Packet Routing and Job-Shop Scheduling in O(Congestion + Dilation) Steps

F. T. Leighton[1]
Bruce M. Maggs[2]
Satish B. Rao[2]

[1]Mathematics Department and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

[2]NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

In this paper, we prove that there exists a schedule for routing any set of packets with edge-simple paths, on any network, in $O(c+d)$ steps, where c is the congestion of the paths in the network, and d is the length of the longest path. The result has applications to packet routing in parallel machines, network emulations, and job-shop scheduling.

This research was conducted while the authors were at MIT. Support was provided by the Defense Advanced Research Projects Agency under Contract N00014–87–K–825, the Office of Naval Research under Contract N00014–86–K–0593, the Air Force under Contract OSR–86–0076, and the Army under Contract DAAL–03–86–K–0171. Tom Leighton is supported by an NSF Presidential Young Investigator Award with matching funds provided by IBM.

1 Introduction

Packet routing plays a central role in the design of large-scale parallel computers. Simply stated, packet routing consists of moving packets of data from one location to another in a network. The goal is to move all of the packets to their desired locations as quickly as possible, and with as little queuing as possible. The packet routing problem has been extensively studied, and we refer the reader to [5] for a broader coverage of the topic.

The method of packet routing considered in this paper is known as *store-and-forward routing.* In a store-and-forward routing algorithm, packets are viewed as atomic objects. At each step, a packet can either wait in a queue or jump from one queue to another.

Figure 1 shows a graph model for store-and-forward routing. The shaded nodes labeled 1 through 5 in the figure represent processors or switches, and the edges between the nodes represent wires. A packet is depicted by a square box containing the label of its destination. The goal is to route the packets from their origins to their destinations via a series of synchronized time steps, where at each step at most one packet can traverse each edge.

Packets wait in three different kinds of queues. Before the routing begins, packets are stored at their origins in special *initial queues.* For example, packets 4 and 5 are stored in the initial queue at node 1. When a packet traverses an edge, it enters the *edge queue* at the end of that edge. A packet can traverse an edge only if at the beginning of the step the edge queue at the end of that edge is not full. In the example of Figure 1 the edge queues can hold two packets. Upon traversing the last edge on its path, a packet is removed from the edge queue and placed in a special *final queue* at its destination. For simplicity, the final queues are not shown in Figure 1. Independent of the routing algorithm used, the size of the initial and final queues are determined by the particular packet routing problem to be solved. Thus, any bound on the maximum queue size required by a routing algorithm refers only to the edge queues.

This paper focuses on the problem of timing the movements of the packets along their paths. A *schedule* for a set of packets specifies which move and which wait at each time step. Given any underlying network, and any selection of paths for the packets, our goal is to produce a schedule for the packets that minimizes the total time and the maximum queue size needed to route all the packets to their destinations.

Of course, there is a strong correlation between the time required to route the packets and the selection of the paths. In particular, the maximum

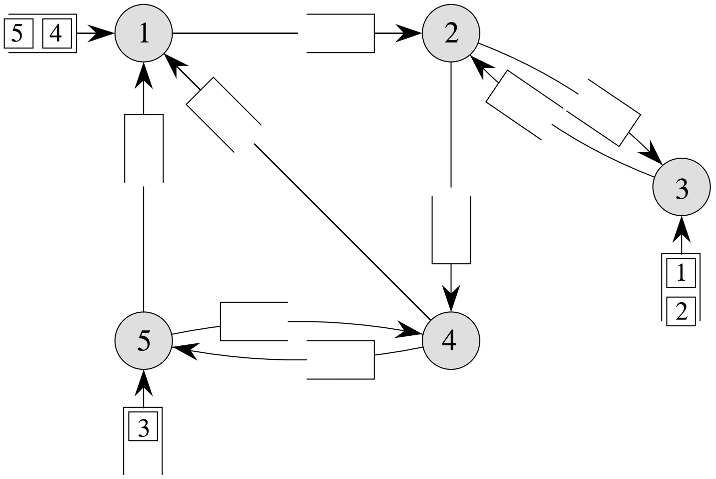

Figure 1: A graph model for packet routing.

distance, d, traveled by any packet is always a lower bound on the time. We call this distance the *dilation* of the paths. Similarly, the largest number of packets that must traverse a single edge during the entire course of the routing is a lower bound. We call this number the *congestion*, c, of the paths. For example, see Figure 2.

Given any set of paths with congestion c and dilation d in any network, it is straightforward to route all of the packets to their destinations in cd steps using queues of size c at each edge. Each packet can be delayed at most $c-1$ steps at each of at most d edges on the way to its destination (since the queues are big enough so that packets can never be delayed by a full queue in front.)

In this paper, we show that there are much better schedules. We begin in Section 2 with a randomized on-line algorithm that produces a schedule of length $O(c + d\log(Nd))$ using queues of size $O(\log(Nd))$, where N is the total number of packets. This algorithm is close to optimal when c is large. Our main result appears in Section 3. It establishes the existence of a schedule using $O(c+d)$ steps and constant-size queues at every edge, thereby achieving the naive lower bounds for any routing problem.

The proof of the main result is nonconstructive. However, the result still has several applications, as described below. In addition, the result is

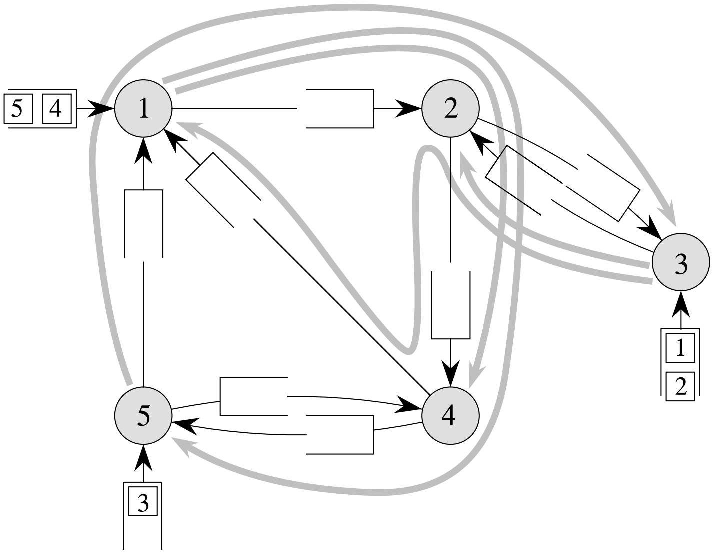

Figure 2: A set of paths for the packets with dilation $d = 3$ and congestion $c = 3$.

highly robust in the sense that it works for any set of edge-simple paths and any underlying network. (Ā priori, it would be easy to imagine that there might be some set of paths on some network that required more than $\Omega(c+d)$ steps or nonconstant queues to route all the packets.) Furthermore, the main result can be made constructive using the recently discovered algorithmic version of the Lovász Local Lemma [1, 2]. A manuscript describing the algorithm is in preparation [7].

If a particular routing problem is to be performed many times over, then the time required to compute the optimal schedule once becomes less important. This situation arises in network emulation problems. Typically, a guest network G is emulated by a host network H by embedding G into H. (For a more complete discussion of emulations and embeddings, see [3].) An *embedding* maps nodes of G to nodes of H, and edges of G to paths in H. There are three important measures of an embedding: the load, congestion, and dilation. The *load* of an embedding is the maximum number of nodes of G that are mapped to any one node of H. The *congestion* is the maximum number of paths corresponding to edges of G that use any one edge of H. The *dilation* is the length of the longest path. Let l, c, and d denote the load, congestion, and dilation of the embedding. Once G has been embedded in H, H can emulate G in a step-by-step fashion. Each node of H first emulates the local computations performed by the l (or fewer) nodes mapped to it. This takes $O(l)$ time. Then for each packet sent along an edge of G, H sends a packet along the corresponding path in the embedding. Using the main result of this paper, routing the packets to their destinations takes $O(c+d)$ steps. Thus, H can emulate each step of G in $O(l+c+d)$ steps.

