1] \times [\ell_2]$, where $[\ell]$ denotes the set of integers $\{1, \ldots, \ell\}$. We denote the number of vertices by n (i.e., $n = \ell_1 \cdot \ell_2$). There are two types of edges: horizontal edges $(i, j) \rightarrow (i + 1, j)$ and vertical edges $(i, j) \rightarrow (i, j+1)$. For each packet, the source node $a_i = (a_i(x), a_i(y))$ and the destination node $b_i = (b_i(x), b_i(y))$ satisfy $a_i \leq b_i$ (i.e., $a_i(x) \leq b_i(x)$ and $a_i(y) \leq b_i(y)$). We refer to an $\ell_1 \times \ell_2$ two dimensional directed grid network simply as a grid.

A d-dimensional grid is defined analogously over a vertex set $V \triangleq [\ell_1] \times \cdots \times [\ell_d]$. Our analysis applies to the case that d is a constant.

Capacities and Buffers We assume uniform capacities and buffer sizes. Namely, (i) all edges in the grid have the same capacity, denoted by c; and (ii) all nodes have the same buffer size, denoted by B.

2.3 Online Maximum Throughput in Networks

The *throughput* of a packet routing algorithm is the number of packets that are delivered to their destination *before their deadline*. We consider the problem of maximizing the throughput of an online centralized deterministic packet-routing algorithm.

Let σ denote an input sequence. Let ALG denote a packet-routing algorithm. Let $\text{ALG}(\sigma)$ denote the subset of requests in σ that are delivered on time by ALG. The throughput obtained by ALG on input σ is the size of the set $\text{ALG}(\sigma)$, i.e., $|\text{ALG}(\sigma)|$. Let $\text{OPT}(\sigma)$ denote the subset of requests in σ that are delivered by an optimal throughput routing. An online deterministic ALG is *ρ-competitive* if for every input sequence σ, $|\text{ALG}(\sigma)| \geq \frac{1}{\rho} \cdot |\text{OPT}(\sigma)|$. An online randomized algorithm is ρ-competitive with respect to an oblivious adversary, if for every input sequence σ, $\mathbb{E}[|\text{ALG}(\sigma)|] \geq \rho \cdot |\text{OPT}(\sigma)|$, where the expected value is over the random choices made by ALG [8].

2.4 Problem Statement

The Input The online input is a sequence of packet requests $\sigma = \{r_i\}_i$. Each packet request is specified by a 4-tuple $r_i = (a_i, b_i, t_i, d_i)$ over a grid network $G = (V, E)$.

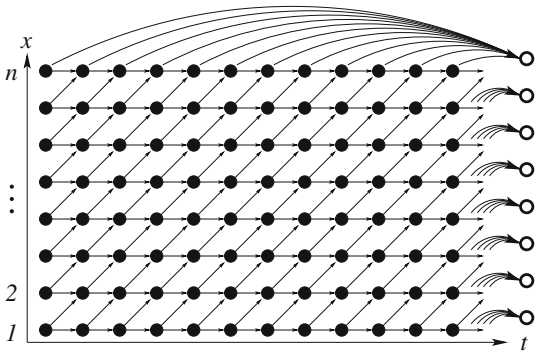


Fig. 2 The space-time graph G^{st} with the new sink nodes (shown on the rightmost column)

We consider an online setting, namely, the requests arrive one-by-one, and no information is known about a packet request r_i before its arrival.

The Output In each time step, the packet-routing algorithm decides what each of the packets in the network should do. This decision can be either reject a new packet, preempt an existing packet, store a packet in a buffer of the node which the packet has reached, or forward the packet to a neighboring node.

The Objective The goal is to maximize the number of packets that are successfully routed (i.e., reach their destination before the deadline expires).

3 Reduction of Packet-Routing to Path Packing

3.1 Space-Time Transformation

A *space-time transformation* is a method to map traffic in a directed graph over time into a directed acyclic graph [1,6,7,22]. Consider a directed graph $G = (V, E)$ with edge capacities c and buffer size B. The space-time transformation of G is the acyclic directed infinite graph $G^{st} = (V^{st}, E^{st})$ with edge capacities $c^{st}(e)$, where: (i) $V^{st} \triangleq V \times \mathbb{N}$. (ii) $E^{st} \triangleq E_0 \cup E_1$ where $E_0 \triangleq \{(u,t) \to (v,t+1) : (u,v) \in E \ , \ t \in \mathbb{N}\}$ and $E_1 \triangleq \{(u,t) \to (u,t+1) : u \in V, t \in \mathbb{N}\}$. (iii) The capacity of all edges in E_0 is c, and all edges in E_1 have capacity B. Note that the space-time graph corresponding to a d-dimensional grid is a $(d+1)$-dimensional grid. Figure 3a depicts the space-time transformation in the one dimensional case.

Adding Sink Nodes Following [7], we add sink nodes to define a specific destination node for each request. For every vertex v in the line, we define a sink node $\hat{v}$ (see Fig. 2). A *copy of a vertex* $v \in V$ in the space-time graph G^{st} is a space-time vertex $(v,t) \in V^{st}$ for some t. We add an incoming edge of infinite capacity to the sink node $\hat{v}$ from each tile s that contains a copy (v,t) of v.

3.2 Untilting

A standard drawing[2] of the space-time graph of a grid is a lattice generated by non-orthogonal vectors. This drawing is hard to depict and deal with, hence we apply a transformation called untilting defined as follows (see [22] for untilting in two dimensions).

We rectify the drawing of the space-time graph of a grid by applying an automorphism $q : \mathbb{Z}^{d+1} \to \mathbb{Z}^{d+1}$ defined by $q(x_1, \ldots, x_d, t) \triangleq (x_1, \ldots, x_d, t - \sum_{i=1}^{d} x_i)$. We refer to this transformation as *untilting*. The sole purpose of applying untilting is to obtain a drawing of the space-time graph of a grid in which the edges are axis parallel. Such an axis parallel drawing simplifies the definition of tiles. Note that the image of some of the vertices in G^{st} is outside the positive quadrant. Figure 3b depicts the untilted space-time graph in the one dimensional case. [e.g., the node $(2, 1)$ is mapped to $(2, -1)$.]

3.3 Tiling

The term *tiling* refers to a partitioning of the nodes of the space-time graph G^{st} into finite sets with identical geometric "shape".

Tiling is obtained by a partitioning of $\mathbb{Z}^{d+1}$ by disjoint $(d+1)$-dimensional cubes with side-length k. (For the sake of simplicity $\mathbb{Z}^{d+1}$ is partitioned to cubes. One can save a logarithmic factor in the competitive ratio by a partitioning to boxes with unequal side length. See Sect. 7.2 for an example of such a partitioning).

A tile s is a maximal subset of V^{st} such that its image $q(s)$ (after untilting) is contained in a cube. Formally, given a cube side-length k, a tile is defined by its *lower corner* $p \in \mathbb{Z}^{d+1}$, where the coordinates of p are integral multiples of k. The lower corner p defines the tile $s_p \triangleq \{v \in V^{st} : p \leq q(v) < p + k \cdot \vec{1}\}$, where $\vec{1}$ is the all ones vector. Note that some of the tiles in V^{st} are *partial*, namely contain less than k^d vertices (see Fig. 3c, d). In this case, we augment partial tiles by dummy vertices so that they are complete. Note that a dummy vertex is never an internal vertex in a path between non-dummy vertices, and hence, this augmentation has no effect on routing.

3.4 The Sketch Graph

The sketch graph is the graph obtained from the space-time graph after coalescing each tile into a single node (sink nodes remain unchanged). There is a directed edge (s_1, s_2) between two tiles s_1, s_2 in the sketch graph if there is a directed edge $(\alpha, \beta) \in E^{st}$ such that $\alpha \in s_1$ and $\beta \in s_2$. The capacity $c(s_1, s_2)$ of an edge (s_1, s_2) in the sketch graph is simply the sum of the capacities of the edges in G^{st} from vertices in s_1 to vertices in s_2 (i.e., the capacity of a vertical edge between two tiles $c \cdot \tau$ and the capacity of a horizontal edge is $B \cdot Q$). Figure 3e depicts an untilted sketch graph of a space-time graph of a one dimensional grid.

[2] For $d = 2$, the G^{st} has a 3-dimensional standard drawing in which: (i) a node $(i, j, t) \in V^{st}$ is mapped to the point (i, j, t), and (ii) edges are mapped to straight segments between their endpoints.

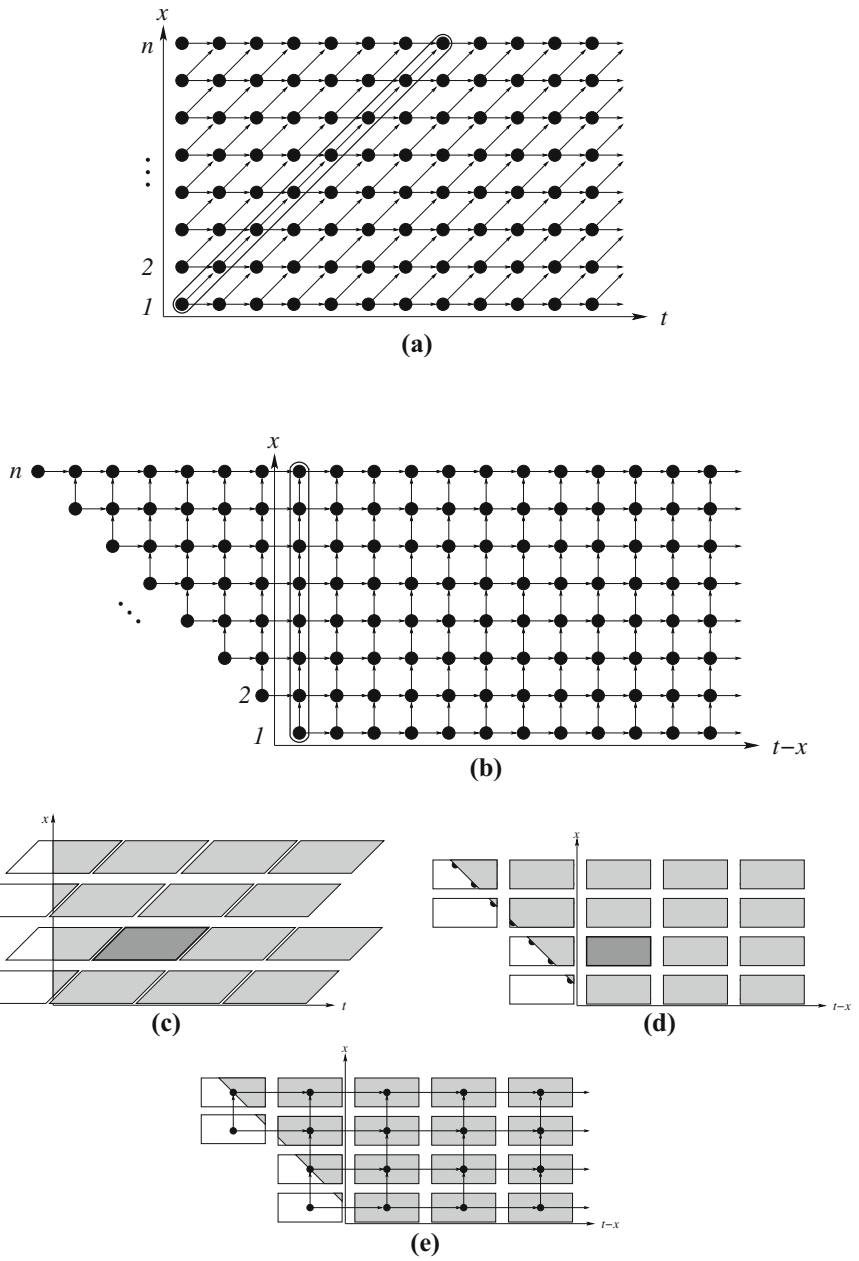


Fig. 3 **a** The tilted space-time graph G^{st}. The horizontal axis is the (infinite) time axis and the vertical axis is the (finite) node axis. **b** The untilted space-time graph G^{st}. The encapsulated path in (**a**) corresponds to the encapsulated path in (**b**). Diagonal edges depict edges in E_0. Edges in E_1 are depicted by horizontal edges. **c** The corresponding tiling of the (tilted) space-time graph G^{st}. The *bolded parallelogram with dark grey* in (**c**) corresponds to the *bolded rectangle with dark grey* in (**d**). **d** Tiling of the untilted space-time graph G^{st} by 2×4 rectangles. **e** The sketch graph over the tiles S

The sketch graph also has node capacities for nodes that correspond to tiles (i.e., not sinks). The capacity of every node that corresponds to a tile is $c(s) = 2 \cdot k^2 \cdot (B + c)$.

Notation We denote the sketch graph by $S = (V(S), E(S))$. We abuse notation and often refer to the nodes of S (that are not sinks) as tiles.

3.5 Online Packing of Paths

A reduction of packet routing to packing of paths is presented in Sect. 5.1. We briefly overview the topic of online packing of paths.

Consider a graph $G = (V, E)$ with edge capacities $c(e)$. Edges have soft capacity constraints (i.e., the capacity constraint may be violated, and one goal is to minimize the violation). The adversary introduces a sequence of connection requests $\{r_i\}_i$, where each request is a source-destination pair (a_i, b_i). The online packing algorithm must either return a path p_i from a_i to b_i or reject the request.

Consider a sequence $R = \{r_i\}_{i \in I}$ of requests. A sequence $P = \{p_i\}_{i \in J}$ is a (partial) *routing* with respect to R if $J \subseteq I$ and each path p_i connects the source-destination pair r_i. The *load* of an edge e induced by a routing P is the ratio $|\{p_j \in P : e \in p_j\}|/c(e)$. A routing P with respect to R is called a *β-packing* (or *β-feasible*) if the load of each edge is at most β. The *throughput* of a packing $P = \{p_i\}_{i \in J}$ is simply $|J|$.

An online path packing algorithm is (α, β)-*competitive* if it computes a β-packing P whose throughput is at least $1/\alpha$ times the maximum throughput over all 1-packings.

If each request is served by a single path, then the routing is *nonsplittable*.

A *fractional* packing is a multi-commodity flow. Each demand can be (partly) served by a combination of fractions of flows along paths. A sequence $P_f = \{P_i\}_{i \in I}$ is a *fractional (splittable) routing* with respect to R if each path $p_i \in P_i$ connects the source-destination pair r_i, and the total flow allocated by paths in P_i is at most one. The *throughput* of a fractional splittable path packing $P_f = \{P_i\}_{i \in I}$ is the sum of the allocated flows along every path in P_f. An optimal offline fractional packing can be computed by solving a linear program. Obviously, the throughput of an optimal fractional packing is an upper bound on the throughput of an optimal integral packing.

The proof of the following theorem appears in "Appendix 5". The proof is based on techniques from [2,10]. We refer to the online algorithm for online integral path packing by IPP.

Theorem 1 *Consider an infinite graph with edge capacities such that* $\min_e c(e) \geq 1$. *Consider an online path packing problem in which a path is legal if it contains at most* $p_{\max}$ *edges. Assume that there is an oracle, that given edge weights and a connection request, finds a lightest legal path from the source to the destination. Then, there exists a* $(2, \log(1 + 3 \cdot p_{\max}))$*-competitive online integral path packing algorithm. Moreover, the throughput is at least* $1/2$ *times the maximum throughput over all fractional packings.*

3.6 Polynomial Path Lengths

Notation Consider a directed graph $G = (V, E)$ over n vertices with edge capacities c and buffer size B in each vertex. Let G^{st} denote the space-time graph of G (see Sect. 3.1).

Consider a sequence $R = \{r_i\}_i$ of routing requests (without deadlines) over G^{st}, i.e., each request is a three-tuple $r_i = (a_i, b_i, t_i)$ that requires a path from (a_i, t_i) to a copy of b_i in G^{st}, that is, (b_i, t) for $t \geq t_i$.

Let $\textsc{opt}_f(R)$ denote an optimal fractional path packing in G^{st} with respect to $R = \{r_i\}_i$. Let $\textsc{opt}_f(R \mid p_{\max})$ denote an optimal fractional path packing in G^{st} with respect to $R = \{r_i\}_i$ under the constraint that each request is routed along a path of length at most $p_{\max}$. Let $|g|$ denote the throughput of a fractional path packing g.

The following lemma shows that bounding path lengths (in a fractional path packing problem over a space-time graph) by a polynomial decreases the throughput only by a constant factor. The lemma is an extension of a similar lemma from [7]. The proof of Lemma 2 appears in "Appendix 1".

Lemma 2 *Let* $p_{\max} \geq 2n \cdot (1 + \frac{B}{c})$. *Then,*

$$|\textsc{opt}_f(R \mid p_{\max})| \geq \frac{1}{2} \cdot \left(1 - \frac{1}{e}\right) \cdot |\textsc{opt}_f(R)| \,.$$

We remark that a trivial lower bound on the path lengths is $\Omega(B/c)$ if we want to be able to route a constant fraction of the optimal throughput. Indeed, if B packets are injected simultaneously to the same node in a line, then at most c packets can be forwarded in each step. Hence $\Omega(B/c)$ steps are required to forward a constant fraction of the packets. This justifies the term B/c in the definition of the maximum path length (see Lemmas 2 and 19).

4 Outline of the Deterministic Algorithm

The listing of the deterministic framework appears in Algorithm 1. Upon arrival of a request r_i, the algorithm reduces the packet request to an online integral path packing over the sketch graph with bounded paths. The algorithm then executes the online algorithm for online integral path packing (IPP) with respect to this path request. If the path request is rejected by the IPP algorithm, then the algorithm rejects r_i. Otherwise, let $\hat{p}_i$ denote the sketch path assigned to the request r_i. The algorithm injects the request r_i with its sketch path $\hat{p}_i$ and performs detailed routing in the space-time graph G^{st}. Detailed routing in G^{st} may fail (see Sect. 5.2). In case of failure, the algorithm preempts r_i.

To simplify the description, we begin in, Sect. 5, by presenting a detailed description and proof for the one-dimensional case. The required modifications for higher dimensions are described in Sect. 6. We also assume that there are no deadlines (i.e., $d_i = \infty$), hence each packet is specified by a 3-tuple $r_i = (a_i, b_i, t_i)$; we reintroduce deadlines in Sect. 5.4.

Algorithm 1 The deterministic framework. The algorithm receives a sequence of packet requests over the network $G = (V, E)$ and it either rejects, injects, or preempts these packet requests. A packet arrives at its destination if it is not rejected or preempted. The deterministic algorithm executes the IPP algorithm as a sub-procedure.

Upon arrival of a packet request $r_i = (a_i, b_i, t_i)$, for $i \geq 1$ (if r_i is rejected or preempted in any step, then the algorithm does not continue with the next steps), the algorithm proceeds as follows:

1. Reduce $r_i = (a_i, b_i, t_i)$ to a path request $\hat{r}_i$ in the $\{1, 2, \infty\}$-sketch graph $\hat{S}$ as follows:
   (a) The source of the path request $\hat{r}_i$ is the half tile s_{in}, where the tile s contains the vertex (a_i, t_i).
   (b) The destination of the path request $\hat{r}_i$ is simply the sink $\hat{b}_i$.
2. Execute the IPP algorithm over $\hat{S}$ with respect to the reduced path request $\hat{r}_i$.
   (a) If the IPP algorithm rejects the $\hat{r}_i$ then **reject** r_i.
   (b) Else, let $\hat{p}_i$ denote the path output by IPP, i.e., the sketch path assigned to $\hat{r}_i$.
3. **Inject** the request r_i (the request "includes" its sketch path $\hat{p}_i$) and perform detailed routing in the space-time graph G^{st}. Detailed routing proceeds by processing the *first segment* of $\hat{p}_i$, the *internal segments* of $\hat{p}_i$, the *last segment* of $\hat{p}_i$, and finally the *last tile* of $\hat{p}_i$. Failure in one of these parts causes a **preemption** of r_i.
4. Packet request r_i **arrives** to its destination b_i if it is not rejected or preempted.

5 The One Dimensional Case

In this section we present the details of Algorithm 1 for $d = 1$. We refer to Algorithm 1 by (an outline of algorithm for $d = 1$ is depicted in Figs. 4, 5).

Parameters The parameters of the uni-directional line network G are: n nodes, buffer size B in each node, and the capacity of each link is c. We assume that $B, c \in [3, \log n]$. Let $p_{\max} = 2n \cdot (1 + \frac{B}{c}) = O(n \cdot \log n)$. Let $k \triangleq \lceil \log(1 + 3p_{\max}) \rceil$. The length of a tile's side is k.

Proposition 3 *If $B, c \leq \log n$, then (i) $k = O(\log n)$, and (ii) the capacity of each edge in the sketch graph is at most $k \cdot \max\{B, c\} = O(\log^2 n)$.*

5.1 Reduction to Online Integral Path Packing

Downscaling of Capacities We regulate the number of paths that traverse each edge and node in the sketch graph by downscaling capacities. There are three types of

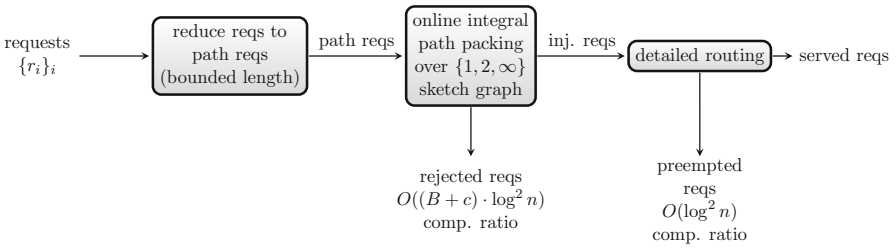


Fig. 4 An outline of the deterministic routing algorithm

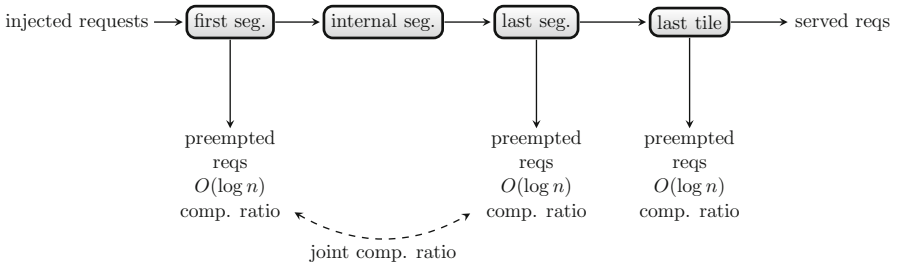


Fig. 5 An outline of the detailed routing algorithm

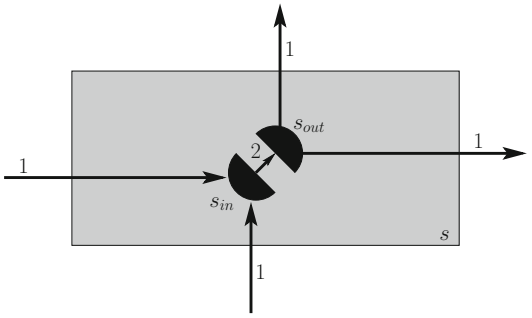


Fig. 6 Capacity assignment in the $\{1, 2, \infty\}$-sketch graph $\hat{S}$. Unit capacities are assigned to sketch edges and capacity of 2 is assigned to interior edges

capacities: (1) edges between tiles are assigned unit capacities, (2) incoming edges to sink nodes are unchanged and remain with infinite capacities, and (3) each tile is assigned two units of capacity.[3]

To apply a reduction to integral path packing, we reduce node capacities to edge capacities. Namely, each node $s \in V(S)$ is split to two "halves" s_{in} and s_{out}. After the split, edges are "redirected" as follows: the incoming edges of s enter s_{in} and the outgoing edges of s emanate from s_{out}. We add an additional edge called an *interior edge* between s_{in} and s_{out}. All interior edges are assigned two units of capacity (see Fig. 6). We refer to the augmented sketch graph with these capacities as the $\{1, 2, \infty\}$-*sketch graph*. We denote the $\{1, 2, \infty\}$-sketch graph by $\hat{S}$. Let $\hat{c} : E(\hat{S}) \to \{1, 2, \infty\}$ denote the downscaled capacity function of the $\{1, 2, \infty\}$-sketch graph $\hat{S}$.

Note that, since nodes are split and sinks are added, we need to increase the maximum path length to $p_{\max} \leftarrow 2 \cdot p_{\max} + 1$.

The Reduction A request $r_i = (a_i, b_i, t_i)$ to deliver a packet is reduced to a path request $\hat{r}_i$ in the $\{1, 2, \infty\}$-sketch graph $\hat{S}$. The source of the path request $\hat{r}_i$ is the vertex s_{in}, where the vertex (a_i, t_i) is in tile s. The destination of the path request is simply the sink node $\hat{b}_i$.

[3] In the case of d-dimensional grid, the capacity of a tile is $d + 1$. This saves a factor of d in the competitive ratio.

The sole purpose of the sink node is for a clean reduction to path packing. Once the IPP algorithm returns the sketch path $\hat{p}_i$, the sink node is removed from $\hat{p}_i$, and the last tile in the sketch path is regarded as the end of the sketch path.

Theorem 1 implies that the IPP algorithm returns an integral packing of paths in $\hat{S}$ that is $(2, k)$-competitive with respect to the optimal fractional path packing in $\hat{S}$. The length of each path in the packing is at most $p_{\max}$.

5.2 Detailed Routing

This section deals with the translation of paths in the sketch graph to paths in the space-time graph. This translation, called *detailed routing*, is adaptive and computed in a distributed on-the-fly fashion. The detailed path respects the sketch path in the sense that it traverses the same tiles and bends only where the sketch path bends. Note that, some of the packets are dropped during detailed routing.

More formally, the goal in detailed routing is to compute a (detailed) path p_i in the space-time graph G^{st} given a sketch path $\hat{p}_i$ in the $\{1, 2, \infty\}$-sketch graph $\hat{S}$. The projection of p_i on $\hat{S}$ equals $\hat{p}_i$.

5.2.1 Preliminaries

Terminology A *bend* in the sketch path is a node in which the sketch path changes direction, i.e., vertical to horizontal or horizontal to vertical.

A *segment* of a path in a grid is a maximal subpath, all the vertices of which belong to the same row or column of the grid. A segment is *special* if it is the first or the last segment of a path. Otherwise, it is an *internal* segment.

We refer to the side through which the detailed path enters a tile as the *entry side*. Similarly, we refer to the side through which the detailed path exits a tile as the *exit side*.

Packing Intervals Online The problem of packing intervals in a line is defined as follows.

1. Input: A set $I = \{p_i\}_{i=1}^{r}$, where each p_i is an open interval $(a_i, b_i) \subseteq (1, n)$.[4]
2. Output: A maximum cardinality subset $I' \subseteq I$ of pairwise disjoint intervals.

In the online setting, we assume that the intervals appear one by one, and that $a_1 \leq a_2 \leq \cdots \leq a_r$. The online algorithm must maintain a maximum subset I' such that (i) I' is a subset of the prefix of the intervals input so far, and (ii) the intervals in I' are pairwise disjoint.

The online algorithm is based on an optimal algorithm for maximum independent sets in interval graphs [17]. Upon arrival of an interval $p_i = (a_i, b_i)$, the algorithm proceeds as follows: (1) If p_i does not intersect the intervals in I', then p_i is added to I'. (2) Else, p_i intersects an interval $p_j = (a_j, b_j)$. If $b_i > b_j$, then p_i is rejected

[4] We consider open intervals rather than closed intervals. One could define the problem with respect to closed intervals, but then instead of requiring disjoint intervals in the packing, one would need to require that intervals may only share endpoints.

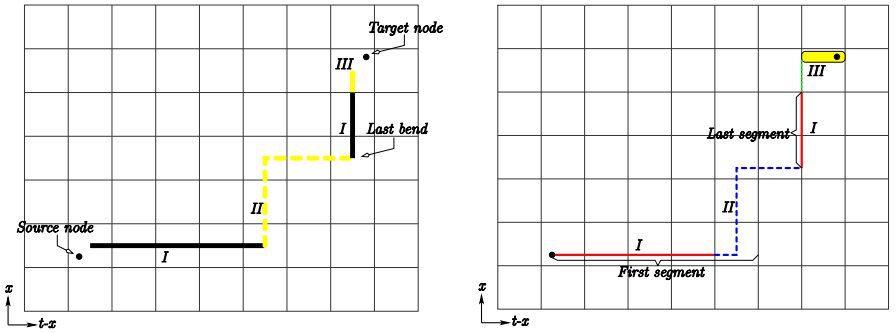


Fig. 7 The untilted space-time graph G^{st} is partitioned into tiles depicted by *square rectangles*. These tiles are the vertices of the $\{1, 2, \infty\}$-sketch graph $\hat{S}$, in fact, two neighboring squares correspond to two neighboring vertices in $\hat{S}$. **a** The sketch path $\hat{p}_i$ is overlayed on G^{st}. We partition $\hat{p}_i$ into three parts: (I) first and last segments, which are depicted by *solid* segments, (II) internal segments, which are depicted by *dashed* segments, and (III) routing in the last tile, which is depicted by a *grey line*. The source node of the packet request is in the first tile of the sketch path, the target node of the packet request is in the last tile of the sketch path. **b** The detailed path p_i is depicted by a *thin line* that traverses the same tiles traversed by the sketch path $\hat{p}_i$. The detailed routing of the first segment is depicted by the *horizontal line* emanating from the source node. The *dashed line* depicts the detailed routing after the first segment. The detailed routing of the last segment takes a turn on the entry side of the tile that contains the last bend. The detailed routing in the last tile is depicted by an straight *dotted thin line*. The space-time copies of b_i are depicted by the *grey rectangle* that surrounds the target node. The intervals that are input to the interval packing algorithm are depicted by braces

(namely, I' remains unchanged). Otherwise, if $b_i \leq b_j$, then p_i preempts p_j (namely, $I' = (I' \cup \{p_i\}) \setminus \{p_j\}$).

Note, that this online algorithm can be executed in a distributed fashion in a line. Namely, the local input of each processor a_i is the interval $p_i = (a_i, b_i)$ (or the empty input). Additionally, a_i receives I' from its neighbor a_{i-1}. Now, a_i can verify by itself whether to preempt an interval from I' and accept p_i or to reject p_i. After a_i completes his local computation, a_i sends I' to its neighbor a_{i+1}.

Partitioning of Detailed Routing Detailed routing is partitioned into at most three parts,[5] as follows (See Fig. 7b).

(I) Special segments,
(II) Internal segments, and
(III) Last tile: detailed routing in the last tile deals with routing the request from the point that it enters the last tile till a copy of the destination vertex within the tile.

Preemptions may occur in parts (I) and (III) of the detailed routing. Preemptions are caused by conflicts between detailed routing of packets that belong to the same part. Namely, a special segment can only preempt another special segment. Similarly, detailed routing in the last tile preempts only routes that end in the same tile.

[5] Degenerate cases of detailed routing consist of two parts or just a single part; for example, detailed routing of requests whose sketch path is a single tile consists only of part (III).

Reservation of Capacities The algorithm reserves one unit of capacity in each edge $e \in E^{st}$ for each part of detailed routing. This is the reason for the requirement that $B, c \geq 3$. Note that the algorithm is wasteful in the sense that it only uses 3 units of capacity in each edge. We refer to each of these 3 units of capacity as a *track*, i.e., each part uses a different track.