In a related paper, Leighton, Maggs, Ranade, and Rao [6] show how to route packets in $O(c+L+\log N)$ steps using a simple randomized algorithm provided that the underlying network is leveled and has depth L. As a consequence, optimal routing algorithms can be derived for most of the networks that are commonly used for parallel computation. Unfortunately, it seems to be difficult to extend this result to hold for all networks. In fact, we have considered many simple on-line algorithms (including the algorithm presented in [6]), and found routing problems for each algorithm that result in schedules that use asymptotically more than $\Omega(c+d+\log N)$ steps. Several of these examples are included in Section 4.

The results of this paper also have applications to job-shop scheduling. In particular, consider a scheduling problem with jobs $j_1, \ldots, j_r$, and machines $m_1, \ldots, m_s$, for which each job must be performed on a specified sequence of machines. In our application, we assume that each job occupies each machine that works on it for a unit of time, and that no machine has to work on any job more than once. Of course, the jobs correspond to packets, and the machines correspond to edges in the packet routing problem. Hence, we can define the *dilation* of the scheduling problem to be the maximum number of machines that must work on any job, and the *congestion* to be the maximum number of jobs that have to be run on any machine. As a consequence of our packet routing result, we show that any scheduling problem can be solved in $O(c+d)$ steps. In addition, we will prove that there is a schedule for which each job waits at most $O(c+d)$ steps before it starts running, and that each job waits at most a constant number of steps in between consecutive machines. The queue of jobs waiting for any machine will also always be at most a constant. These results are optimal, and are substantially better than previously known bounds for this problem [4, 10].

Recently some results were proved in [11] for the more general problem of job-shop scheduling where jobs are not assumed to be unit length and a machine may have to work on the same job more than once. They give randomized and deterministic algorithms that produce schedules that are within a polylogarithmic factor of the optimal length for the more general job-shop problem. However, it is not known whether there exist schedules of length $O(c+d)$ for this problem.

This paper also leaves open the question of whether or not there is an on-line algorithm that can schedule any set of paths in $O(c+d)$ steps using constant-size queues. We suspect that finding such an algorithm (if one exists) will be a challenging task. Our negative suspicions are derived from the fact that we can construct counterexamples to most of the simplest on-line algorithms. In other words, for several natural on-line algorithms we can find paths for N packets for which the algorithm will construct a schedule using asymptotically more than $\Omega(c+d+\log N)$ steps. Several of the counterexamples are included in Section 4.

2 An on-line algorithm

There is a simple randomized on-line algorithm for producing a schedule of length $O(c+d\log(Nd))$ using queues of size $O(\log(Nd))$, where c is the congestion, d is the dilation, and N is the number of packets.

First, each packet is assigned a delay chosen randomly, independently, and uniformly from the interval $[1, \frac{\alpha c}{\log(Nd)}]$, where α is a constant that will be specified later. A packet that is assigned a delay of x waits in its initial queue for x time steps, and then moves on to its final destination without ever stopping.

The trouble with this schedule is that several packets may traverse the same edge in a single step. However, we can bound the number of packets that are likely to do so. The probability that more than $\log(Nd)$ packets use a particular edge g at a particular time step t is at most

$$\sum_{k=\log(Nd)+1}^{c} \binom{c}{k} \left(\frac{\log(Nd)}{\alpha c}\right)^{k} \left(1-\frac{\log(Nd)}{\alpha c}\right)^{c-k},$$

since at most c different packets pass through g, and for each of these, at most one of the $\frac{\alpha c}{\log(Nd)}$ possible delays sends it through g at step t. This sum is at most $\binom{c}{\log(Nd)}(\log(Nd)/\alpha c)^{\log(Nd)}$. To bound the probability that more than $\log(Nd)$ packets pass through any edge at any time step, we multiply this quantity by the number of choices for g, at most Nd, and the number of choices for t, at most $d+\frac{\alpha c}{\log(Nd)}$. Using the inequality $\binom{a}{b} \leq (ae/b)^b$ for $0<b<a$, and noting that $c \leq N$, we see that for large enough, but fixed, α, the product is at most $1/(Nd)$. Thus, with high probability, no more than $O(\log(Nd))$ packets will want to traverse any edge at any step of this unconstrained schedule.

Each step of the unconstrained schedule can be simulated by $O(\log(Nd))$ steps of a legitimate schedule. The final schedule requires $O(c+d\log(Nd))$ steps to complete the routing, and uses queues of size $O(\log(Nd))$.

3 An $O(c+d)$-step schedule

In this section, we prove that for any set of packets whose paths are edge-simple[1] and have congestion c and dilation d, there is a schedule of length $O(c+d)$ in which at most one packet traverses each edge of the network at each step, and at most a constant number of packets wait in each queue at each step. Note that there are no restrictions on the size, topology, or degree of the network or on the number of packets.

Our strategy for constructing an efficient schedule is to make a succession of refinements to the "greedy" schedule, S_0, in which each packet moves at every step until it reaches its final destination. This initial schedule is as short as possible; its length is only d. Unfortunately, as many as c packets

[1] An *edge-simple* path uses no edge more than once.

may have to use an edge at a single time step in S_0, whereas in the final schedule at most one packet is allowed to use an edge at each step. Each refinement will bring us closer to meeting this requirement by bounding the congestion within smaller and smaller frames of time.

The proof uses the Lovász Local Lemma [12, pp. 57–58] at each refinement step. Given a set of "bad" events in a probability space, the lemma provides a simple inequality which, when satisfied, guarantees that with probability greater than zero, no bad event occurs. The inequality relates the probability that each bad event occurs with the dependence among them. A set of events $A_1, \ldots, A_m$ in a probability space has *dependence* at most b if every event is mutually independent of some set of $m - b - 1$ other bad events. The lemma is nonconstructive; for a discrete probability space, it shows only that there exists some elementary outcome that is not in any bad event.

Lemma 3.1 (Lovász) *Let $A_1, \ldots, A_m$ be a set of "bad" events each occurring with probability p with dependence at most b. If $4pb < 1$, then with probability greater than zero, no bad event occurs.* □

3.1 A preliminary result

Before proving the main result of this section, we show that there is a schedule of length $(c+d)2^{O(\log^*(c+d))}$ that uses queues of size $\log(c+d)2^{O(\log^*(c+d))}$. This preliminary result is substantially simpler to prove because of the relaxed bounds on the schedule length and queue size. Nevertheless, it illustrates the basic ideas necessary to prove the main result. We begin by proving a lemma that is used in the proofs of both the preliminary result and the main result.

Before proceeding, we need to introduce some notation. A T-frame is a sequence of T consecutive time steps. The *frame congestion*, C, in a T-frame is the largest number of packets that traverse any edge in the frame. The *relative congestion*, R, in a T-frame is the ratio C/T of the congestion in the frame to the size of the frame.