5.2.2 Detailed Routing in Special Segments

Consider the first segment of a sketch path $\hat{p}_i$ (see Fig. 7a). The detailed routing corresponding to this segment is a straight path that starts in the source-vertex (a_i, t_i) and ends in the tile in which $\hat{p}_i$ bends for the first time. As there may be contention for capacity allocated for special segments, detailed routing needs to decide which request is dropped. We reduce the problem of routing the first segment of detailed paths to the problem of packing intervals in a line (described in detail in Sect. 5.2.1).

A separate reduction to interval packing in a line takes place for every row and column of the untilted space-time grid.

Detailed routing in the last segment of $\hat{p}_i$ (before the last tile) is similar. Consider a last segment of a sketch path $\hat{p}_i$ that starts in tile s_1 and ends in tile s_2. The detailed routing of a last segment begins in the entry side of s_1 that is reached by the detailed routing of the previous segment, and ends in the entry side of s_2. Between these two endpoint, detailed routing is along a straight path. As in the case of detailed routing of the first segment, routing in the last segment is reduced to interval packing in a line.

Consider a sketch path $\hat{p}_i$ whose first bend is in tile s. Suppose that the detailed routing of the first segment of $\hat{p}_i$ is not preempted before it enters the tile s. We claim that detailed routing will not preempt p_i before it performs a "turn" in s. Indeed, due to distinct tracks used in each part, it suffices to focus on conflicts with first or last segments of other requests. There are two types of conflicting requests whose first segment conflicts with the first segment of r_i depending on the relative location of the source vertex (either before or after the entry to tile s). If the source vertex of r_j appears before the entry to s, then r_i since r_i enters s, we know that r_i "wins" and r_j is preempted. If the source vertex of r_j appears after the entry to s, then r_i "wins" again because $\hat{p}_j$ has a first segment, and therefore r_j requests an interval that ends outside the tile s while r_i requests an interval that ends in tile s. We also need to consider a conflict with a last segment of a request r_j: (1) If r_j ends inside s, then it must also begin in s (because it is not possible for r_i and r_j to enter the tile through the same edge). If r_j begins and ends s, then it is routed using only the third track (reserved for detailed routing in the last tile) and r_j does not conflict with the first segment of r_i. (2) If r_j ends outside s, then it is preempted by r_i because r_i requests an interval that ends inside s.

5.2.3 Detailed Routing in Internal Segments

Detailed routing of internal segments takes place in a tile as follows. Fix a node v. The node v has two incoming edges and two outgoing edges. We denote these edges by $horz_{in}$, $vert_{in}$ and $horz_{out}$, $vert_{out}$. We refer to the request that traverses an edge e by $e.r$. For example, $horz_{in}.r$ is the name of the request that enters v via the horizontal

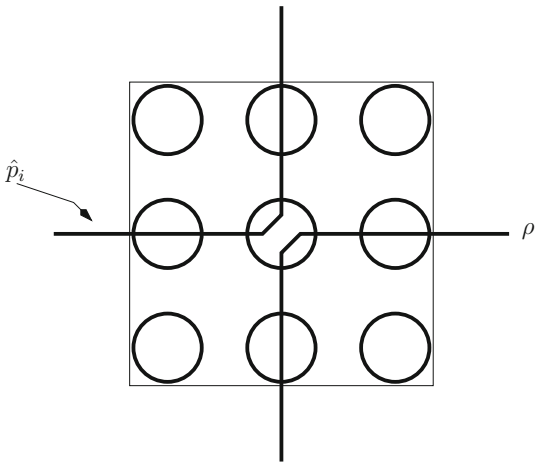


Fig. 8 A knock-knee bend in detailed routing in G^{st}. Space-time nodes are depicted by *white circles*. The detailed route of $\hat{p}_i$ makes a turn in the vertical direction, thus freeing the suffix of the row ρ. The conflicting detailed route takes a turn in *horizontal* direction, thus freeing the suffix of the column in the *vertical* direction

edge. If an edge e is not assigned to a request, then we set $e.r$ to null. The rules for detailed routing of these paths are as follows:

1. If one of the incoming edges e is not assigned to a request, then the other edge e' (if $e'.r$ is not null) chooses the outgoing edge according to its exit side.
2. (Precedence to straight traffic.) Else, if the exit side of $horz_{in}.r$ is east or the exit side of $vert_{in}.r$ is north, then the paths continue without a bend, namely, $horz_{out}.r \leftarrow horz_{in}.r$ and $vert_{out}.r \leftarrow vert_{in}.r$.
3. (Simultaneous bends.) Else, a knock-knee bend takes place, namely, $horz_{out}.r \leftarrow vert_{in}.r$ and $vert_{out}.r \leftarrow horz_{in}.r$ (see Fig. 8).

We claim that detailed routing in an internal segment always succeeds. If the detailed path is headed towards its exit side (e.g., traverses the tile without a bend), then detailed routing gives it priority so that it reaches its exit side. If the sketch path bends in the tile, then the detailed path must encounter either a null path or another detailed path that also bends in the tile (in which case the path takes the required turn). This is true because, otherwise, there would be more than k paths that exit the tile from the same side, contradicting the congestion guarantee by the IPP algorithm (that at most k paths traverses the edges between tiles).

We now deal with transitions from part (I) to part (II) of detailed routing. Recall, that each part of the detailed path uses a different track. Consider a sketch path $\hat{p}_i$ whose first bend is in tile s. If the detailed routing of $\hat{p}_i$ reaches s, then it is not preempted by another special segment (see Sect. 5.2.2). As in detailed routing in internal segments, the detailed route of $\hat{p}_i$ in tile s bends when it meets a null path or a detailed path that also wants to bend. The same argument shows that such a bend is always successful. After the bend, the path transitions from the first track to the second track.

We conclude that detailed routing is always successful in internal segments.

5.2.4 Detailed Routing in the Last Tile

We refer to requests whose sketch path is a single tile as *near* requests. Note that detailed routing of a near request consists only of part (III).

Detailed routing in the last tile routes a path along a straight vertical path from the entry point to the row in the tile that corresponds to the destination node. Note that if the destination vertex of r_i is b_i, then it suffices to route the path to one of the space-time copies of b_i. Hence, every copy of b_i in the tile is a valid destination. Contentions occur only in each column, and a path with a closest destination preempts the conflicting paths.

5.3 Analysis of the Algorithm for $d = 1$

Recall that the length of a tile's side is $k = \lceil \log(1 + 3p_{\max}) \rceil$. Moreover, in the case where $B, c \in [3, \log n]$, it follows that $k = O(\log n)$.

Theorem 4 *The competitive ratio of the algorithm for uni-directional line networks is* $O(\log^5 n)$ *provided that* $B, c \in [3, \log n]$.

Proof sketch of Theorem 4 The algorithm starts with the path packing algorithm IPP over the $\{1, 2, \infty\}$-sketch graph. This means that capacities are reduced by a factor of at most $O(k^2 \cdot \max\{B, c\}) = O(k^3)$ (by the capacity assignment "inside" a tile and "between" tiles). The fact that path lengths are bounded by $p_{\max}$ reduces the throughput only by a constant factor. The throughput of algorithm IPP is $O(1)$-competitive.

Detailed routing succeeds in routing at least a k^2 fraction of the sketch paths. There are two causes for loss of packets: routing of special segments and routing in the last tile. Routing of special segments (i.e., first and last segment) succeeds for a fraction of $1/k$. we show that the success rate is not multiplied and that the success rate for special segments is $1/2k$. Routing in the last tile succeeds for a fraction of $1/2k$ per tile. Putting things together we get a competitive ratio of $O(k^5)$, as required. □

Note that the Theorem 4 actually applies for $B, c \in [3, O(\log n)]$. The constant in the $O(\log n)$ linearly affects the constant in the competitive ratio of the algorithm.

Notation Let R be a fixed sequence of packet requests introduced by the adversary. Let $R_s \subseteq R$ denote the set of requests whose sketch path ends in tile s. For every $X \subseteq R$ and for every tile s let $X_s \triangleq X \cap R_s$. We interpret requests in R as path requests in G^{st}. Let OPT (respectively OPT_f) denote a maximum integral (respectively fractional) packing of paths from R in G^{st}. Let IPP(R) denote the set of requests that algorithm IPP injected when given input R. For brevity, we denote IPP(R) simply by IPP. Similarly, let ALG denote the set of requests that ALG routed to their destination. Let IPP$' \subseteq$ IPP denote the set of requests that are not preempted before they reach the entry side of their last tile. (Note that ALG $\subseteq$ IPP$' \subseteq$ IPP $\subseteq R$.) Let f^* denote an optimal fractional flow with respect to R over the sketch graph S. Let $f^*_{\{1,2,\infty\}}$ denote an optimal fractional flow with respect to R over the $\{1, 2, \infty\}$-sketch graph $\hat{S}$. (Note that OPT and OPT_f are packings of paths in G^{st}, while f^* and $f^*_{\{1,2,\infty\}}$ are packings

in sketch graphs.) Let $\text{OPT}_f(R \mid p_{\max})$ denote an optimal fractional path packing in G^{st} with respect to R under the constraint that each request is routed along a path of length at most $p_{\max}$. Let $f^*(R \mid p_{\max})$ denote an optimal fractional flow in the sketch graph S with respect to R under the constraint that flow paths have a length of at most $p_{\max}$. Let $f^*_{\{1,2,\infty\}}(R \mid p_{\max})$ denote an optimal fractional flow in the $\{1, 2, \infty\}$-sketch graph $\hat{S}$ with respect to R under the constraint that flow paths have a length of at most $p_{\max}$. Let $|g|$ denote the throughput of flow g.
We now present a detailed proof of Theorem 4, based on the following propositions.

Proposition 5 $|f^*(R \mid p_{\max})| \geq |\text{OPT}_f(R \mid p_{\max})|$.

Proof Consider a fractional packing h of paths in G^{st} in which paths lengths are bounded by $p_{\max}$. Let g denote the flow in sketch graph S where $g(e)$ is simply the sum of the flows of h along the edges in G^{st} that are coalesced to e in S. Clearly, $|g| = |h|$. We claim that g is a feasible fractional flow in the sketch graph S whose flow paths are not longer than the flow paths in h. (In fact, they are shorter by a factor of k.)

We show that the flow g satisfies the capacity constraints in S as follows. If e is a sketch edge between tiles, then, by linearity, the capacity constraint is satisfied. We now focus on interior edges. The amount of flow in h that traverses a tile in G^{st} is bounded by the sum of the capacities of the edges in the tile, namely, it is at most $(B + c) \cdot k^2$. It follows that the amount of flow in g that traverses a node (that corresponds to a tile) in the sketch graph is bounded by the node's capacity (which equals $2 \cdot k^2 \cdot (B + c)$). We conclude that g is a feasible flow in S, and the proposition follows. □

Proposition 6 $k^2 \cdot (B + c) \cdot |f^*_{\{1,2,\infty\}}(R \mid p_{\max})| \geq |f^*(R \mid p_{\max})| \geq |f^*_{\{1,2,\infty\}}(R \mid p_{\max})|$

Proof Recall that f^* is a maximum flow in the sketch graph S while $f^*_{\{1,2,\infty\}}$ is a maximum flow in $\hat{S}$. The proof is a direct consequence of the following bounds between capacities in S and in $\hat{S}$.

For every edge e that is both in S and in $\hat{S}$, we have

$$k \cdot (B + c) \cdot \hat{c}(e) \geq c(e) \geq \hat{c}(e). \tag{1}$$

For every node s that corresponds to a tile, we have

$$c(e) = k^2 \cdot (B + c) \cdot \hat{c}(e). \tag{2}$$

□

Proposition 7 $|\text{IPP}| \geq \left(\frac{1}{2 \cdot k^2 \cdot (B+c)}\right) \cdot |f^*(R \mid p_{\max})|$

Proof By Theorem 1 (i.e., $(2, k)$-competitiveness of IPP),

$$|\text{IPP}| \geq \frac{1}{2} \cdot f^*_{\{1,2,\infty\}}(R \mid p_{\max}).$$

Downscaling of capacities implies

$$f^*_{\{1,2,\infty\}}(R \mid p_{\max}) \geq \left(\frac{1}{k^2 \cdot (B+c)}\right) \cdot |f^*(R \mid p_{\max})|,$$

and the proposition follows. □

The following proposition proves that a fraction of at most $(1 - \frac{1}{2k})$ of the requests in IPP are preempted before they reach their last tile.

Proposition 8 $|\text{IPP}'| \geq \frac{1}{2k} \cdot |\text{IPP}|$

Proof Consider a row or a column L of nodes in G^{st}. Let $R \cap L$ denote the set of requests that contain special segments that compete over edges in L. From the point of view of L, each request $r_i \in R \cap L$ is a request for an interval $I_i \subseteq L$. As described in Sect. 5.2.1, the detailed routing of the requests $R \cap L$ along L simulates an optimal interval packing algorithm. In particular, the simulation has the property that if an interval $I_i = (a_i, b_i)$ preempts an interval $I_j = (a_j, b_j)$, then the intervals overlap and $b_i \leq b_j$. Hence, the edge $(b_i - 1, b_i)$ is in I_j.

Focus on preemptions that occur during the detailed routing of first segments (the case of last segments is similar). Consider the "forest of preemptions" over the intervals, where the set of intervals that were preempted by I_i are children of I_i. We claim that if interval I_j is a descendant of I_i in this forest, then the edge $(b_i - 1, b_i)$ is in I_j. The proof is by induction on the distance between I_i and I_j in the forest of preemptions. The induction basis holds for a child I_j by the discussion above. Suppose that I_k preempted I_j (hence $b_k \leq b_j$). Since I_k is a descendent of I_i, by the induction hypothesis $(b_i - 1, b_i)$ is an edge in I_k. Because I_j is preempted by I_k in a vertex to the left of b_i, it follows that the edge $(b_i - 1, b_i)$ is in I_j, as required. By Theorem 1, the load induced by IPP on each $\{1, 2, \infty\}$-sketch edge is at most k. Therefore, the maximum number of proper descendants of I_i in the forest is $(k - 1)$ (not including I_i).

Consider a bipartite graph of preemptions over $\text{IPP}' \cup (\text{IPP} \setminus \text{IPP}')$ (now we consider both first segments and last segments). There is an edge (r_i, r_j) if the request $r_i \in \text{IPP}'$ is an ancestor of the request $r_j \in (\text{IPP} \setminus \text{IPP}')$ in the forest of preemptions corresponding to detailed routing. Since a preempted request is preempted only once, the degree of the nodes in $\text{IPP} \setminus \text{IPP}'$ is one. Recall that each sketch path contains at most 2 special segments. By the discussion above, the degree of a node in IPP′ is bounded by $2 \cdot (k-1)$. By counting edges in the bipartite graph, we conclude that $|\text{IPP}'| \cdot 2 \cdot (k-1) \geq |\text{IPP} \setminus \text{IPP}'|$, and the proposition follows. □

The following proposition states that a fraction of at least $1/(2k)$ of the requests that reach their last tile are successfully routed.

Proposition 9 $|\text{ALG}| \geq \frac{1}{2k} \cdot |\text{IPP}'|$

Proof Since $\{\text{IPP}'_s\}_{s \in V(S)}$ is a partition of IPP′ and $\{\text{ALG}_s\}_{s \in V(S)}$ is a partition of ALG, it suffices to prove that $|\text{ALG}_s| \geq \frac{1}{2k} \cdot |\text{IPP}'_s|$ for every tile s.

Fix a tile s. Every sketch path of a request in IPP'_s traverses the interior edge of s in $\hat{S}$ whose capacity is 2. Theorem 1 implies that this capacity is violated by at most a factor of k, hence $|\text{IPP}'_s| \leq 2k$.

Detailed routing in the last tile successful routes at least one request from IPP'_s if $\text{IPP}'_s \neq \emptyset$, and the proposition follows. □

We now put things together to complete the proof of Theorem 4.

proof of Theorem 4 The proof is as follows.

$$
\begin{aligned}
|\text{ALG}| &\geq \frac{1}{2k} \cdot |\text{IPP}'| && \text{(by Prop. 9)} \\
&\geq \frac{1}{2k} \cdot \frac{1}{2k} \cdot |\text{IPP}| && \text{(by Prop. 8)} \\
&\geq \left(\frac{1}{8 \cdot k^4 \cdot (B+c)} \right) \cdot |f^*(R \mid p_{\max})| && \text{(by Prop. 7)} \\
&\geq \left(\frac{1}{8 \cdot k^4 \cdot (B+c)} \right) \cdot |\text{OPT}_f(R \mid p_{\max})| && \text{(by Prop. 5)} \\
&\geq \left(\frac{1}{8 \cdot k^4 \cdot (B+c)} \right) \cdot \frac{1}{2} \cdot \left(1 - \frac{1}{e} \right) \cdot |\text{OPT}_f(R)| && \text{(by Lemma. 2)} \\
&\geq \Omega \left(\frac{1}{k^4 \cdot (B+c)} \right) \cdot |\text{OPT}| \,.
\end{aligned}
$$

The last line holds because every integral path packing is also a fractional one. The theorem follows. □

5.4 Requests With Deadlines

In this section we present the modification needed to deal with packet requests with deadlines. The change to the algorithm is in the reduction to online integral path packing (see Sect. 5.1), i.e., we need to change the sink node in the reduction as described below.

Adding Sink Nodes for Requests with Deadlines A request to deliver a packet is of the form $r_i = (a_i, b_i, t_i, d_i)$, where d_i is the deadline. In terms of a path request in the space-time graph G^{st}, this means that we need to assign a path from (a_i, t_i) to a vertex (b_i, t'), where $t_i \leq t' \leq d_i$. Thus, the destination is a set of vertices rather than one specific vertex. We connect this set of destinations to a new sink. Formally, for every request r_i, introduce a new vertex $sink_i$ and connect every vertex in $\{(b_i, t')\}_{t'=t_i}^{d_i}$ to $sink_i$ with an edge of infinite capacity.

Now, a packet request $r_i = (a_i, b_i, t_i, d_i)$ is reduced to a path request in the $\{1, 2, \infty\}$-sketch graph from the half-tile s_{in} (where the tile s contains (a_i, t_i)) to $sink_i$. A path from (a_i, t_i) to $sink_i$ contains at most $d_i - t_i + 1$ edges. We still bound the path length by $p_{\max}$, as before, to obtain a load of $O(\log p_{\max})$ by IPP.

We claim that a request that is not preempted by detailed routing reaches its destination on time. To see this fix a packet request r_i that is not preempted by detailed

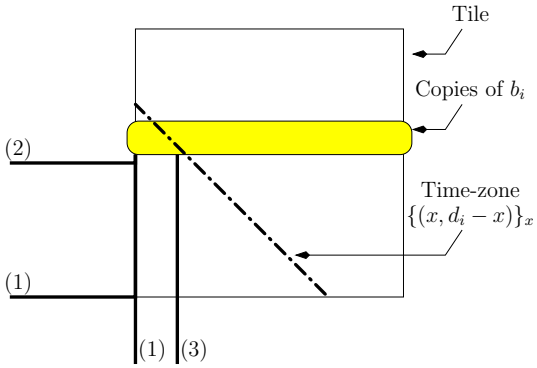


Fig. 9 The 3 possible starting points of detailed routing in a tile

routing, and let $\hat{p}_i$ denote its sketch path. Let s denote the tile in which $\hat{p}_i$ ends. We now show that the detailed path p_i ends in a vertex (b_i, t) such that $t \leq d_i$. There are 3 cases (see Fig. 9): (1) p_i enters s via a last segment from the south-west corner of s, (2) p_i enters s via a first segment from the west, or (3) p_i enters s via a first segment from the south.[6] In the first two cases, p_i enters s and moves north until it reaches a copy of b_i. The copy (b_i, t') of b_i that is reached must satisfy $t' \leq d_i$ if (b_i, d_i) is in the tile. Indeed, because s is the last tile of $\hat{p}_i$, the copy of b_i in the leftmost column of s lies below the "time-zone" $\{(x, d_i - x)\}_x$ in the untilted space-time graph. Moreover, the entry point of p_i to tile s lies below this copy of b_i (if it were above this copy of b_i, then it has already reached b_i). In the third case, p_i enters via the south side. This means that (before entering s) p_i consists only of a first segment, i.e., starting from its arrival the packet was forwarded and was not buffered at all. Since the deadlines are "feasible", i.e., the deadline $d_i \geq t_i + dist(a_i, b_i)$, where $dist(a_i, b_i)$ is the distance between a_i to b_i. The packet keeps moving north and reaches the copy of b_i at time $t_i + dist(a_i, b_i)$. It follows that the packet reaches its destination on time in this case as well. We conclude that requests that are not preempted reach their destination on time, as required.

6 Generalizations

In this section we present a generalization of the algorithm to the d-dimensional case as well as extensions to the special cases: bufferless grids and grids with large buffers\capacities.

The d-Dimensional Case The following modifications are needed to extend the algorithm to d-dimensional grids.

[6] Note that cases (2) and (3) are degenerate cases in the sense that the detailed routing consists only of the a first segment and routing in the last tile.

(1) $k = \lceil \log(1 + 3p_{\max}) \rceil$, where in the d-dimensional case

$$p_{\max} \triangleq 2 \cdot diam(G) \cdot \left(1 + n \cdot \left(\frac{B}{c} + d\right)\right).$$

In the case where $B, c \in [3, \log n]$, it follows that $k = O(\log n)$.

(2) Apply tiling with side length k, e.g., a face of a cube contains k^d vertices.

(3) Similarly to the 1-dimensional case, the sketch graph also has node capacities for nodes that correspond to tiles (i.e., not sinks). The capacity of every node that corresponds to a tile is $c(s) = (d + 1) \cdot k^{d+1} \cdot (B + d \cdot c)$. Edges in the sketch path have unit capacities.

(4) Similarly to the definition of $\{1, 2, \infty\}$-sketch graph, we define the $\{1, d+1, \infty\}$-sketch graph by assigning a capacity of $d + 1$ (instead of 2) to the interior edges.

(5) Detailed routing of internal segments is generalized as follows. Each node has $d+1$ incoming edges and $d+1$ outgoing edges. Fix a node v. Let $in_1, \ldots, in_{d+1}$ denote edges that enter v. Similarly, let $out_1, \ldots, out_{d+1}$ denote edges that exit v. Detailed routing in v proceeds as follows: For every $j \in [1, d + 1]$, let ℓ_j denote the exit side of request $in_j.r$ in the tile s that contains v.

   (a) (Precedence to straight paths.) If $\ell_j = j$, then $out_j.r = in_j.r$.

   (b) (Try next crossing.) Else, if the exit side of $in_{\ell_j}.r$ is not j or null, then $out_j.r = in_j.r$.

   (c) Else, if $in_{\ell_j}.r = j$ or ($in_{\ell_j}.r = null$ and j is the smallest index j' for which $in_{j'}.r = \ell_j$), then a knock-knee takes place: $out_{\ell_j}.r = in_j.r$ and $out_j.r = in_{\ell_j}.r$.

   (d) (Try next crossing.) Else, $out_j.r = in_j.r$.

   The key observation for detailed routing in an internal segment is that if a request r_i fails to bend at node v, then another request proceeds in v toward its exit side (in the tile that contains v). Thus, as a request r_i continues to try to turn in the next crossing, it crosses a new request that will exit the tile successfully. Since the number of requests in IPP that traverse the same sketch edge is at most k, it follows that r_i is bound to find a crossing in which it turns toward its exit side.

The following theorem bounds the competitive ratio of the algorithm for general dimensionality d. The proof of Theorem 10 is outlined in "Appendix 2".

Theorem 10 *The competitive ratio of the algorithm for d-dimensional grid networks is*

$$O\left(k^{d+3} \cdot (B + d \cdot c)\right) = O\left(\log^{d+4} n\right)$$

provided that $B, c \in [3, \log n]$.

Bufferless Grids For the case $B = 0$ and $c \geq 3$ (no upper bound on c), we obtain the following result. The proof of the following theorem is sketched in "Appendix 3".

Theorem 11 *There exists an online deterministic preemptive algorithm for packet routing in bufferless d-dimensional grids with a competitive ratio of $O(\log^{d+2} n)$.*

In the one dimensional case without buffers, the optimality of online interval packing implies that the nearest-to-go policy [4] is optimal.

Proposition 12 *Nearest-to-go is an optimal policy for packet routing in a line when* $B = 0$.

Large Buffers and Large Link Capacities In this section we consider the case that the size of the buffers and the capacities of the links are at least logarithmic.

Redefine the parameter ν, by

$$\nu \triangleq n^{O(1)}.$$

This of course influences $p_{\max}$ and k because $p_{\max} \triangleq p_{\max} \geq (\nu + 2) \cdot diam(G)$ and $k \triangleq \lceil \log(1 + 3p_{\max}) \rceil$. However, in this setting $p_{\max}$ is polynomial in n and $k = \Theta(\log n)$.

The following theorem shows that it is easy to achieve a logarithmic competitive ratio if $B/c = n^{O(1)}$ and $B, c \geq k$.

Theorem 13 *There exists an online deterministic algorithm for packet routing in d-dimensional grids with a competitive ratio of* $O(\log n)$ *if* $B/c = n^{O(1)}$, *and* $B, c \geq k$. *In this algorithm, packets are either rejected or routed but not preempted.*

Proof Scale B and c by setting $B' \leftarrow \lfloor \frac{B}{k} \rfloor$ and $c' \leftarrow \lfloor \frac{c}{k} \rfloor$. Run the IPP algorithm over the space-time graph G^{st} with the scaled capacities B' and c' to decide which requests are rejected and which are routed. We claim that the routes computed by the IPP algorithm are a valid routing. Indeed, IPP is $(2, k)$-competitive with respect to B' and c'. Hence, the same packing of paths is $(O(k), 1)$-competitive with respect to B and c. The theorem follows since $k = O(\log n)$. □

7 A Randomized Algorithm for the One Dimensional Case

In this section we design and analyze a randomized algorithm for routing packets in uni-directional line networks. Our randomized algorithm achieves a competitive ratio of $O(\log n)$.

The randomized algorithm applies only to the setting in which requests are without deadlines (i.e., $d_i = \infty$), hence each packet is specified by a 3-tuple $r_i = (a_i, b_i, t_i)$.

The randomized algorithm deals with all values of buffer sizes and communication link capacities in the range $[1, O(\log n)]$. We do not require that $B, c \geq 3$ as in the deterministic algorithm.

In particular, it holds also for unit buffers. In Sects. 7.3, 7.4, 7.5 and 7.6 we deal with the case that both B and c are in $[1, \log n]$. We consider this case to be the most interesting one. In Sect. 7.7 we deal with the case of $\log n \leq B/c \leq n^{O(1)}$. In Sect. 7.8 we deal with the case of $B \in [1, \log n]$ and $c \in [\log n, \infty)$ (for a summary of the results presented in this section, see Table 2).

Table 2 Values of B and c in which our algorithm achieves logarithmic competitive ratio. In particular, it holds also for unit buffers, i.e., $B = 1$. We consider the first case to be the most interesting one

B	c	Sections
$[1, \log n]$	$[1, \log n]$	7.3, 7.4, 7.5 and 7.6
$[\log n, \infty)$	$\left[\lceil \frac{B}{n^{O(1)}} \rceil, \frac{B}{\log n}\right]$	7.7
$[1, \log n]$	$[\log n, \infty)$	7.8

7.1 Outline of Modifications

Our goal is to reduce the $O(\log^5 n)$ competitive ratio of the deterministic algorithm (see Theorem 4) to a logarithmic competitive ratio with the help of randomization. In this section we outline the techniques that are employed to achieve this goal.

In the randomized algorithm, the online integral packing algorithm is applied to the sketch graph (without downscaling of capacities). To simplify the discussion assume that $B = c = 1$. Since the load on every edge in the sketch graph is at most k, and k also equals the length of the tile side, this implies that $O(k^2)$ paths traverse each tile side.

The ratio between the area and the perimeter of a tile is $\Theta(k)$. As the number of requests that start in a tile is proportional to the area of a tile, and the number of requests that can enter or exit a tile is proportional to the perimeter of a tile, we need to avoid losing a factor of $\Theta(k)$ in the competitive ratio. We do this by *randomly sparsifying* the requests. The goal of this sparsification is to leave a $\Theta(1/k)$ fraction of the requests so that a constant fraction of the remaining requests can be routed out of their starting tile.

To facilitate detailed routing, we consider three (non-disjoint) areas within each tile: (1) a part in which new requests may start, (2) a part dedicated to routing, and (3) a part in which requests reach their destination. The tiles are randomly shifted so that a constant fraction of the requests "agree" with the designated parts in the tiles.

Detailed routing of requests not rejected by the IPP algorithm or by random sparsification is simpler and always succeeds.

7.2 Preliminaries

Tiling The untilted space-time graph G^{st} is partitioned into rectangular tiles. We denote length of each tile by τ and the height by Q (we also require that τ and Q are even). Note that tiles may not be squares as in the deterministic algorithm. Dummy nodes are added to the space-time graph G^{st} so that all the tiles are complete.

Random Shifting The tiling is specified by two additional parameters $\phi_\tau \in [0, (\tau - 1)]$ and $\phi_Q \in [0, (Q - 1)]$, called the *phase shifts*. The phase shifts determine the position of the "first" rectangle; namely, the node (ϕ_τ, ϕ_Q) is the bottom left corner of the first rectangle.

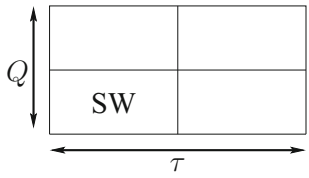


Fig. 10 The south-west (SW) quadrant of a tile

Recall that the sketch graph has a node for every tile in the space-time graph (see Sect. 3.4). Each horizontal edge has a capacity of $Q \cdot B$, and each vertical edge has a capacity of $\tau \cdot c$,

Near and Far Requests A request $r_i = (a_i, b_i, t_i)$ is classified as a *near request* if the tile that contains (a_i, t_i) also contains a copy of b_i (namely, the tile contains a vertex (b_i, t') for some t'). A request that is not a near request is classified as a *far request*. We denote the set of near and far requests by *Near* and *Far*, respectively.

A routing of a request $r_i \in Far$ cannot be confined to a single tile. A routing of a request $r_i \in Near$ may be within a tile or may span more than one tile (our algorithm attempts to route near requests only within a single tile).

SW-Far requests We partition each tile of the untilted space-time graph into four "quadrants" as depicted Fig. 10.

The tiling and random shifting defines the following random subset of the requests. Let $R^+ \subseteq R$ denote the subset of requests whose source vertex is in SW-quadrant of a tile. The subset Far^+ is defined by

$$Far^+ \triangleq R^+ \cap Far.$$

Online Integral Packing of Paths of Far Requests The IPP algorithm is applied only to Far^+ requests over the sketch graph S (see Line 1 in Algorithm 2).

Multiple Simultaneous Requests from The Same Node If multiple requests arrive simultaneously to the same node, then even the optimal routing can serve at most $c + B$ packets among these packets. Since this limitation is imposed on the optimal solution, the path packing algorithm can abide this limitation as well without decreasing its competitiveness. The online algorithm chooses $c + B$ packets whose destination is closest to the source node, as formalized in the following proposition.

Proposition 14 *W.l.o.g. each node injects at most the closest $c + B$ requests at each time step.*

7.3 Randomized Algorithm: Preprocessing

Tiling parameters The tile side lengths are set so that the trivial greedy routing algorithm is $O(\log n)$-competitive for requests classified as near. Each tile has length τ and height Q. Recall that $B, c \leq \log n$.