Lemma 3.2 *For any set of packets whose paths are edge-simple and have congestion c and dilation d, there is a schedule of length $O(c + d)$ in which packets never wait in edge queues and in which the relative congestion in any frame of size $\log d$ or greater is at most 1.*

Proof: The proof uses the Lovász Local Lemma. The first step is to assign an initial delay to each packet. Without loss of generality, we assume that $c = d$. The delays are chosen from the range $[1, \alpha d]$, where α is a fixed constant that will be determined later. In the resulting schedule, S_1, a packet that is assigned a delay of x waits in its initial queue for x steps, then moves on to its destination without waiting again until it enters its final queue. The length of S_1 is at most $(1+\alpha)d$. We use the Lovász Local Lemma to show that if the delays are chosen randomly, independently, and uniformly, then with nonzero probability the relative congestion in any frame of size $\log d$ or greater is at most 1. Thus, such a set of delays must exist.

To apply the Lovász Local Lemma, we associate a bad event with each edge. The bad event for edge g is that more than T packets use g in some T-frame, for $T \geq \log d$. To show that there is a way of choosing the delays so that no bad event occurs, we need to bound the dependence, b, among the bad events and the probability, p, of each individual bad event occurring.

The dependence calculation is straightforward. Whether or not a bad event occurs depends solely on the delays assigned to the packets that pass through the corresponding edge. Thus, two bad events are independent unless some packet passes through both of the corresponding edges. Since at most c packets pass through an edge, and each of these packets passes through at most d other edges, the dependence, b, of the bad events is at most $cd = d^2$.

Computing the probability of each bad event is a little trickier. Let p be the probability of the bad event corresponding to edge g. Then

$$p \leq \sum_{T=\log d}^{d} (1+\alpha)d \binom{d}{T} \left(\frac{T}{\alpha d}\right)^{T}.$$

This expression is derived as follows. Frames of size greater than d cannot have relative congestion greater than 1, since the total congestion is only d. Thus, we can ignore them. We bound the probability that any frame has relative congestion greater than 1 by summing, over all frame sizes T from $\log d$ to d, the probability that some T-frame has relative congestion greater than 1. Furthermore, for any T, there are at most $(1+\alpha)d$ different T-frames and we bound the probability that any one of them has relative congestion greater than 1 by summing their individual probabilities. The number of packets passing through g in any T-frame has a binomial distribution. There are d independent Bernoulli trials, one for each packet that uses g. Since at most T of the possible αd delays will actually send a packet through g

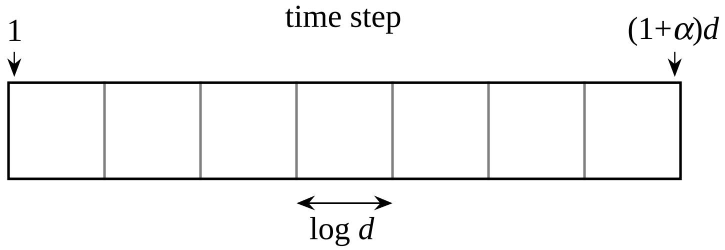

Figure 3: Schedule S_1. The schedule is derived from the greedy schedule, S_0, by assigning an initial delay in the range $[1, \alpha d]$ to each packet. We use the Lovász Local Lemma to show that within each $\log d$-frame, at most $\log d$ packets pass through each edge.

in the frame, each trial succeeds with probability $T/\alpha d$. (Here we use the assumption that the paths are edge-simple.) The probability of more than T successes is at most $\binom{d}{T}(T/\alpha d)^T$.

For sufficiently large, but fixed, α the product $4pb$ is less than 1, and thus, by the Lovász Local Lemma, there is some assignment of delays such that the relative congestion in any frame of size $\log d$ or greater is at most 1. □

Theorem 3.3 *For any set of packets whose paths are edge-simple and have congestion c and dilation d, there is a schedule having length $(c+d)2^{O(\log^*(c+d))}$ and maximum queue size $\log(c+d)2^{O(\log^*(c+d))}$ in which at most one packet traverses each edge at each step.*

Proof: For simplicity, we shall assume without loss of generality that $c = d$, so that the bounds on the length and queue size are $d2^{O(\log^* d)}$ and $(\log d)2^{O(\log^* d)}$, respectively.

The proof has the following outline. We begin by using Lemma 3.2 to produce a schedule S_1 in which the number of packets that use an edge in any $\log d$-frame is at most $\log d$. Next we break the schedule into $(1+\alpha)d/\log d$ $\log d$-frames, as shown in Figure 3. Finally, we view each $\log d$-frame as a routing problem with dilation $\log d$ and congestion $\log d$, and solve it recursively.

Each $\log d$-frame in S_1 can be viewed as a separate scheduling problem where the origin of a packet is its location at the beginning of the frame, and its destination is its location at the end of the frame. If at most $\log d$ packets use each edge in a $\log d$-frame, then the congestion of the problem is

$\log d$. The dilation is also $\log d$ because in $\log d$ time steps a packet can move a distance of at most $\log d$. In order to schedule each frame independently, a packet that arrives at its destination before the last step in the rescheduled frame is forced to wait there until the next frame begins.

All that remains is to bound the length of the schedule and the size of the queues. The recursion proceeds to a depth of $O(\log^* d)$ at which point the frames have constant size, and at most a constant number of packets use each edge in each frame. The resulting schedule can be converted to one in which at most one packet uses each edge in each time step by slowing it down by a constant factor. Since the length of the schedule increases by a constant factor during each recursive step, the length of the final schedule is $d2^{O(\log^* d)}$. The bound on the queue size follows from the observation that no packet waits at any one spot (other than its origin or destination) for more than $(\log d)2^{O(\log^* d)}$ consecutive time steps, and in the final schedule at most one packet traverses each edge at each time step. □

3.2 The main result

Proving that there is a schedule of length $O(c+d)$ using constant-size queues is more difficult. Removing the $2^{O(\log^*(c+d))}$ factor in the length of the schedule seems to require delving into second-order terms in the probability calculations, and reducing the queue size to a constant mandates greater care in spreading delays out over the schedule.

Theorem 3.4 *For any set of packets with edge-simple paths having congestion c and dilation d, there is a schedule having length $O(c+d)$ and constant maximum queue size in which at most one packet traverses each edge of the network at each step.*

Proof: To make the proof more modular, we bound the frame size and relative congestion after each step of the construction in lemmas. These lemmas and their proofs are included within the proof of the theorem. We assume without loss of generality that $c = d$, so that the bound on the length of the schedule is $O(d)$.

As before, the strategy is to make a succession of refinements to the greedy schedule, S_0. The first refinement is special. It transforms S_0 into a schedule S_1 in which the relative congestion in each frame of size $\log d$ or more is at most 1. Thereafter, each refinement transforms a schedule S_i with relative congestion at most $r^{(i)}$ in any frame of size $I^{(i)}$ or greater

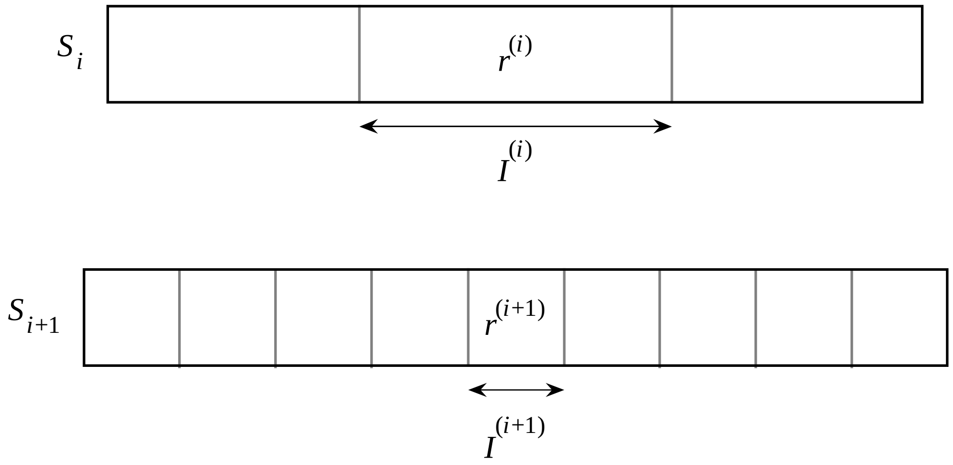

Figure 4: A refinement step. Each refinement transforms a schedule S_i into a slightly longer schedule S_{i+1}. The frame size is greatly reduced in S_{i+1}, yet the relative congestion within a frame remains about the same, i.e., $I^{(i+1)} \ll I^{(i)}$ and $r^{(i+1)} \approx r^{(i)}$.

into a schedule S_{i+1} with relative congestion at most $r^{(i+1)}$ in any frame of size $I^{(i+1)}$ or greater, where $r^{(i+1)} \approx r^{(i)}$ and $I^{(i+1)} \ll I^{(i)}$, as shown in Figure 4. As well shall see, after j refinements, where $j = O(\log^* d)$, we obtain a schedule S_j with constant relative congestion in every frame of size k_0 or greater, where k_0 is some constant. From S_j it is straightforward to construct a schedule of length $O(c+d)$ in which at most one packet traverses each edge of the network at each step, and at most a constant number of packets wait in each queue at each step.

At the start, the relative congestion in a d-frame of S_0 is at most 1. We begin by using Lemma 3.2 to produce a schedule S_1 of length $O(d)$ in which the relative congestion is at most $r^{(1)} = 1$ in any frame of size $I^{(1)} = \log d$ or greater.

Next, we repeatedly refine the schedule to reduce the frame size. As we shall see, the relative congestion $r^{(i+1)}$ and frame size $I^{(i+1)}$ for schedule

S_{i+1} are given by the recurrences

$$r^{(i+1)} = \begin{cases} 1 & i = 0 \\ r^{(i)}(1 + O(1)/\sqrt{\log I^{(i)}}) & i > 0 \end{cases}$$

and

$$I^{(i+1)} = \begin{cases} \log d & i = 0 \\ \log^5 I^{(i)} & i > 0 \end{cases}$$

which have solutions $I^{(j)} = O(1)$ and $r^{(j)} = O(1)$ for some j, where $j = O(\log^* d)$.

We have not explicitly defined the values of $r^{(i)}$ and $I^{(i)}$ for which the recursion terminates. However, in several places in the proof that follows we implicitly use the fact that $I^{(i)}$ is sufficiently large that some inequality holds. The recursion terminates when the first of these inequalities fails to hold. When this happens, $I^{(i)}$ is bounded from above by some constant. Furthermore, independent of the depth of the recursion, $r^{(i)}$ is bounded from above by a constant.

Throughout the following lemmas we make references to quantities such as rI packets or $\log^4 I$ time steps, when in fact rI and $\log^4 I$ may not be integral. Rounding these quantities to integer values when necessary does not affect the correctness of the proof. For ease of exposition, we shall henceforth cease to consider the issue.

An important invariant that we maintain throughout the construction is that in schedule S_{i+1} every packet waits at most once every $I^{(i)}$ steps. As a consequence, there is a constant k_1 such that a packet waits at most once every k_1 steps in S_j, which implies both that the queues in S_j cannot grow larger than a constant and that the total length of S_j is $O(d)$. Schedule S_j almost satisfies the requirement that at most one packet traverses each edge in each step. By simulating each step of S_j in a constant number of steps we can meet this requirement with only a factor of 2 increase in the queue size and a constant factor increase in the running time.

The rest of the proof describes the refinement step in detail. For ease of notation, we use I and r in place of $I^{(i)}$ and $r^{(i)}$.

The first step in the ith refinement is to break schedule S_i into *blocks* of $2I^3+2I^2-I$ consecutive time steps. Each block is rescheduled independently.

For each block, each packet is assigned a delay chosen from 1 to I. We will use the Lovász Local Lemma to show that if the delays are chosen randomly, uniformly, and independently, then with non-zero probability the resulting schedule will have the properties that we want.

A packet that is assigned a delay of x must wait for x steps at the beginning of the block. In order maintain the invariant that in schedule S_{i+1} every packet waits at most once every $I^{(i)}$ steps, the packet is not delayed for x consecutive steps at the beginning of the block, but instead a delay is inserted every I steps in the first xI steps of the block. A packet that is delayed x steps reaches its destination at the end of the block by step $2I^3 + 2I^2 - I + x$.

In order to independently reschedule the next block, the packets must reside in exactly the same queues at the end of the rescheduled block that they did at the end of the block of S_i. Since some packets arrive early, they must be slowed down. Thus, if a packet is assigned delay x, then $I - x$ delays are inserted in the last $I(I - x)$ steps of the block, one every I steps. Since every packet experiences a total delay of I, the rescheduled block must have length $2I^3 + 2I^2$.

Before the delays for schedule S_{i+1} have been inserted, a packet is delayed at most once in each block of S_i, provided that $2I^3 + 2I^2 - I < I^{(i-1)}$, which holds as long as I is larger than some constant. Prior to inserting each new delay into a block, we check if it is within I steps of the single old delay. If the new delay would be too close to the old delay, then it is simply not inserted. The loss of a single delay in a block has a negligible effect on the probability calculations in the lemmas that follow.

The following two lemmas are used several times in the proof of the theorem. Lemma 3.5 shows that if we can bound the relative congestion in frames of size T to $2T - 1$, then we can bound the relative congestion in all frames of size T or greater. Lemma 3.6 bounds the probability that too many packets use any particular edge g in any small frame in the center of a block after every packet has been delayed for a random number of steps at the beginning of the block.

Lemma 3.5 *In any schedule, if the number of packets that use a particular edge g in any y-frame is at most Ry, for all y between T and $2T - 1$, then the number of packets that use g in any y-frame is at most Ry for all $y \geq T$.*

Proof: Consider a frame of size T', where $T' > 2T - 1$. The first $(\lfloor T'/T \rfloor - 1)T$ steps of the frame can be broken into T-frames. In each of these frames, at most RT packets use g. The remainder of the T'-frame consists of a single y-frame, where $T \leq y \leq 2T-1$, in which at most Ry packets use g. $\square$

Lemma 3.6 *Suppose that there are positive constants ρ, α_1, and α_2, such that in a block of size I^{α_1} or smaller the relative congestion is at most ρ in frames of size I^{α_2} or larger. Furthermore, suppose that each packet is assigned a delay chosen randomly, independently, and uniformly from the range $[1, I^{\alpha_2}]$ and that if a packet is assigned a delay of x, then x delays are inserted in the first I^{α_3} steps of the block and $I^{\alpha_2} - x$ delays are inserted in the last I^{α_3} steps, where α_3 is also a positive constant. Then for any constant α_4 there is a σ such that the probability that more than $\rho_1 T$ packets use any one edge g in any frame of size $T \geq I_1$ in-between the first and last I^{α_3} steps in the new block is at most $1/I^{\alpha_4}$, where $I_1 = \log^2 I$, $\rho_1 = \rho(1+\sigma)$, and $\sigma = O(1)/\sqrt{\log I}$.*