Definition 15 (i) If $B \cdot c < \log n$, then $\tau = 2\lceil(\log n)/c\rceil$ and $Q = 2 \cdot \lceil(\log n)/B\rceil$. (ii) If $B \cdot c \geq \log n$, then $\tau = 2B$ and $Q = 2c$.

Proposition 16 *The choice of the tiling parameters implies the following:*

1. $\tau + Q = O(\log n)$.
2. *The capacity of each sketch edge is at least* $\log n$.
3. *The ratio of maximum capacity to minimum capacity in the sketch graph is bounded by* 2.

Proof The first part of the proposition follows from the assumption that $B, c \in [1, \log n]$. The capacity $c(e)$ of a horizontal edge e in the sketch graph is $Q \cdot B$. If $Bc \geq \log n$, then $c(e) = 2Bc > \log n$ and all the sketch edges have the same capacity. If $Bc < \log n$, then $c(e) \geq 2\frac{\log n}{B} \cdot B = 2\log n$. Moreover, the ratio of maximum capacity to minimum capacity is bounded by 2. Indeed,

$$\begin{aligned}\frac{Q \cdot B}{\tau \cdot c} &\leq \frac{2 \cdot (1 + \log n/B) \cdot B}{2 \cdot (\log n/c) \cdot c} \\ &= \frac{\log n + B}{\log n} \leq 2\,.\end{aligned}$$

Similarly, the ratio $\frac{\tau c}{QB} \leq 2$, and the proposition follows. □

To simplify the presentation, we assume that $\tau c = QB$ (we can obtain this by reducing the capacities by a factor of at most 2, which affects the competitive ratio only by a factor of 2). Let c^S denote the capacity of the sketch edges to the neighboring tiles.

Proposition 17 *If the phase shifts ϕ_τ and ϕ_Q are chosen independently and uniformly at random, then $E(|\textsc{opt}(R^+)|) = \frac{1}{4} \cdot |\textsc{opt}(R)|$. By a reverse Markov inequality,*

$$\Pr\left[|\textsc{opt}(R^+)| \geq \frac{1}{8} \cdot |\textsc{opt}(R)|\right] \geq \frac{1}{7}.$$

Proof Since the phase shifts ϕ_τ and ϕ_Q are independent and uniformly distributed, the probability that a request $r_i \in R$ is also in R^+ is 1/4. By linearity of expectation, $E(|\textsc{opt}(R^+)|) = \frac{1}{4} \cdot |\textsc{opt}(R)|$.

Plugging $X = |\textsc{opt}(R^+)|$, $d = \frac{1}{8} \cdot |\textsc{opt}(R)|$ and $a = |\textsc{opt}(R)|$ in Lemma 37 (see "Appendix 4") yields the second part of the proposition, i.e., $\Pr\left[|\textsc{opt}(R^+)| \geq \frac{1}{8} \cdot |\textsc{opt}(R)|\right] \geq \frac{1}{7}$. □

7.4 Algorithm for Requests in *Far*$^+$

In this section we present an online algorithm for the requests in the subset *Far*$^+$. Similarly to the deterministic algorithm in Sect. 4, the *Far*$^+$-Algorithm invokes the IPP algorithm (in Step 1) and applies detailed routing (in Step 4). The additional

randomized steps are employed in Step 2, and Step 3. Note that randomized algorithm is non-preemptive, that is, if a packet is not rejected then it is guaranteed to arrive to its destination.

7.4.1 Description of The Far$^+$*-Algorithm*

Parameters Set the maximal path length in the sketch graph to be $p_{\max} \triangleq 4n$. We set the probability λ of the biased coin in Step 2 of ALG_{Far^+} to be $\lambda = 1/(200k)$, where $k = \lceil \log(1 + 3p_{\max}) \rceil$.

Algorithm 2 The *Far*$^+$-Algorithm. The input to the algorithm is a sequence of packet requests in *Far*$^+$ and it either rejects or injects.

Upon arrival of a packet request $r_i = (a_i, b_i, t_i)$ in *Far*$^+$ proceeds as follows (if r_i is rejected in any step, then the algorithm does not continue with the next steps):

1. Reduce the packet requests to an online integral path packing over the sketch graph with paths of length at most $p_{\max}$. Execute the IPP algorithm with respect to these path requests. If the path request is rejected by the IPP algorithm then **reject** r_i. Otherwise, let $\hat{p}_i$ denote the sketch path assigned to request r_i.
2. Toss a biased 0-1 coin X_i such that $\Pr(X_i = 1) = \lambda$. If $X_i = 0$, then **reject** r_i.
3. If the addition of $\hat{p}_i$ causes the load of any sketch edge to be at least $1/4$, then **reject** r_i.
4. Apply I-routing to r_i. If I-routing fails, then **reject** r_i. Otherwise, **inject** r_i with the sketch path $\hat{p}_i$ and alternate between T-routing and X-routing.

The listing of the randomized algorithm appears in Algorithm 2. The input to the algorithm is the sequence of requests in *Far*$^+$ which is processed as follows: (1) The IPP algorithm computes an integral packing of paths over the sketch graph S under the constraint that the length of a path is at most $p_{\max}$. In Proposition 2, we show that this constraint reduces the optimal fractional throughput by a factor of at most two. Algorithm IPP remembers all accepted requests, even those that are rejected in subsequent steps. By Theorem 1, the computed paths constitute an $(O(1), k)$-competitive packing, for $k = O(\log n)$. (2) The probability λ is set to $\frac{1}{\Theta(k)}$. (3) We maintain the invariant that after Line 3, the load of every sketch edge is at most $1/4$. (4) I-routing deals with routing the request out of the initial SW-quadrant and is described in Sect. 7.4.2. The rest of the path is computed based on the sketch path $\hat{p}_i$. This computation is performed locally and on-the-fly by alternating between two routing algorithms called T-routing and X-routing (described in Sect. 7.4.2).

Remark One may consider applying random sparsification before the IPP algorithm is invoked. The motivation for such a variation is to avoid congesting the network with requests destined to be rejected. Apart from reducing the load of sketch edges, random sparsification facilitates successful I-routing (see Lemma 23). This means that sparsification needs to be applied after the online path packing algorithm.

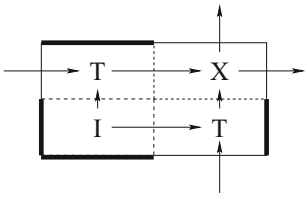


Fig. 11 Allowed detailed routes in tile quadrants. Paths may not cross the *thick lines*

7.4.2 Detailed Routing

The IPP Algorithm computes a sketch path $\hat{p}_i$. If we wish to route the packet, we need to compute a path in G^{st}. We refer to this path as the *detailed path*. Three routing algorithms are employed for computing different parts the detailed path (see Fig. 11): (1) I-routing: from (a_i, t'_i) to the north or east boundaries of the SW-quadrant. (2) T-routing: deals with routing in the north-west quadrant (NW-quadrant) and the south-east quadrant (SE-quadrant) of a tile. (3) X-routing: X-routing deals with routing in the north-east quadrant (NE-quadrant).

Let $\text{ALG}_{Far^+} \subseteq R^+$ denote the subset of requests that were successfully routed by I-routing. Let p_i denote the detailed path of a request $r_i \in \text{ALG}_{Far^+}$. The packing $\{p_j \mid r_j \in \text{ALG}_{Far^+}\}$ satisfies the following invariants:

1. The source of p_j is in the SW-quadrant of a rectangle.
2. The prefix of p_j till it exits the SW-quadrant is straight.
3. For every tile, p_j may enter the tile only through the right half of the south side or the upper half of the west side.
4. For every tile, p_j may exit the tile only through the right half of the north side or the upper half of the east side.
5. Except for the first bend of p_j, every bend corresponds to a bend in the sketch path $\hat{p}_j$.
6. At most $c^S/4$ paths are routed out of the SW-quadrant.
7. The load of every edge in G^{st} is at most one (i.e., all capacity constraints are satisfied).

I-Routing The goal of I-routing is simply to exit the SW-quadrant either from its east side or its north side. I-routing deals with routing paths that start in the SW-quadrant of a tile till the north or east side of the SW-quadrant. I-routing uses only straight paths.

By Proposition 14, at most $B+c$ requests are input at each node of G^{st} to Algorithm IPP. These requests are ordered arbitrarily. We therefore consider each SW-quadrant as a three dimensional cube of dimensions $\frac{Q}{2} \times \frac{\tau}{2} \times (B+c)$ where each node in the quadrant has $B+c$ copies. The ith request that arrives at node (v,t) is input to node (v,t,i) in the cube. We refer to each copy of the quadrant in the cube as a *plane*. Namely, the ith plane is the set of nodes (v,t,i) in the cube. I-routing deals with each $\frac{Q}{2} \times \frac{\tau}{2}$ plane separately,

I-routing tries to route horizontally the first B requests that start at a node. Similarly, I-routing tries to vertically route the requests that arrive after that. By trying to route a request, we mean that if the corresponding row or column in the plane is free, then

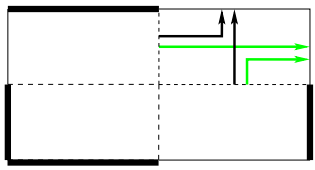


Fig. 12 X-routing is implemented by super-positioning two instances of T-routing depicted by *black* and *grey arrow*

the request is routed (and that row or column in the plane is marked as occupied); otherwise the request is rejected.

Finally, we limit the number of paths that emanate from each side of the SW-quadrant by $c^S/4$, where c^S denotes the capacity of the sketch edges to the neighboring tiles. Thus after $c^S/4$ requests have been successfully I-routed out of the SW-quadrant, all subsequent requests from this SW-quadrants fail.

Note that I-routing is computed before the packet is injected and does not preempt packets (after they are injected) since precedence is given to existing paths.

T-routing The NW-quadrant and the SE-quadrant have a "blocked" side that is depicted by a thick link in Fig. 11. Paths may not traverse the blocked side. T-routing deals with routing in these two quadrants. Paths may enter these quadrants from two sides but must exit through a third side (unless they reach a copy of their destination). We show that T-routing is always successful (in fact, T-routing is similar to detailed routing in internal segments described in Sect. 5.2.3).

Consider a SE-quadrant: each path enters through the south or west sides of the quadrant, and should be routed to the north side of the quadrant. The detailed paths of south-to-north paths are simply vertical paths without bends (such paths are given precedence). The detailed paths of west-to-north paths are obtained by traveling eastward until a bend can be made, namely, the vertical path to the north side is not saturated. Since both path types contain at most $c^S/4$ paths, and since $c^S/2$ paths can cross the north side of the quadrant, T-routing never fails.

X-routing X-routing deals with routing in the NE-quadrant. Note that a path may enter the NE-quadrant from its west side or from its south side. Moreover, a path may exit the NE-quadrant from its east or north side. We show that X-routing is always successful.

X-routing is implemented by super-positioning two instances of T-routing (see Fig. 12). We partition the traffic in a NE-quadrant to two parts based on the side from which the path exits the quadrant. As in T-routing, precedence is given to straight traffic. A bend takes place whenever a free path is available. Clearly, a straight path is successfully routed. Paths that needs to turn are blocked by at most $c^S/4$ paths from the other part. There are at most $c^S/4$ paths that need to turn, and the capacity of the side of the quadrant is $c^S/2$, hence X-routing is always successful. (Note that knock-knee bends are not required, although they could be incorporated).

Last Tile Detailed routing in the last tile employs greedy shortest path routing. If a packet enters the last tile from the south side, then it simply continues north until it reaches its destination. Note that no such packet may enter the last tile from the west side. Indeed, if a sketch path enters s from the west side and s is the last tile in the sketch path, then the neighboring tile from the west contains a copy of the destination, and hence s is not the last tile in the sketch path.

7.4.3 Analysis

Notation We define the following chain subsets of requests

$$\textsc{alg}_{Far^+} \subseteq \textsc{ipp}^{\lambda}_{1/4} \subseteq \textsc{ipp}^{\lambda} \subseteq \textsc{ipp}(Far^+ \mid p_{\max}) \subseteq Far^+ ,$$

as follows. $\textsc{ipp}(Far^+ \mid p_{\max})$ is the subset of requests accepted by the IPP algorithm in Line 1. $\textsc{ipp}^{\lambda} \subseteq \textsc{ipp}(Far^+ \mid p_{\max})$ is the subset of requests for which the biased coin flip X_i equals 1 in Line 2. $\textsc{ipp}^{\lambda}_{1/4} \subseteq \textsc{ipp}^{\lambda}$ is the subset of requests whose addition did not cause a sketch edge to be at least 1/4 loaded in Line 3. $\textsc{alg}_{Far^+} \subseteq \textsc{ipp}^{\lambda}_{1/4}$ is the subset of requests for which detailed routing is successful in Line 4 (recall, that only I-routing may fail).

Let $\textsc{opt}_f(R)$ (respectively, $\textsc{opt}(R)$) denote an optimal fractional (respectively, integral) packing of paths in G^{st} with respect to the requests R. An optimal packing of paths in the space-time graph G^{st} in which the length of the paths in the packing is bounded by $p^{st}_{\max}$ is denoted by $\textsc{opt}_f(R \mid p^{st}_{\max})$.

The following theorem states that the invocation of the IPP algorithm assigns routes to a constant fraction of an optimal solution.

Theorem 18

$$|\textsc{ipp}(\text{Far}^+ \mid p_{\max})| \geq \frac{1}{4} \cdot |\textsc{opt}(\text{Far}^+)|.$$

Proof The proof of the theorem is divided into three parts [summarized by Eqs. (3)–(5)]. The first part states that a fractional packing is not smaller than an integral one.

$$|\textsc{opt}_f(Far^+)| \geq |\textsc{opt}(Far^+)|. \tag{3}$$

The second part shows that bounding the path lengths reduces the throughput only by a factor of 2.

Lemma 19 *([7, Claim 4.5]) Let* $p^{st}_{\max} \triangleq 2 \cdot (n-1) \cdot (1 + B/c)$. *Then,*

$$|\textsc{opt}_f(\text{Far}^+ \mid p^{st}_{\max})| \geq \frac{1}{2} \cdot |\textsc{opt}_f(\text{Far}^+)|. \tag{4}$$

The third part shows that paths of length at most $p^{st}_{\max}$ in the space-time graph are mapped to paths of length at most $4n$ in the sketch graph.

Proposition 20 *Every path p in G^{st} of length at most $p^{st}_{\max}$ is mapped to a path $\hat{p}$ in the sketch graph S of length at most $4n$. Hence, by the $(2, k)$-competitiveness of the* IPP *Algorithm, it follows that:*

$$|\textsc{ipp}(\text{Far}^+ \mid p_{\max})| \geq \frac{1}{2} \cdot |\textsc{opt}_f(\text{Far}^+ \mid p^{st}_{\max})|. \tag{5}$$

Proof Let p denote a path of length at most $p^{st}_{\max} \triangleq 2 \cdot (n-1) \cdot (1 + B/c)$ in G^{st}. We partition the edges of $\hat{p}$ into horizontal edges and vertical edges in $\hat{p}$. The number of vertical edges in p is bounded is n and the same holds also for $\hat{p}$.

We now prove that the number of horizontal edges in $\hat{p}$ is at most $3n$. For every row i in G^{st}, let n_i denote the number of horizontal edges of p in the ith row. Similarly, for every row i in the sketch graph, let $\hat{n}_i$ denote the length of the intersection of $\hat{p}$ with the ith row of the sketch graph. Let $[\alpha_i, \beta_i]$ denote the interval of rows of G^{st} that are mapped to the ith row of the sketch graph (note that $\beta_i - \alpha_i$ is simply the height of a tile).