Proof: We begin by computing an upper bound on the probability, p_1, that more than $\rho_1 I_1$ packets use an edge g in a particular I_1-frame. Since a packet may be delayed up to I^{α_2} steps before the frame, any packet that used g in the I_1-frame spanning the same steps in the block before the delays were inserted or in the I^{α_2} steps before that frame may use g after the delays are inserted. Thus, there are at most $\rho(I^{\alpha_2} + I_1)$ packets that can use g in the frame. For each of these, the probability that the packet uses g in the frame after being delayed is at most (I_1/I^{α_2}), provided that the packet's path uses g at most once. Thus, the probability p_1 that more than $\rho_1 I_1$ packets use g in the frame is bounded by

$$p_1 \leq \sum_{k=\rho_1 I_1}^{\rho(I^{\alpha_2}+I_1)} \binom{\rho(I^{\alpha_2}+I_1)}{k} (I_1/I^{\alpha_2})^k (1 - I_1/I^{\alpha_2})^{\rho(I^{\alpha_2}+I_1)-k}.$$

Let $\rho_1 = \rho(1+\sigma)$. We bound the series as follows. The expected number of packets that use g in the frame is $\rho I_1(1 + I_1/I^{\alpha_2})$. For $I_1 = \log^2 I$ and $\sigma = O(1)/\sqrt{\log I}$, $\rho I_1(1+\sigma)$ is larger than the expectation, so the first term in the series is the largest, and there are at most $\rho(I^{\alpha_2} + I_1)$ terms. Applying the inequalities $(1+x) \leq e^x$, $\ln(1+x) \geq x - x^2/2$ for $0 \leq x \leq 1$, and $\binom{a}{b} \leq (ae/b)^b$ for $0 < b < a$ to this term, we have

$$p_1 \leq \rho(I^{\alpha_2} + I_1) e^{-\rho I_1 \sigma^2 (1/2 - \sigma/2 - I_1/\sigma^2 I^{\alpha_2} - 2I_1/\sigma I^{\alpha_2})}.$$

For $I_1 = \log^2 I$ and $\sigma = k_1/\sqrt{\log I}$, we can ensure that $p_1 \leq 1/I^{k_2}$, for any constant $k_2 > 0$ by making constant k_1 large enough.

Next we need to bound the probability p_2 that more than $\rho_1 I_1$ packets use g in any I_1-frame of the block. There are at most $I^{\alpha_1} + I^{\alpha_2}$ I_1-frames. Thus $p_2 \leq (I^{\alpha_1} + I^{\alpha_2})p_1$. By making the constant k_2 large enough, we can ensure that $p_2 \leq 1/I^{k_3}$, for any constant $k_3 > 0$.

To bound the relative congestion in frames of size greater than I_1, we appeal to Lemma 3.5. The calculations for frames of size $I_1 + 1$ through $2I_1 - 1$ are similar to those for frames of size I_1. There are at most $I^{\alpha_1} + I^{\alpha_2}$ frames of any one size, and I_1 frame sizes between I_1 and $2I_1 - 1$. By adjusting the constants as before, we can guarantee that the probability p that more than $\rho_1 T$ packets use g in any T-frame for T between I_1 and $2I_1 - 1$ is at most $1/I^{\alpha_4}$ for any constant $\alpha_4 > 0$. □

Lemma 3.7 shows that by inserting delays at the beginning and end of the block we can reduce the frame size in the center of the block while only slightly increasing the relative congestion. The bounds proved in Lemma 3.7 are shown in Figure 5.

Lemma 3.7 *There is some way of assigning delays to the packets so that in-between the first and last I^2 steps of a block, the relative congestion in any frame of size $I_1 = \log^2 I$ or greater is at most $r_1 = r(1 + \varepsilon_1)$, where $\varepsilon_1 = O(1)/\sqrt{\log I}$.*

Proof: The proof uses the Lovász Local Lemma. With each edge we associate a bad event. For edge g, a bad event occurs when more than $r_1 T$ packets use g in any T-frame for $T \geq I_1$. To show that no bad event occurs, we need to bound both the dependence of the bad events and the probability that an individual bad event occurs.

We first bound the dependence, b. At most $r(2I^3 + 2I^2 - I)$ packets use an edge in the block. Each of these packets travels through at most $2I^3 + 2I^2 - I$ other edges in the block. Furthermore, $r = r^{(i)} = O(1)$. Thus, a bad event depends on $b = O(I^6)$ other bad events.

For any constant α_4, we can bound the probability that a bad event occurs by $1/I^{\alpha_4}$ by applying Lemma 3.6 with $\rho = r$, $I^{\alpha_1} \geq 2I^3 + 2I^2 - I$, $I^{\alpha_2} = I$, $I^{\alpha_3} = I^2$, $\varepsilon_1 = \sigma = O(1)/\sqrt{\log I}$, and $r_1 = \rho_1 = r(1+\sigma) = r(1+\varepsilon_3)$.

Since a bad event depends on only $b = O(I^6)$ other bad events, we can make $4pb < 1$ by making α_4 large enough. By the Lovász Local Lemma, there is some way of choosing the packet delays so that no bad event occurs. □

Inserting delays into the schedule may increase the relative congestion in I-frames (or smaller frames) in the I^2 steps at the beginning and end

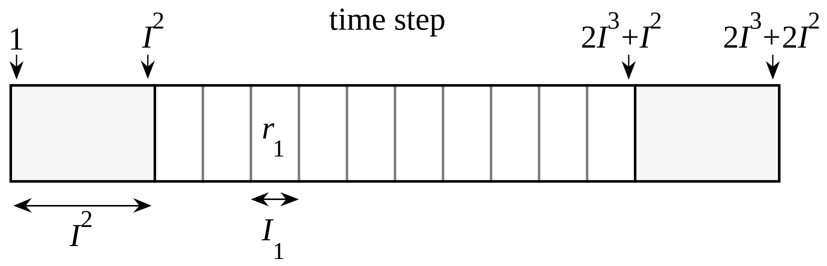

Figure 5: Bounds on frame size and relative congestion after inserting delays into S_i. Here $I_1 = \log^2 I$ and $r_1 = r(1 + O(1)/\sqrt{\log I})$.

of each block. In order to bound the relative congestion in small frames in these regions, we first move the block boundaries to the centers of the blocks, as shown in Figure 6. Now each block of size $2I^3 + 2I^2$ has a "fuzzy" region of size $2I^2$ in its center. Lemma 3.8 shows that after moving the block boundaries, the relative congestion in any frame of size I^2 or larger in the block is at most $r(1 + 2/I)$. We will later insert more delays into the schedule and uses Lemmas 3.6 and 3.8 to help bound the relative congestion in small frames in the fuzzy region.

Lemma 3.8 *For any choice of delays, after the delays are inserted and the block boundaries are moved the relative congestion in any frame of size I^2 or greater is at most $r(1 + 2/I)$.*

Proof: There are two cases to consider. First, consider a T-frame that lies entirely in the first half of a block, or entirely in the second half of a block. After the delays are inserted, a packet can use an edge in the T-frame only if it used the edge in some $(T + I)$-frame in S_i. Thus, at most $r(T + I)$ packets can use an edge in the T-frame. For $T \geq I^2$, the relative congestion is at most $r(1 + 1/I)$. Second, consider a T-frame that spans the center of the block. Suppose that the frame consists of T_1 steps before the center and T_2 after, so that $T = T_1 + T_2$. Then a packet can use an edge in the T_1 steps before the center only if it used the edge in one of the last T_1 steps before the end of a block in S_i. Since T_1 may be less than I, we can't bound the relative congestion in the last T_1 steps at the end of a block. But we know that at most $r(T_1 + I)$ packets used the edge in the last $T_1 + I$ steps, and hence in the last T_1 steps. Similarly, a packet can use an edge in the T_2 steps after the center only if it used an edge in one of the first T_2 steps of a

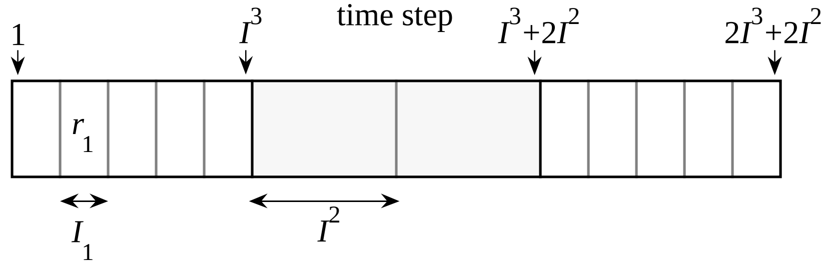

Figure 6: A block after recentering. The "fuzzy region" in the center of the block is shaded. The line bisecting the shaded region denotes the block boundary before recentering.

block in S_i. Hence, at most $r(T_2 + I)$ packets use the edge in the T_2 steps after the center. Since a total of at most $r(T_1 + T_2 + 2I) = r(T + 2I)$ packets use the edge, for $T \geq I^2$ the relative congestion is at most $r(1 + 2/I)$. □

To reduce the frame size in the fuzzy region, we assign a delay from 1 to I^2 to each packet. As before, we will use the Lovász Local Lemma to show that if the delays are chosen randomly, independently, and uniformly then with non-zero probability the resulting schedule has the properties we want. A packet with delay x waits once every I^3/x steps in the I^3 steps before the fuzzy region. In addition, a packet with delay x waits once every $I^3/(I^2 - x)$ steps in the last I^3 steps of the rescheduled block. Thus, every packet waits for a total of I^2 steps (except we do not insert a delay if it is within I steps of an old delay), and the rescheduled block now has size $2I^3 + 3I^2$. Note that in the rescheduled block the width of the fuzzy region grows by I^2 time steps; it spans steps I^3 through $I^3 + 3I^2$.

We now show that there is some way of inserting delays into the schedule before the fuzzy region that both reduces the frame size in the fuzzy region and does not increase either the frame size or the relative congestion before or after the fuzzy region by much.

Lemma 3.9 *There is some way of choosing the packet delays so that between steps $I \log^3 I$ and I^3 and between steps $I^3 + 3I^2$ and $2I^3 + 3I^2 - I \log^3 I$, the relative congestion in any frame of size I_1 or greater is at most $r_2 = r(1 + \varepsilon_2)$, where $\varepsilon_2 = O(1)/\sqrt{\log I}$, and so that in the fuzzy region the relative congestion in any frame of size I_1 or greater is at most $r_3 = r(1 + \varepsilon_3)$, where $\varepsilon_3 = O(1)/\sqrt{\log I}$.*

Proof: The proof uses the Lovász Local Lemma as before. With each edge we associate a bad event. For edge g, a bad event occurs

1. if more than r_3T packets use g in any T-frame between steps I^3 and $I^3 + 3I^2$ (i.e., in the fuzzy region), for any $T \geq I_1$, or
2. if more than r_2T packets use g in any T-frame between steps $I \log^3 I$ and I^3, for any $T \geq I_1$, or
3. if more than r_2T packets use g in any T-frame between steps $2I^3 + 3I^2 - I \log^3 I$ and $2I^3 + 3I^2$, for any $T \geq I_1$.

The calculation for the dependence b is the same as in Lemma 3.7. At most $O(I^3)$ packets pass through each edge g, and each of these packets passes through at most $O(I^3)$ other edges. Hence, $b = O(I^6)$.

To bound the probability that a bad event occurs, we consider the three cases separately, and sum their individual probabilities of occurrence.

Since no delays are inserted into the fuzzy region, we can use Lemma 3.6 to prove that for any constant k_5, there is an $\varepsilon_3 = O(1)/\sqrt{\log I}$ such that the probability that more than $r(1+\varepsilon_3)T$ packets use g in any T-frame between steps I^3 and $I^3 + 3I^2$, for any $T \geq I_1$, is at most $1/I^{k_5}$. We apply Lemma 3.6 with $\rho = r(1 + 2/I)$, $I^{\alpha_1} \geq 2I^3 + 3I^2$, $I^{\alpha_2} = I^2$, $I^{\alpha_3} = I^3$, $\varepsilon_3 = \sigma + 2(1+\sigma)/I = O(1)/\sqrt{\log I}$, $r_3 = \rho_1 = r(1+2/I)(1+\sigma) = r(1+\varepsilon_3)$, and $\alpha_4 = k_5$.

Before the fuzzy region, the situation is more complex. By the kth step, $0 \leq k \leq I^3$, a packet with delay x has waited xk/I^3 times. Thus, the delay of a packet at the kth step varies essentially uniformly from 0 to $u = k/I$. For $u \geq \log^3 I$, or equivalently, $k \geq I \log^3 I$, we can show that the relative congestion in any frame of size I_1 or greater has not increased much.

The probability p_2 that more than $r_2 I_1$ packets use an edge g in a particular I_1-frame is given by

$$p_2 \leq \sum_{s=r_2 I_1}^{r_1(I_1+u)} \binom{r_1(I_1+u)}{s} (I_1/u)^s (1 - I_1/u)^{r_1(I_1+u)-s}.$$

Using the same inequalities as in the proof of Lemma 3.6, we have

$$p_2 \leq r_1(I_1+u) e^{-r_1 I_1 \varepsilon_2^2 (1/2 - \varepsilon_2/2 - I_1/\varepsilon_2^2 u - 2I_1/\varepsilon_2 u)}.$$

The calculations for frame of size $I_1 + 1$ through $2I_1 - 1$ are similar. Thus for any constant k_6, for $I_1 = \log^2 I$, $u \geq \log^3 I$, and $\varepsilon_2 = O(1)/\sqrt{\log I}$, the probability p_4 that more than $r(1+\varepsilon_2)T$ packets use g in any T-frame between steps $I \log^3 I$ and I^3, for any $T \geq I_1$, is at most $1/I^{k_6}$.

By symmetry, the probability that more than $r_2 T$ packets use g between steps $2I^3 + 3I^2 - I \log^3 I$ and $2I^3 + 3I^2$, for any $T \geq I_1$, is also at most $1/I^{k_6}$.

Thus, the probability that a bad event occurs for edge g is at most $1/I^{k_5} + 2/I^{k_6}$. Since the dependence is at most $O(I^6)$, by adjusting the constants k_5 and k_6 we can apply the Lovász Local Lemma. □

For steps 0 to $I \log^3 I$, we use the following lemma to bound the frame size and relative congestion.

Lemma 3.10 *The relative congestion in any frame of size I_2 or greater between steps 0 and $I \log^3 I$ is at most r_4, where $I_2 = \log^4 I$ and $r_4 = r_1(1 + 1/\log I)$.*

Proof: The proof is similar to that of Lemma 3.8. □

We have now completed our transformation of schedule S_i into schedule S_{i+1}. Let us review the relative congestion and frame sizes in the different parts of a block. Between steps 0 and $I \log^3 I$, the relative congestion in any frame of size I_2 or greater is at most r_4. Between this region and the fuzzy region, the relative congestion in any frame of size I_1 or greater is at most r_2. In the fuzzy region, the relative congestion in any frame of size I_1 or greater is at most r_3. After the fuzzy region, the relative congestion in any frame of size I_1 or greater is again r_2, until step $2I^3 + 3I^2 - I \log^3 I$, where the relative congestion in any frame of size I_2 or greater is r_4. These bounds are shown in Figure 7. Finally we must bound the relative congestion in frames that span the different parts of a block (or two different blocks). Since we have bound the relative congestion in blocks of size $\log^4 I$ or greater, it is safe to say that in the the entire schedule S_{i+1} the relative congestion in any frame of size $I^{(i+1)} = \log^5 I$ or greater is at most $r^{(i+1)} = r(1 + O(1)/\sqrt{\log I})$. □

4 Counterexamples to on-line algorithms

This section presents examples where several natural on-line scheduling strategies do poorly. Based on these examples, we suspect that finding an on-line algorithm that can schedule any set of paths in $O(c+d)$ steps using constant-size queues will be a challenging task.