By Def. 15, the length of every tile is at least $2B$. Indeed, if $B \cdot c > \log n$, then the length τ equals $2B$. If $B \cdot c \leq \log n$, then the length $\tau \geq 2 \log n / c \geq 2B$. It follows that

$$\hat{n}_i \leq \left\lceil \frac{\sum_{j=\alpha_i}^{\beta_i} n_j}{2B} \right\rceil \leq \frac{1}{2B} \cdot \sum_{j=\alpha_i}^{\beta_i} n_j + 1.$$

Hence, $\sum_i \hat{n}_i \leq \frac{p^{st}_{\max}}{2B} + n \leq 3n$. We conclude that the length of the path $\hat{p}$ is at most $4n$, as required. □

Equations (3)–(5) completes the proof of Theorem 18 □

The following proposition shows that, in expectation over the biased coins tosses in Line 2, at most a quarter of the sketch paths are rejected due to "$\frac{1}{4}$-loaded" edges in Line 3 of the Far^{+}-Algorithm.

Lemma 21 *If $n > 16$, then*

$$E(|\text{IPP}^{\lambda}_{1/4}|) \geq \frac{3}{4} \cdot E(|\text{IPP}^{\lambda}|) .$$

Proof The idea it to show that, after random sparsification, the load of every sketch edge is at most $1/4$ with high probability. This implies that few requests are rejected as a result of causing the load of an edge to be greater than $1/4$.

Let $\hat{p}_i$ denote the sketch path of r_i. Given a sketch edge e, let $P(e) \triangleq \{\hat{p}_i : r_i \in \text{IPP}(Far^{+} \mid p_{\max}), e \in \hat{p}_i\}$ denote the set of sketch paths that traverse e. Similarly, let $P^{\lambda}(e) \triangleq \{\hat{p}_i : r_i \in \text{IPP}^{\lambda}, e \in \hat{p}_i\}$ denote the set of paths that traverse e after random sparsification. We first claim that, for a constant $\gamma > 200$, for $n > 24$, and for every sketch edge e,

$$\Pr\left(|P^{\lambda}(e)| > \frac{c(e)}{4}\right) < \frac{1}{16n}. \tag{6}$$

We now prove Eq. (6). Since $\text{IPP}(Far^{+} \mid p_{\max})$ is $(2, k)$-competitive, it follows that

$$|P(e)| \leq k \cdot c(e) .$$

The tossing of the biased coins with $\lambda = 1/(\gamma k)$ with $\gamma = 200$, implies that

$$E(|P^{\lambda}(e)|) = \lambda \cdot |P(e)| \leq \lambda k \cdot c(e) = \frac{1}{\gamma} \cdot c(e).$$

The following sequence of equations is explained below.

$$\begin{aligned}
\Pr\left(|P^{\lambda}(e)| > \frac{c(e)}{4}\right) &= \Pr\left(|P^{\lambda}(e)| \geq (1+\delta)\frac{c(e)}{\gamma}\right) \\
&< \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\frac{c(e)}{\gamma}} \\
&\leq \left(\frac{e^{\delta/\gamma}}{(1+\delta)^{(1+\delta)/\gamma}}\right)^{2\cdot\log n} \\
&= \left(\frac{e^{\frac{1}{4}-\frac{1}{\gamma}}}{(\frac{\gamma}{4})^{\frac{1}{4}}}\right)^{2\cdot\log n},
\end{aligned}$$

The first line holds if δ satisfies $\frac{1+\delta}{\gamma} = \frac{1}{4}$. The second line is due to a multiplicative Chernoff bound [21]. The third line is implied by Proposition 16 since $c(e) \geq 2 \cdot \log n$. The last line follows by the definition of δ.

Since $\gamma = 200$ and $n > 16$, then $\left(\frac{e^{\frac{1}{4}-\frac{1}{\gamma}}}{(\frac{\gamma}{4})^{\frac{1}{4}}}\right)^2 < 2^{-2} < 2^{-\frac{\log 16n}{\log n}}$ and therefore, $\Pr\left(|P^{\lambda}(e)| > \frac{c(e)}{4}\right) < 2^{-\log 16n}$ and Eq. (6) holds.

Since $p_{\max} = 4n$, the length of each sketch path is at most $4n$. By Eq. (6) and by applying a union bound it follows that

$$\begin{aligned}
\Pr\left(r_i \notin \textsc{ipp}^{\lambda}_{1/4} \mid r_i \in \textsc{ipp}^{\lambda}\right) &\leq \Pr\left(\exists\, e \in \hat{p}_i : P^{\lambda}(e) > \frac{1}{4} \cdot c(e)\right) \\
&\leq 4n \cdot \frac{1}{16n} = \frac{1}{4}.
\end{aligned}$$

The lemma follows by linearity of expectation. □

The following theorem states that, in expectation, a $1/\Theta(k)$ fraction of the requests that are accepted by the IPP algorithm are successfully routed.

Theorem 22 $E(|\textsc{alg}_{\text{Far}^+}|) \geq \frac{\lambda}{4} \cdot |\textsc{ipp}(\text{Far}^+ \mid p_{\max})|$.

Proof We first prove a Lemma and a Proposition. Lemma 23 deals with a projection of a random sparsification of a 0-1 matrix. This lemma helps estimate the number of requests from $\textsc{ipp}^{\lambda}$ for which I-routing is successful in each plane (ignoring the effect of Line 3 in the algorithm). Proposition 24 helps analyze the effect of Line 3 on the number of requests for which I-routing is successful. □

Definitions Let $I(\cdot)$ be an operator over 0-1 matrices defined as follows. $I(X)$ is all zeros except for the first nonzero entry in each row of X. Namely,

$$I(X)_{i,j} \triangleq \begin{cases} 1 & \text{if } X_{i,j} = 1 \text{ and } X_{i,\ell} = 0, \text{ for every } \ell < j \\ 0 & \text{otherwise.} \end{cases}$$

The motivation for this definition is as follows. Suppose that the matrix X indicates the existence of packets in a plane of a SW-quadrant in which packets are routed by I-routing along rows out of the quadrant. The only packets for which I-routing succeeds in this plane are the packets that correspond to ones in $I(X)$.

Let $L \wedge B$ denote the matrix obtained by the coordinate-wise conjunction of L and B. For a matrix X, let $w(X)$ denote the number of 1's in X.

In the following lemma we analyze the effect of random sparsification on I-routing along the rows of the SW-quadrant. A similar effect occurs when considering I-routing along the columns of the SW-quadrant.

Lemma 23 *Let A and Z be* 0-1 *matrices whose dimensions are $\frac{Q}{2} \times \frac{\tau}{2}$. Assume that the entries of Z are i.i.d.* 0-1 *random variables with $E(z_{ij}) = \lambda$. Let $\lambda < \frac{2}{\tau}$. Then,*

$$E\Big(w\big(I(A \wedge Z)\big)\Big) \geq \frac{\lambda}{2} \cdot w(A) .$$

Proof Consider each row A_i of A and Z_i of Z separately. The expectation of the 0-1 random variable $w(I(A_i \wedge Z_i))$ equals the probability that it equals 1. Note that

$$\begin{aligned} \Pr(w(A_i \wedge Z_i) = 0) &= (1-\lambda)^{w(A_i)} \\ &\leq e^{-\lambda \cdot w(A_i)}. \end{aligned}$$

Since $\lambda \cdot \tau/2 \leq 1$, it follows that $\lambda \cdot w(A_i) \leq 1$, and hence

$$\begin{aligned} \Pr(w(A_i \wedge Z_i) = 1) &\geq 1 - e^{-\lambda \cdot w(A_i)} \\ &\geq \frac{\lambda}{2} \cdot w(A_i). \end{aligned}$$

The lemma follows by linearity of expectation. □

We now return to the proof of Theorem 22. For every tile consider its SW-quadrant as a three dimensional cube of dimensions $\frac{\tau}{2} \times \frac{Q}{2} \times (B + c)$. Recall that I-routing deals with each $\frac{\tau}{2} \times \frac{Q}{2}$ plane separately.

The lengths τ and Q of each tile are at most $2 \log n$. Recall that $\lambda = \frac{1}{\gamma \cdot k}$ where $k \geq \log(1 + 3 \cdot 4n)$. Hence, if $\gamma = 200$, then $1/\lambda = \gamma \cdot k \geq \frac{\tau}{2}$.

Assume that we skip Step 3 of the algorithm (namely, we do not check that the load is bounded by 1/4), and apply directly I-routing to the requests in IPP^{λ}. Let $I_{\text{IPP}^{\lambda}}$

denote the set $\{r_i \in \text{IPP}^\lambda : I\text{—routing succeeds in routing } r_i\}$. We consider each of the $(B + c)$ planes separately, and by Lemma 23 and linearity of expectation, we obtain

$$\begin{aligned} E(|I_{\text{IPP}^\lambda}|) &\geq \frac{\lambda}{2} \cdot |\text{IPP}(Far^+ \mid p_{\max})| \\ &= \frac{1}{2} \cdot E(|\text{IPP}^\lambda|) \,. \end{aligned} \tag{7}$$

Furthermore, Lemma 21 implies that:

$$E(|\text{IPP}^\lambda \setminus \text{IPP}^\lambda_{1/4}|) \leq \frac{1}{4} \cdot E(|\text{IPP}^\lambda|) \,. \tag{8}$$

Hence,

$$E(|I_{\text{IPP}^\lambda}|) - E(|\text{IPP}^\lambda \setminus \text{IPP}^\lambda_{1/4}|) \geq \frac{1}{4} \cdot E(|\text{IPP}^\lambda|) \,. \tag{9}$$

Notations For a 0-1 matrix L, let $\bar{L}$ denote negated matrix $\bar{L}_{i,j} \triangleq 1 - L_{i,j}$. For matrices L and B, let $L \leq B$ denote $L_{i,j} \leq B_{i,j}$, for every i and j.

Proposition 24 *If $L \leq B$ then:*

$$w(I(L)) \geq w(I(B)) - w(B \wedge \bar{L}).$$

Proof It suffices to deal with each row separately. Let B_i denote the ith row of the matrix B. We claim that if $w(I(B_i)) = 1$, then $w(I(L_i)) = 1$ or $w(B_i \wedge \bar{L}_i) \geq 1$. Indeed, assume that $w(I(B_i)) = 1$ and $w(I(L_i)) = 0$. Then, L_i is all zeros. Hence, $B_i \wedge \bar{L}_i = B_i$, and the proposition follows. □

We now prove the following lemma.

Lemma 25 *For every outcome of the random biased coins:*

$$|\text{ALG}_{\text{Far}^+}| \geq |I_{\text{IPP}^\lambda}| - |\text{IPP}^\lambda \setminus \text{IPP}^\lambda_{1/4}| \,.$$

Proof Consider a specific tile s and its SW-quadrant. Fix an i-plane used by I-routing. W.l.o.g this i-plane corresponds to a horizontal I-routing. Define three 0-1 matrices A, Z and L with dimensions $\frac{Q}{2} \times \frac{\tau}{2}$, as follows:

1. Let A be the matrix whose entries indicate the existence of a request $r \in \text{IPP}(Far^+ \mid p_{\max})$ whose source vertex is in the ith plane of the SW-quadrant of the tile s. Namely, $A_{v,t} = 1$ iff node (v, t) receives at least i requests in $\text{IPP}(Far^+ \mid p_{\max})$.
2. Let Z denote a random matrix in which the entries are i.i.d. Bernoulli random variables with $Pr(Z_{v,t} = 1) = \lambda$. These Bernoulli random variables correspond to the outcomes of the biased coin tosses in Step 2 of the algorithm.
3. Let L be the matrix whose entries indicate the existence of a request $r \in \text{IPP}^\lambda_{1/4}$ whose source vertex is in the ith plane of the SW-quadrant of the tile s.

For a subset W of requests, a tile s, and a plane index i, let $W(s,i) \subseteq W$ denote the subset of requests in W whose source vertex is in the ith plane of the tile s. Let $\bar{L}$ denote the negation of L. By definition the following identities hold:

(i) $|\text{ALG}_{Far^+}(s,i)| = w(I(L))$,
(ii) $|\text{IPP}^{\lambda}(s,i)| = w(A \wedge Z)$,
(iii) $|I_{\text{IPP}^{\lambda}}(s,i)| = w(I(A \wedge Z))$,
(iv) $|\text{IPP}^{\lambda}(s,i) \setminus \text{IPP}^{\lambda}_{1/4}| = w(A \wedge Z \wedge \bar{L})$.

It suffices to prove that

$$w(I(L)) \geq w(I(A \wedge Z)) - w(A \wedge Z \wedge \bar{L}). \tag{10}$$

Since $L \leq (A \wedge Z)$, Eq. (10) follows from Proposition 24, and the lemma follows. □

We now complete the proof of Theorem 22. By Lemma 25 and Eq. (9), it follows that $E(|\text{ALG}_{Far^+}|) \geq \frac{1}{4} \cdot E(|\text{IPP}^{\lambda}|)$. Theorem 22 follows since $E(|\text{IPP}^{\lambda}|) = \lambda \cdot |\text{IPP}(Far^+ \mid p_{\max})|$. □

Theorem 26 $E(|\text{ALG}_{\text{Far}^+}|) \geq \Omega(\frac{1}{\log n}) \cdot |\text{OPT}(\text{Far}^+)|$.

Proof By Theorem 22, it follows that $E(|\text{ALG}_{Far^+}|) \geq \Omega(\lambda) \cdot |\text{IPP}(Far^+ \mid p_{\max})|$. By Theorem 18, $|\text{IPP}(Far^+ \mid p_{\max})| \geq \Omega(|\text{OPT}(Far^+)|)$. The theorem follows since $\lambda = 1/\Theta(k) = 1/\Theta(\log n)$. □

7.5 Algorithm for Requests in *Near*

In this section we present an online algorithm for the requests in the subset *Near*. The algorithm is a straightforward greedy vertical routing algorithm. Given a request $r_i \in Near$, the algorithm attempts to routs the request vertically.

We emphasize that an optimal routing is not restricted to routing a request $r_i \in Near$ within the tile.

Notations Let ALG_{Near} denote the set of requests successfully routed by the *Near*-Algorithm with respect to the requests in *Near*. Let $\text{ALG}_{Near}(s)$ denote the set of requests routed by the *Near*-Algorithm within the tile s. Let $Near_s$ denote the set of requests in *Near* whose starting node is in the tile s. We abuse notation and refer to the set of routed packets in an optimal routing with respect to $Near_s$ also by $|\text{OPT}(Near_s)|$.

Theorem 27 *For every tile s,* $|\text{ALG}_{\text{Near}}(s)| \geq \Omega(\frac{1}{\log n}) \cdot |\text{OPT}(\text{Near}_s)|$.

Proof It suffices to prove that

$$|\text{ALG}_{Near}(s)| > \Omega\left(\frac{1}{\log n}\right) \cdot |\text{OPT}(Near_s) \setminus \text{ALG}_{Near}(s)|$$

We consider a bipartite conflict graph between requests in $\text{ALG}_{Near}(s)$ and $\text{OPT}(Near_s) \setminus \text{ALG}_{Near}(s)$. There is an edge $(r, r') \in \text{ALG}_{Near}(s) \times \text{OPT}(Near_s) \setminus \text{ALG}_{Near}(s)$ if the vertical path of r shares an edge with the path of r' in $\text{OPT}(Near_s) \setminus \text{ALG}_{Near}(s)$.