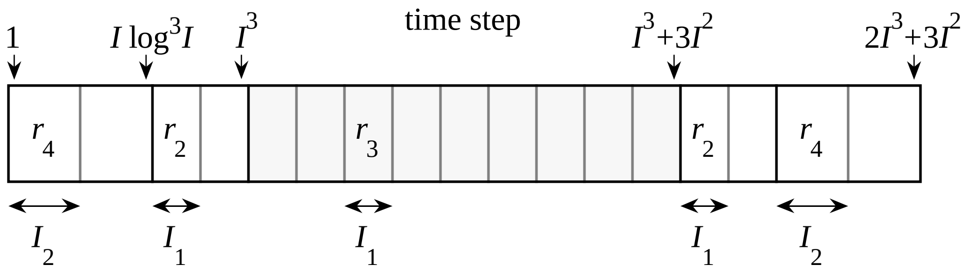

Figure 7: Final bounds on frame size and relative congestion. To reduce the frame size in the fuzzy regions, delays are inserted only outside the shaded region. Here $I_1 = \log^2 I$, $I_2 = \log^4 I$, $r_2 = r(1 + O(1)/\sqrt{\log I})$, $r_3 = r(1 + O(1)/\sqrt{\log I})$, and $r_4 = r_1(1 + 1/\log I) \leq r(1 + O(1)/\sqrt{\log I})$.

4.1 Counterexample for routing on leveled networks

In the first example, we examine a routing strategy for scheduling packets on leveled networks from [6, 8, 9]. A *leveled network* is a network whose switches can be partitioned into sets or levels labeled with integers so that every edge goes from a switch in some level i to a switch in the next level $i + 1$. The *depth* of the network is the maximum distance between two switches.

The routing strategy consists of randomly choosing ranks for the packets to be routed and using this value as a priority in a very strong manner; all the packets that use a switch must use it in order of rank. That is, the lowest ranked packet that uses the switch passes through the switch first, then the second lowest ranked packet passes through the switch and so on. Notice that at some point a packet with some rank may reach a switch before a packet with a lower rank reaches the switch through a different edge. In this case the packet must wait for the lower ranked packet to reach and use the switch before it can use the switch. So in order for a packet to decide if it can use a switch or not it must somehow know what the highest ranked packet that is going to enter the switch through some other edge is. This is achieved through the use of ghost messages. When a packet uses an outgoing edge of a switch it sends a ghost message consisting only of the packet's rank down all the other edges. These messages serve as a lower bound to each of these switches for the rank of any packet coming through this incoming edge, and are appropriately forwarded. Finally, end-of-stream(EOS) messages are used to indicate that no more packets will come from a switch. Thus, a packet is allowed to go if it is the lowest ranked packet on any incoming edge and it has a lower rank than the last ghost that arrived on incoming edges that do not have a packet and have not recieved an EOS message. This strategy is described in more detail in each of [6, 8, 9]. With high probability, it produces a schedule of length $O(c+L+\log N)$ using constant-size queues for any set of N packets whose paths have congestion c on any bounded-degree leveled network with depth L. For a wide variety of networks (both leveled and non-leveled), this algorithm can be used as a subroutine to derive a routing algorithm that delivers all the packets to their destinations in $O(c + d + \log N)$ time, with high probability.

In our first example, however, we show that this is not always the case. We describe an N-node leveled network in which a set of packets with congestion and dilation $O(1)$ requires $\Omega(\log^2 N/\log\log N)$ steps to be delivered using the strategy for scheduling packets on leveled networks from [6, 8, 9]. Our example does not contradict the previous analysis of the algorithm, since the network has $L = \Theta(\log^2 N)$ levels. However, it shows that reducing the congestion and dilation below the depth of the network does not necessarily improve the running time.

Observation 4.1 *For the leveled network scheduling strategy there is an N-node directed acyclic network of degree 3 and a set of paths with congestion $c = 3$ and dilation $d = 3$ where the expected length of the schedule is $\Omega(\log^2 N/\log\log N)$.*

Proof: The network consists of many disjoint copies of the subnetwork pictured in Figure 8. For simplicity, we dispense with the initial queues; the packets originate in edge queues. The subnetwork is composed of $k/\log k$ linear chains of length k, where k shall later be shown to be $\Theta(\log N)$. The second node of each linear chain is connected to the second to last node of the previous chain by a diagonal edge. We assume that at the end of each edge there is a queue that can store 2 packets. Initially, the queue into the first node of each chain contains an end-of-stream (EOS) signal and one packet, and the queue into the second node contains two packets. A packet's destination is the last node in the previous chain. Each packet takes the diagonal edge to the previous chain and then the last edge in the chain. Thus, the length of the longest path is $d = 3$. However, the depth of this subnetwork or any number of disjoint copies of this subnetwork is $\Theta(k^2/\log k)$. That is, there are at least $\Omega(k^2/\log k)$ levels in this network. We now proceed by showing that the time for routing can be $\Omega(k^2/\log k)$.

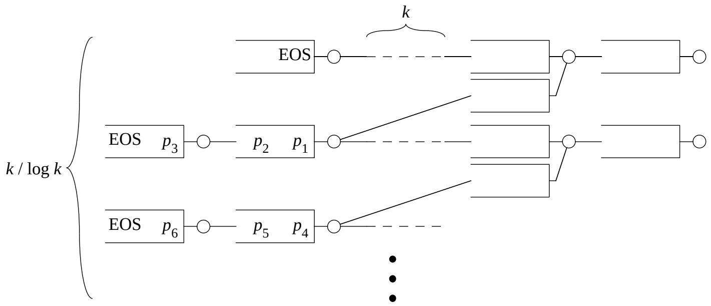

Figure 8: Example 1.

When the ranks $r_1, \ldots, r_{3k/\log k}$ of the packets $p_1, \ldots, p_{3k/\log k}$ are chosen so that $r_i < r_{i+1}$ for $1 \leq i < 3k/\log k$, packet $p_{3k/\log k}$ requires $\Omega(k^2/\log k)$ steps to reach its destination. The scenario unfolds as follows. Packets p_1 and p_2 take a diagonal edge in the first two steps. These packets cannot advance until the EOS reaches the end of the first chain, in step k. Thus p_3 remains in the previous queue until step k. In the meantime, ghosts with ranks r_1, r_2, and r_3, travel down the second chain, but packet p_3 blocks an EOS signal from traveling down the chain. Packets p_4 and p_5 move out of their chain and must wait for this EOS signal. They cannot advance until step $2k$. So p_6 cannot move out of its chain and let the EOS signal behind it through until this step, so p_9 cannot move out of its chain until step $3k$ and so on. In this fashion, a delay of $k^2/\log k$ is propagated down to packet $p_{3k/\log k}$.