Since at most c requests can traverse the same vertical edge, it follows that a route of a request in $\text{ALG}_{Near}(s)$ conflicts with at most

$$deg(r) \leq Q \cdot c\,.$$

If $r' \notin \text{ALG}_{Near}(s)$, then it either encountered a saturated horizontal edge or a saturated vertical edge. Hence, the degree of $r' \in \text{OPT}(Near_s) \setminus \text{ALG}_{Near}(s)$ is at least

$$deg(r') \geq c\,.$$

By counting edges on each side we conclude that

$$\begin{aligned}\frac{|\text{OPT}(Near_s) \setminus \text{ALG}_{Near}(s)|}{|\text{ALG}_{Near}(s)|} &\leq \frac{\max deg(r)}{\min deg(r')} \\ &\leq \frac{Q \cdot c}{c}.\end{aligned}$$

By Definition 15, $Q \leq 2 \cdot \log n$, and the theorem follows. □

Corollary 28 $|\text{ALG}_{\text{Near}}| \geq \Omega\left(\frac{1}{\log n}\right) \cdot |\text{OPT}(\text{Near})|$.

7.6 Putting Things Together

The online randomized algorithm ALG for packet routing on a directed line proceeds as follows.

1. Choose the tiling parameters τ, Q according to Definition 15.
2. Choose the phase shifts $\phi_\tau \in [0, \tau - 1]$, $\phi_Q \in [0, Q - 1]$ of tiling independently and uniformly at random.
3. Flip a random fair coin $b \in \{0, 1\}$.
4. If $b = 1$, then consider only requests in Far^+, and apply the Far^+-algorithm to these requests.
5. If $b = 0$, then consider only requests in *Near*, and apply the *Near*-algorithm to these requests.

Theorem 29 *If $B, c \in [1, \log n]$, then the competitive ratio of* ALG *is* $O(\log n)$.

Proof sketch of Theorem 29 The chosen tiling parameters and phase shifts induce a classification of the requests to two classes: *Near* and *Far+*. With probability $\frac{1}{2}$ the random fair coin b chooses the bigger class. Theorem 26 and Corollary 28 state that ALG_{Far^+} and ALG_{Near} are $O(\log n)$ competitive, and the theorem follows.

7.7 Large Buffers

In this section we consider a special setting in which the buffers are large. Note that the Algorithm fails if $B = \omega(\log n)$ both with near and far requests. Formally, assume that $\log n \leq B/c \leq n^{O(1)}$.

We briefly mention the required modifications. The tiling parameters are $\tau = B/c$ and $Q = 1$. This implies that there are no near requests and all requests are classified as far. Each tiles is partitioned in to a left half and a right half. The algorithm considers only requests whose source vertex is in the left half of a tile; such requests are denoted by R^+. Note that random shifting is employed so that on the average R^+ contains half the requests.

The north and south side of the left half of each tile are "blocked" so that detailed routing does not traverse these sides. This means that I-routing is only along horizontal edges. In the right half of each tile, three T-routing are super imposed. The first T-routing is for the paths that enter the tile from the west side. These paths traverse the left half horizontally and then in the right half undergo T-routing (so that they exit from the east or north side of the right half). The second T-routing is for the paths that enter the tile from the south side of the right half. Finally, the third T-routing is for continuing the paths of the I-routing from the border between the halves to the north and east sides of the right half of the tile.

Path lengths are bounded as before (this is why we require that B/c is polynomial). In addition the random sparsification parameter λ is the same.

The algorithm proceeds as follows:

1. Execute the IPP algorithm with respect to the path requests in R^+ over the sketch graph.
2. Toss a biased 0-1 coin X_i such that $\Pr(X_i = 1) = \lambda$. If $X_i = 0$, then **reject** r_i.
3. If the addition of $\hat{p}_i$ causes the load of any sketch edge to be at least $1/4$, then **reject** r_i.
4. Apply I-routing to r_i. If I-routing fails, then **reject** r_i. Otherwise, **inject** r_i with the sketch path $\hat{p}_i$ and apply T-routing till the destination is reached.

In this setting, the ratio between the capacity of the sketch edges that emanate from a tile to the number of requests whose source vertex is in the tile is constant. This constant ratio simplifies the proof of the following theorem compared to the proof of Theorem 29.

Theorem 30 *If* $\log n \leq B/c \leq n^{O(1)}$, *then there exists a randomized online algorithm that achieves a logarithmic competitive ratio for packet routing in a uni-directional line.*

Recall that for the case where $B, c \in [\Omega(\log n), \infty)$ and $B/c = n^{O(1)}$, there is an even simpler and *deterministic* online algorithm with $O(\log n)$ competitive ratio, as stated in Theorem 13.

7.8 Small Buffers and Large Link Capacities

The case $B \in [1, \log n]$ and $c \in [\log n, \infty)$ is dealt with by simplifying the algorithm. We briefly mention the required modifications. The tile size is $\tau = 1$ and $Q = \log n/B$. The maximum path length is set to $2(n-1)(1+B/c)$ which is polynomial (i.e., tiling is not needed to reduce the path length). Instead of partitioning a tile into quadrants, we partition each tile into an upper half and a lower half. The set R^+ is defined to the set of requests whose origin is in the lower half of a tile.

The set *Near* is dealt by a vertical path. Since in every tile s, $|\text{ALG}_{Near}(s)| \geq \min\{c, |\text{OPT}(Near_s)|\}$ and since $|\text{OPT}(Near_s)| \leq \frac{\log n}{B} \cdot (B + c)$, it follows that $\frac{|\text{ALG}_{Near}(s)|}{|\text{OPT}(Near_s)|} \geq \frac{1}{\log n}$.

The set Far^+ is dealt by invoking a variation of the Far^+-Algorithm. The modified invariants for detailed routing are that paths may not enter or exit horizontally through the lower half of a tile (but, of course, may traverse the tile vertically). I-routing simply routes the first $\frac{3}{4} \cdot c$ requests vertically. The remaining capacity of $\frac{c}{4}$ is reserved for incoming paths from the south side. In the upper half of each tile, X-routing on a single column is employed.

We conclude with the following theorem.

Theorem 31 *If $B \in [1, \log n]$ and $c \in [\log n, \infty)$, then there exists a randomized online algorithm that achieves a logarithmic competitive ratio for packet routing in a uni-directional line.*

Remark The space-time graph seems to assign symmetric roles to the time axis and the space axis. Such a symmetry would imply that one could reduce the case of large buffers to the case of large link capacities. However, this is not true due to the definition of a destination. A destination (in the space-time graph) is a row of vertices (namely, the set of copies of an original vertex). This implies that one cannot simply transpose the graph and exchange the roles of space and time.

8 Open Problems

Two basic problems related to the design and analysis of online packet routing remain open even for uni-directional lines. (i) Achieve a constant competitive ratio or prove a lower bound that rules out a constant competitive ratio. (ii) Achieve a logarithmic competitive ratio by a distributed algorithm (as opposed to a centralized algorithm).

In a follow-up paper [15] we showed an improved $O(\log n)$-competitive deterministic online algorithm for uni-directional lines. We also extended this algorithm to the d-dimensional case, in which its competitive ratio is $O(\log^d n)$-competitive. In this context, it remains to see wether competitiveness for the d-dimensional case can be improved.

Acknowledgments Open access funding provided by Max Planck Society. We thank Niv Buchbinder, Boaz Patt-Shamir and Adi Rosén for useful discussions.

Appendix 1: Proof of Lemma 2

Lemma 2 *Let* $p_{\max} \geq 2n \cdot (1 + \frac{B}{c})$. *Then,*

$$|\textsc{opt}_f(R \mid p_{\max})| \geq \frac{1}{2} \cdot \left(1 - \frac{1}{e}\right) \cdot |\textsc{opt}_f(R)| \,.$$

Proof Let f^* denote an optimal fractional path packing in G^{st} with respect to a set of flow requests R, that is $f^* = \textsc{opt}_f(R)$. The flow $f^*/2$ has a throughput that is half the throughput of f^* and the load of each edge is at most $\frac{1}{2}$.

We construct two flows g and h, each of which incurs a load of at most $\frac{1}{2}$ on each edge and $|g + h| \geq (1 - e^{-1}) \cdot |f^*|/2$. Let $0 < \alpha < 1$ and ℓ denote parameters to be defined later. The flows g and h are constructed for each flow path p of f^* as follows. Initially, h is zero and g starts in the first vertex of p with half the flow amount of f^* along p. During the first ℓ edges of p, the flows g and h are defined as follows. In each "forward" edge, the incoming flow of g is forwarded to the outgoing edge along p, and no flow is added to h. In each "store" edge of p, we decimate g by leaving only a fraction $(1 - \alpha)$ of g along the store edge. The remaining α fraction is deflected to h which routes the flow along a shortest path in G to the destination (i.e., uses only forward edges). After ℓ edges, we drop whatever flow is left of g.

Clearly, the load that g incurs on each edge is at most $\frac{1}{2}$. To obtain the same bound for h, we need to set α to be small enough. Indeed, the flow along each forward edge in h is bounded by $\alpha \cdot nB$. Thus it suffices to have

$$\alpha \cdot nB \leq \frac{c}{2}.$$

So we set $\alpha \triangleq \frac{c}{2nB}$.

The fraction of dropped flow is bounded by $(1-\alpha)^{\ell'}$, where ℓ' denotes the number of store edges along the first ℓ edges of p. Let $\ell \triangleq n + \alpha^{-1}$, then $\ell' \geq \alpha^{-1}$; otherwise, we would have reached the end of p as it contains less than n edges. Hence the fraction of flow in f^* that is dropped by $g + h$ is at most $(1 - \alpha)^{1/\alpha} \leq 1/e$.

To conclude, the length of the flow paths in $g+h$ is bounded by $\ell+n = 2n(1+\frac{B}{c})$, and the flow amount of $g + h$ is at least $\frac{1}{2} \cdot (1 - \frac{1}{e})$ times the flow amount of f^*. □

Appendix 2: Proof sketch of Theorem 10

The proof of Theorem 10 follows the proof of Theorem 4. The proof of the propositions below follows the analogous proofs in Sect. 5.3.

Proposition 32 $|f^*(R \mid p_{\max})| \geq |\textsc{opt}_f(R \mid p_{\max})|$.

Proposition 33

$$\frac{1}{d+1} \cdot k^{d+1} \cdot (B + d \cdot c) \cdot |f^*_{\{1,d+1,\infty\}}(R \mid p_{\max})| \geq |f^*(R \mid p_{\max})| \geq |f^*_{\{1,d+1,\infty\}}(R \mid p_{\max})|$$

Proposition 34 $|\text{IPP}| \geq \Omega\left(\frac{d+1}{k^{d+1}\cdot(B+d\cdot c)}\right) \cdot |f^*(R \mid p_{\max})|$

Proposition 35 $|\text{IPP}'| \geq \frac{1}{2k} \cdot |\text{IPP}|$

Proposition 36 $|\text{ALG}| \geq \frac{1}{(d+1)\cdot k} \cdot |\text{IPP}'|$

Theorem 10. *The competitive ratio of the algorithm for d-dimensional grid networks is*

$$O\left(k^{d+3} \cdot (B + d \cdot c)\right) = O\left(\log^{d+4} n\right)$$

provided that $B, c \in [3, \log n]$.

Proof sketch of Theorem 10 Bounding path lengths incurs a constant loss to the competitive ratio. Algorithm IPP incurs an additional constant loss to the competitive ratio. The capacity assignment of $\{1, d+1\}$ reduces the throughput by a factor of $\frac{1}{d+1} \cdot k^{d+1} \cdot (B + d \cdot c)$. Similarly to the uni-dimensional case, a fraction of at most $(1 - \frac{1}{2k})$ of the requests in IPP are preempted before they reach their last cube. Finally, a fraction of at least $\frac{1}{(d+1)\cdot k}$ of the requests that reach their last tile are successfully routed, i.e., by detailed routing in the last tile. Hence, the total fraction of requests that are successful routed is $\Omega\left(\frac{1}{k^{d+3}\cdot(B+d\cdot c)}\right)$. The theorem follows since $B, c \in [3, \log n]$. □

Appendix 3: Proof of Theorem 11

Theorem 11. *There exists an online deterministic preemptive algorithm for packet routing in bufferless d-dimensional grids with a competitive ratio of* $O(\log^{d+2} n)$.

Proof Since $B = 0$, the space-time graph G^{st} after untilting consists of unconnected d-dimensional grids. Within each such d-dimensional grid, we apply a version of our algorithm. Note that since $B = 0$, trivially $p_{\max} \leq \sum_i \ell_i$ (i.e., the diameter of the grid) and does not depend on c. Note also that the destination is a single node (b_i, t'), where $t' = t_i + \|a_i - b_i\|_1$. Thus we need not introduce sink nodes. The edge capacities are $d \cdot c$ to every interior edge (instead of $(d+1)$). Hence, the capacity assignment reduces the throughput by a factor of k^d (instead of $k^{d+1} \cdot (B + d \cdot c)$). □

Appendix 4: Proof of Lemma 37

Lemma 37 *(A Reverse Markov Inequality) Let X be a nonnegative bounded random variable attaining values in* $[0, a]$. *For every* $d < a$,

$$\Pr(X \geq d) \geq \frac{E(X) - d}{a - d}.$$

Proof We prove that $\Pr(X < d) \leq 1 - \frac{E(X)-d}{a-d}$. Let Y be a random variable such that $Y \triangleq a - X$. Note that, Y is also a nonnegative bounded random variable attaining values in $[0, a]$. Hence, $X < d$ if and only if $Y > a - d$. The expected value of Y is $E(Y) = a - E(x)$. The lemma follows by applying Markov Inequality [21], as follows:

$$\begin{aligned}\Pr(X < d) &= \Pr(Y > a - d) \\ &\leq \frac{E(Y)}{a-d} \\ &= \frac{a - E(x)}{a-d} \\ &= 1 - \frac{E(x) - d}{a-d}.\end{aligned}$$

□

Appendix 5: Online Integral Path Packing Algorithm IPP

In this section we present algorithm IPP and prove Theorem 1. The presentation follows the framework of [10,11]. The presentation emphasizes two points: (1) The graph over which the requests arrive may be infinite. (2) There is an upper bound $p_{\max}$ on the length of a path that may serve a request.

Linear Programming Formulation Fractional path packing is a multi-commodity flow problem, and is formulated by a linear program (LP). In Fig. 13, the dual LP

$$\min \sum_{e \in E} x_e \cdot c(e) + \sum_i z_i \quad s.t.$$
$$\forall i \ \forall p \in P_i \ : \ \sum_{e \in p} x_e + z_i \geq 1 \quad \text{(covering const.)}$$
$$x, z \geq 0$$

(I)

$$\max \sum_i \sum_{p \in P_i} f(i,p) \quad s.t.$$
$$\text{(demand const.)} \quad \forall i \quad \sum_{p \in P_i} f(i,p) \leq 1$$
$$\text{(capacity const.)} \quad \forall e \in E \quad \mathit{flow}(e) \leq c(e)$$
$$f \geq 0$$

(II)

Fig. 13 (I) The Primal linear program. (II) The Dual linear program

corresponds to the fractional path packing problem as well as the corresponding primal LP are listed.

The notation in the LPs is as follows. For each request i, let P_i denote the set of paths in G that can serve the request $r_i = (a_i, b_i)$. The length of every path $p \in P_i$ is at most $p_{\max}$. The variables $f(i, p)$ denote the amount of flow allocated to request i along the path p. The demand constraint in the dual LP states that at most one unit of flow can be jointly allocated to all the paths in P_i. The capacity constraint states that at most $c(e)$ units of flow can traverse an edge e. The objective is to maximize the flow amount.

The primal LP has two types of variables: one variable z_i per request r_i and one variable x_e per edge e. The variable x_e can be interpreted as a weight assigned to the edge e. The covering constraint states that for every request r_i and every path $p \in P_i$, the weight of the path p plus z_i should be at least 1. The objective is to minimize the sum of edge weights times their capacities plus the sum of the variables z_i.

The Online Algorithm for Integral Packing of Paths The listing of algorithm IPP appears in Fig. 3. Note that the graph $G = (V, E)$ may be infinite. This implies that the primal LP has an infinite number of variables (however, all but a finite subset of the primal LP variables are zero). We assume that there exists a lightest path oracle that, given edge weights x_e and a request r_i, finds a lightest path $p \in P_i$.

Algorithm 3 The IPP algorithm. We assume that all the variables are initialized to zero using lazy initialization. We assume that given edge variables x_e, there exist an oracle that returns a lightest path in P_i.

Input: $G = (V, E)$ (possibly infinite), sequence of requests $\{r_i\}_{i=1}^{\infty}$ where $r_i \triangleq (a_i, b_i)$.
Upon arrival of request r_i:

1. Let $\alpha(p, i) \triangleq \sum_{e \in p} x_e$.
2. $p \leftarrow \text{argmin}\{\alpha(p', i) : p' \in P_i\}$ (find a lightest path from a_i to b_i using an oracle).
3. If $\alpha(p, i) < 1$ then, **route** r_i along p:
   (a) $f(i, p) \leftarrow 1$.
   (b) For each $e \in p$ do

$$x_e \leftarrow x_e \cdot 2^{1/c(e)} + \frac{1}{p_{\max}} \cdot (2^{1/c(e)} - 1) .$$

   (c) $z_i \leftarrow 1 - \alpha(p, i)$.
4. Else, **reject** r_i.
   (a) $z_i \leftarrow 0$.

For a given sequence σ of requests let $F^*(\sigma)$ denote the maximum flow of the dual LP. An online integral path packing algorithm is said to be (α, β)-*competitive* if for every sequence σ of requests (1) its total throughput is at least $F^*(\sigma)/\alpha$, and (2) the load of every edge is at most β.
The proof of the following theorem follows the framework of [11,12].

Theorem 1. *Algorithm* IPP *is a* $(2, \log(1 + 3 \cdot p_{\max}))$*-competitive online integral path packing algorithm under the following assumptions: (1)* $\min_e c(e) \geq 1$. *(2) A path is*

legal if it contains at most $p_{\max}$ edges. (3) There is an oracle, that given edge weights and a request, finds a lightest legal path from the source to the destination.

Proof Let us denote by $\Delta_i P$ (respectively, $\Delta_i D$) the change in the primal (respectively, dual) cost function after request r_i is processed. We claim that $\Delta_i P \leq 2 \cdot \Delta_i D$.

If r_i is rejected, then $\Delta_i P = \Delta_i D = 0$. If r_i is accepted, then $\Delta_i D = 1$ and $\Delta_i P = \sum_{e \in p} \Delta_i x_e \cdot c(e) + \Delta_i z_i$. Step (3b) increases the cost $\sum_e x_e \cdot c(e)$ as follows:

$$\begin{aligned}
\sum_e \Delta_i x_e \cdot c(e) &= \sum_{e \in p} \left[x_e \cdot (2^{1/c(e)} - 1) + \frac{1}{p_{\max}} \cdot (2^{1/c(e)} - 1) \right] \cdot c(e) \\
&= \sum_{e \in p} \left(x_e + \frac{1}{p_{\max}} \right) \cdot (2^{1/c(e)} - 1) \cdot c(e) \\
&\leq c_{\min} \cdot (2^{1/c_{\min}} - 1) \sum_{e \in p} \left(x_e + \frac{1}{p_{\max}} \right) \\
&\leq 1 \cdot (2^1 - 1) \sum_{e \in p} \left(x_e + \frac{1}{p_{\max}} \right) \\
&\leq \sum_{e \in p} x_e + \sum_{e \in p} \frac{1}{p_{\max}} \\
&\leq \alpha(p, i) + 1 \,. \qquad (11)
\end{aligned}$$

Hence after step (3c):

$$\begin{aligned}
\Delta_i P &= \sum_{e \in p} \Delta_i x_e \cdot c(e) + \Delta_i z_i \\
&\leq (\alpha(p, i) + 1) + (1 - \alpha(p, i)) \\
&= 2 \,. \qquad (12)
\end{aligned}$$

Since $\Delta_i D = 1$ it follows that $\Delta_i P \leq 2 \cdot \Delta_i D$, as required.

After dealing with each request, the primal variables $\{x_e\}_e \cup \{z_i\}_i$ constitute a feasible primal solution. Given a dual solution $\{f(i, p)\}$, let $|f| \triangleq \sum_i \sum_{p \in P_i} f(i, p)$. Let $\{f^*(i, p)\}$ denote an optimal dual solution. Using weak duality and since $\Delta_i P \leq 2 \cdot \Delta_i D$ it follows that:

$$|f^*| \leq \textstyle\sum_{e \in E} x_e \cdot c(e) + \sum_i z_i \leq 2 \cdot |f| \,, \qquad (13)$$

which proves 2-competitiveness; namely $|f| \geq \frac{1}{2} \cdot |f^*|$.

We now prove $\log(1 + 3 \cdot p_{\max})$-feasibility of the dual solution, i.e. for each $e \in E$, $\mathit{flow}(e) \leq \log(1 + 3 \cdot p_{\max})$. The update rule of the primal variables $\{x_e\}_e$ in Step 3b implies,

$$x_e = \frac{1}{p_{\max}} (2^{1/c(e)} - 1) \cdot \sum_{j=0}^{\mathit{flow}(e)-1} (2^{1/c(e)})^j$$

$$= \frac{1}{p_{\max}}(2^{1/c(e)} - 1) \cdot \frac{2^{flow(e)/c(e)} - 1}{2^{1/c(e)} - 1}$$
$$= \frac{2^{flow(e)/c(e)} - 1}{p_{\max}} . \tag{14}$$

The update rule requires that $\alpha(p, i) < 1$ for every p. Hence, before the update $x_e < 1$, and after the update $x_e < 2^{1/c(e)} + \frac{1}{p_{\max}} \cdot (2^{1/c(e)} - 1)$. Since $c_{\min} \geq 1$, it follows that $x_e < 3$.

By Eq. (14) it follows that

$$\frac{2^{flow(e)/c(e)} - 1}{p_{\max}} < 3 .$$

Implying that $flow(e) \leq \log(1 + 3 \cdot p_{\max}) \cdot c(e)$, as required. □

Appendix 6: Two Models For Nodes in Store-and-Forward Networks

The literature contains two different models of node functionality. In an effort to make the comparison concrete and perhaps clearer, we present schematic implementations of the nodes in each model.

To simplify the discussion, we use two type of packets: regular packets and ghost packets. A regular packet contributes a unit to the throughput (if delivered) and a ghost packet does not contribute to the throughput and acts as a "place holder". We therefore may treat a buffer as if it always contains B packets. If a buffer contains only ghost packets, then it is empty in reality. A reasonable policy does not drop a regular packet while keeping a ghost packet.

Model 1 This model is used by [6,22]. Figure 14a depicts a block diagram of a node. A node contains a combinational circuit *comb*, a buffer consisting of B flip-flops, and c flip-flops on each link that emanates the node.

In each clock cycle, the combinational circuit *comb* receives c packets from each incoming link, B packets from its buffer, and $B + c$ packets from its local inputs. It outputs B packets to the buffer and c packets along each outgoing link. Packets that were input but not output are considered dropped packets unless the node is their destination.

Model 2 This model is used by [3,7]. Figure 14b depicts a block diagram of a node. A node contains two combinational circuits $comb_0$ and $comb_1$, two sets of B latches, and one latch on the link that emanates the node. Note that this implementation uses a two-phase clock. The phases are denoted by ϕ_0 and ϕ_1.

In the first phase of each clock cycle, the combinational circuit $comb_0$ receives one packet from the incoming link, B packets from its buffer, and B packets from its local input. In total $2B + 1$ packets (either regular or ghost packets) are fed to the $comb_0$ circuit. The $comb_0$ circuit outputs B packets and the rest are dropped unless this is

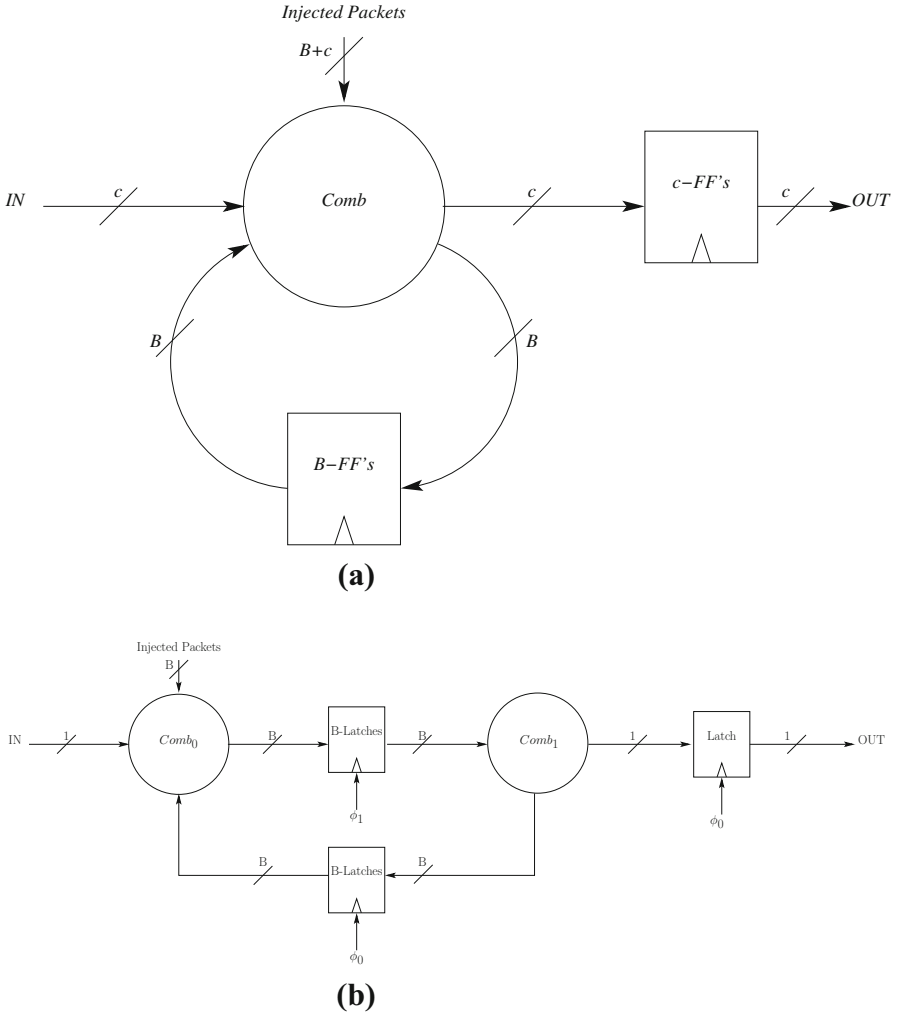


Fig. 14 **a** A schematic of a node in Model-1. **b** A schematic of a node in Model-2

their destination. In the second clock phase of each clock cycle, the combinational circuit $comb_1$ outputs one packet along the outgoing link and B packets are sent back to $comb_0$.

Remark 1 The setting $B = c = 1$ in Model 1 is strictly stronger than $B = 1$ in Model 2. Indeed, in Model 1, if a node receives a regular packet from its neighbor and is also input a regular packet locally, then it may store one packet and forward the other one. On the other hand, in Model 2, one of the packets must be dropped.

2. We could also allow for more injected packets in each node. In this case, the node must drop some of them. Of course, the online algorithm has to decide which packets should be dropped.
3. The linear lower bounds for $B = 1$ in [3,7] hold only with respect to Model 2.
4. It is not clear how to extend Model 2 for the case that $c > 1$ or $B = 0$.
5. Under the common assumption that the cost of a flip-flop is roughly twice the cost of a latch, the hardware needed for the latches of a node in Model 2 is roughly the same as the cost of flip-flops of a node in Model 1 (with $c = 1$).

References

1. Awerbuch, B., Azar, Y., Amos, F.: Packet routing via min-cost circuit routing. In: ISTCS, pp. 37–42 (1996)
2. Awerbuch, B., Azar, Y., Plotkin, S.: Throughput-competitive on-line routing. In: FOCS'93: Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science, pp. 32–40. IEEE Computer Society, Washington (1993)
3. Angelov, S., Khanna, S., Kunal, K.: The network asa storage device: dynamic routing with bounded buffers. Algorithmica **55**(1), 71–94 (2009). (Appeared in APPROX-05)
4. Aiello, W., Kushilevitz, E., Ostrovsky, R., Rosén, A.: Dynamic routing on networks with fixed-size buffers. In: SODA, pp. 771–780 (2003)
5. Adler, M., Khanna, S., Rajaraman, R., Rosén, A.: Time-constrained scheduling of weighted packets on trees and meshes. Algorithmica **36**(2), 123–152 (2003)
6. Adler, M., Rosenberg, A.L., Sitaraman, R.K., Unger, W.: Scheduling time-constrained communication in linear networks. Theory Comput. Syst. **35**(6), 599–623 (2002)
7. Azar, Y., Zachut, R.: Packet routing and information gathering in lines, rings and trees. In: ESA, pp. 484–495 (2005). See also manuscript in http://www.cs.tau.ac.il/~azar/
8. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, New York (1998)
9. Bartal, Y., Leonardi, S.: On-line routing in all-optical networks. In: ICALP, pp. 516–526 (1997)
10. Buchbinder, N., Naor, J.: Improved bounds for online routing and packing via a primal-dual approach. In: Annual IEEE Symposium on Foundations of Computer Science, pp. 293–304 (2006)
11. Buchbinder, N., Naor, J.S.: The design of competitive online algorithms via a primal-dual approach. Found. Trends Theor. Comput. Sci. **3**(2–3), 99–263 (2009)
12. Buchbinder, N., Naor, J.S.: Online primal-dual algorithms for covering and packing. Math. Oper. Res. **34**(2), 270–286 (2009)
13. Even, G., Medina, M.: An $o(logn)$-competitive online centralized randomized packet-routing algorithm for lines. In: Abramsky, S., Gavoille, C., Kirchner, C., auf der Heide, F.M., Spirakis, P.G. (eds.) ICALP (2), volume 6199 of Lecture Notes in Computer Science, pp. 139–150. Springer (2010)
14. Even, G., Medina, M.: Online packet-routing in grids with bounded buffers. In: Rajaraman, R., auf der Heide, F.M. (eds) SPAA, pp. 215–224. ACM (2011)
15. Even, G., Medina, M., Patt-Shamir, B.: Better deterministic online packet routing on grids. In: Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13–15, 2015, pp. 284–293 (2015)
16. Even, G., Medina, M., Rosén, A.: A constant approximation algorithm for scheduling packets on line networks. CoRR. arXiv:1602.06174, 2016. ESA (2016)
17. Gupta, U.I., Lee, D.T., Leung, J.Y.-T.: Efficient algorithms for interval graphs and circular-arc graphs. Networks **12**(4), 459–467 (1982)
18. Kleinberg, J.M., Tardos, É.: Disjoint paths in densely embedded graphs. In: FOCS, pp. 52–61 (1995). See also manuscript in http://www.cs.cornell.edu/home/kleinber/
19. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-shop scheduling in $O(congestion + dilation)$ steps. Combinatorica **14**(2), 167–186 (1994)
20. Leighton, T., Maggs, B., Richa, A.W.: Fast algorithms for finding $O(congestion + dilation)$ packet routing schedules. Combinatorica **19**(3), 375–401 (1999)

21. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
22. Räcke, H., Rosén, A.: Approximation algorithms for time-constrained scheduling on line networks. In: SPAA, pp. 337–346 (2009)
23. Rabani, Y., Tardos, É.: Distributed packet switching in arbitrary networks. In: STOC'96: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 366–375. ACM, New York (1996)
24. Srinivasan, A., Teo, C.P.: A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 636–643. ACM (1997)
25. Turner, J.S.: Strong performance guarantees for asynchronous buffered crossbar scheduler. IEEE/ACM Trans. Netw. **17**(4), 1017–1028 (2009)