A simple calculation reveals that the probability that $r_i < r_{i+1}$ for $1 \leq i \leq 3k/\log k$ is $1/2^{\Theta(k)}$. Thus, if we have $2^{\Theta(k)}$ copies of the subnetwork, we expect the ranks of the packets to be sorted in one of them. For the total number of nodes in the network to be N, we need $k = \Theta(\log N)$. In this case, we expect some packet to be delayed $\Omega(\log^2 N/\log\log N)$ steps in one copy of the subnetwork. □

It is somewhat unfair to say that the optimal schedule for this example has length $O(c + d) = O(1)$, since ghosts and EOS signals must travel a distance of $\Theta(\log N)$. However, even if the EOS signals are replaced by packets with equivalent ranks, the dilation is only $O(\log N)$, and thus the optimum schedule has length $O(\log N)$.

4.2 Counterexample for various deterministic strategies

The second example is quite general. It shows that for any deterministic strategy that chooses the order in which packets pass through a switch independent of the future paths of the packets, there is a network and a set of paths with congestion c and dilation d for which the schedule produced has length at least $c(d-1)/\log c$. This observation covers strategies such as giving priority to the packet that has spent the most (or least) time waiting in queues, and giving priority to the packet that arrives first at a switch. The network is a complete binary tree of height $d-1$ with an *auxiliary edge* from the root to an *auxiliary node*.

Observation 4.2 *For any deterministic strategy that chooses the order in which packets pass through a switch independent of the paths that the packets take after they pass through the switch, there is a network and a set of paths with congestion c and dilation d for which the schedule produced has length $c(d-1)/\log c$.*

Proof: We construct the example for congestion c and dilation d, $E(c,d)$, recursively. The base case is the example $E(c, \log c + 1)$. Each of the c leaves sends a packet to the auxiliary node, causing congestion c in the auxiliary edge. The network for $E(c,d)$ contains c copies of the network for $E(c, d - \log c)$, as shown in Figure 9. First, the auxiliary nodes for these copies are paired up and merged so that there are $c/2$ auxiliary nodes each with two auxiliary edges into it. Next, the auxiliary nodes become the leaves of a complete binary tree of height $\log c - 1$ with its own auxiliary node and edge. For each copy of $E(c, d - \log c)$, the deterministic scheduling strategy chooses some packet to cross its auxiliary edge last. We extend the path of this packet so that it traverses the auxiliary edge in $E(c,d)$. The dilation of the new set of paths is d and the congestion c. The length of the schedule, $T(c,d)$, is given by the recurrence

$$T(c,d) \geq \begin{cases} T(c, d-\log c) + \log c - 1 + c & d > \log c + 1 \\ \log c + c & d = \log c + 1 \end{cases}$$

and has solution $T(c,d) \geq c(d-1)/\log c$. Setting $c = d = \log N$ in this example gives a routing time of $\Theta(\log^2 N/\log\log N)$. □

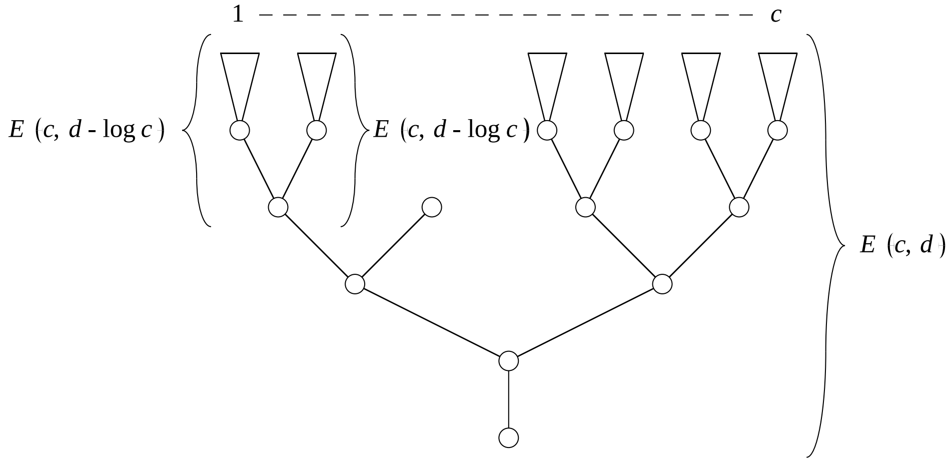

Figure 9: Example 2.

The previous example can be modified to show that the strategies of sending the packet with the farthest distance left to go or the packet with the farthest total initial distance to go first can also be made to require $\Omega(cd/\log c)$ time. We simply extend the paths of the packets winning at each switch so that they have total (or remaining) distance equal to or greater than the packets that lose at a switch.

4.3 Counterexample to a randomized strategy

The third example shows that the natural strategy of assigning priorities to the packets at random is not effective either.

Observation 4.3 *For the strategy of assigning each packet a random rank and giving priority to the packet with the lowest rank, there is an N-node network with diameter $O(\log N/\log\log N)$ and a set of paths with dilation $d = O(\log N/\log\log N)$ and congestion $c = O(\log N/\log\log N)$ where the expected length of the schedule is $\Omega((\log N/\log\log N)^{3/2})$.*

Proof: As in Example 1, the network consists of many copies of a subnetwork. Each subnetwork is constructed so that $d = c = k/\log k$. A subnetwork consists of a linear chain of length d, with loops of length $\sqrt{d}$ between adjacent nodes (see Figure 10). The packets are broken into $\sqrt{d}$ groups numbered 0 through $\sqrt{d} - 1$ of $\sqrt{d}$ packets each. The packets in

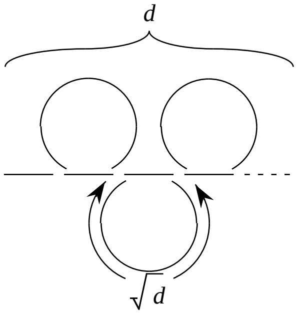

Figure 10: Example 3.

group i use the linear chain for $i\sqrt{d}$ steps and then use $\sqrt{d}-i$ loops as their path. As in the previous example, we assume that queues have unlimited capacity and that at each step a node can send a single packet.

If the random ranks are assigned so that the packets in group i have smaller ranks than the packets in groups with larger numbers, then the packets in group i delay the packets in group $i+1$ by $d-(i+1)\sqrt{d}+i$ steps. Thus the last packet experiences an $\Omega(d\sqrt{d}) = O((k/\log k)^{3/2})$ delay.

Once again the ranks of the packets must have a specific order, which can be shown to happen with high probability given enough copies of the subnetwork. As in Observation 4.1, it is not hard to show this requires $k = \Theta(\log N)$. □

5 Acknowledgements

Thanks to Nick Pippenger and David Shmoys for pointing out the relationship between packet scheduling and job-shop scheduling.

References

[1] N. Alon. A parallel algorithmic version of the Local Lemma. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 586–593, 1991.

[2] J. Beck. An algorithmic approach to the Lovász Local Lemma I. *Random Structures and Algorithms.*

[3] R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg. Work-preserving emulations of fixed-connection networks. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 227–240, May 1989.

[4] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical Report BS-R8909, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, June 1989.

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

[6] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms.* To appear.

[7] T. Leighton, B. Maggs, and S. Rao. Fast algorithms for finding O(congestion + dilation) packet routing schedules. Manuscript in preparation.

[8] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 256–271. IEEE Computer Society Press, October 1988.

[9] A. G. Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 185–194. IEEE Computer Society Press, October 1987.

[10] S. V. Sevast'yanov. Bounding algorithm for routing problem with arbitrary paths and alternate servers. *Kibernetika*, 22(6):74–79, 1986. Translation in Cybernetics 22, pages 773–780.

[11] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. In *Proceedings of the 2nd Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 148–157, January 1991.

[12] J. Spencer. *Ten Lectures on the Probabilistic Method.* SIAM, Philadelphia, PA, 1987.