Distributed Packet Switching in Arbitrary Networks

Yuval Rabani*
The Technion
Haifa 32000, Israel
rabani@cs.technion.ac.il

Éva Tardos†
Cornell University
Ithaca, NY, 14853
eva@cs.cornell.edu

Abstract

In a seminal paper Leighton, Maggs, and Rao consider the packet scheduling problem when a single packet has to traverse each path. They show that there exists a schedule where each packet reaches its destination in $O(C+D)$ steps, where C is the congestion and D is the dilation. The proof relies on the Lovász Local Lemma, and hence is not algorithmic. In a followup paper Leighton and Maggs use an algorithmic version of the Local Lemma due to Beck to give centralized algorithms for the problem. Leighton, Maggs, and Rao also give a distributed randomized algorithm where all packets reach their destinations with high probability in $O(C + D \log n)$ steps. In this paper we develop techniques to guarantee the high probability of delivering packets without resorting to the Lovász Local Lemma. We improve the distributed algorithm for problems with relatively high dilation to $O(C) + (\log^* n)^{O(\log^* n)} D + poly(\log n)$.

We extend the techniques to handle the case of infinite streams of regularly scheduled packets along every path. Here we measure the congestion on an edge e by the sum of the rates of the packet streams that use the edge, denoted by $\lambda(e)$. We require that for some small constant $\epsilon > 0$, for every edge e, $\lambda(e) \leq 1 - \epsilon$. In this case we use the parameter $R = \max_i r_i$, the maximum distance between packets of the same stream, instead of the congestion C above. We notice that $\max\{R, D\}$ is a worst case lower bound on the maximum delay of a packet.

We also extend the results to a model of packet traffic for handling bursty communication. The model is motivated by the new adversarial model suggested by Borodin et al.

*Supported in part by the NSF PYI award of Éva Tardos. Part of this work was performed while visiting the School of OR&IE at Cornell, and while a postdoctoral fellow at the University of Toronto Computer Science Department. Work at the Technion supported in part by the Ruth and David Moskowitz Academic Lecturship award.

†Research supported in part by a Packard Fellowship and an NSF PYI award, by NSF through grant DMS 9505155, and ONR through grant N00014-96-1-0050.

To appear in the Twenty-Eighth ACM Symposium on Theory of Computing, Philadelphia, PA, May 22–24, 1996.

1 Introduction

Packet routing is one of the central issues in large scale parallel computing. Designing efficient packet switching protocols is also one of the main problems in implementing Broadband Integrated Services Digital Network (B-ISDN) standards, such as ATM (Asynchronous Transfer Mode). The packet routing problem is simply that of moving packets in a communication network from their sources to their desired destinations as quickly, as reliably, and using as few resources (such as queues), as possible. A solution to this problem consists of two distinct (though by no means independent) parts: a selection of paths for the packets, and a schedule of the motion of packets along their selected paths. These problems have been extensively studied, mostly in the context of specific network topologies (see [10]).

In this paper we study the issue of scheduling the motion of packets for a given selection of paths in an arbitrary network. We assume a *store and forward* method of routing: The network is modeled as a directed graph, where nodes represent processors or switches, and edges represent communication links. The motion of packets consists of a sequence of synchronized steps. Packets are stored in queues on edges along their respective paths. In each step switches move packets queued at incoming edges to queues on outgoing edges, subject to the constraint that at most one packet may traverse any single edge each step.

Routing individual packets In a seminal paper Leighton, Maggs, and Rao [12] consider the packet scheduling problem when a single packet has to traverse each path. They show that there exists a schedule where each packet reaches its destination in $O(C + D)$ steps, where C is the congestion (the maximum number of paths that use a given edge) and D is the dilation (the length of the longest paths). The schedule uses only constant size queues. Notice that if all packets enter the network at the same time, then $\max\{C, D\}$ is a lower bound on the time it takes to deliver all packets. Therefore, this result is tight up to constant factors. The proof relies on the Lovász Local Lemma [8, 14], and hence is not algorithmic. In a followup paper Leighton and Maggs [11] use an algorithmic version of the Local Lemma due to Beck [5] to give centralized algorithms for the problem.

Leighton, Maggs, and Rao also give a distributed randomized algorithm where all packets reach their destinations with high probability in $O(C+D\log n)$ steps, using $O(\log n)$ size queues. (They call the algorithm on-line, rather than distributed.) For the special case of levelled networks, Leighton, Maggs, Ranade, and Rao [13] give an $O(C+D+\log n)$ steps distributed algorithm. In a distributed algorithm nodes (switches) must make their decisions independently, based on the packets they see, without the use of a centralized scheduler. Leighton et al. [12] state as a challenging open problem to improve their universal distributed algorithm to $O(C+D)$. In this paper we make a step towards resolving this open problem by improving the length of the schedule for the case of large dilation D. We give a distributed algorithm that routes all packets with high probability in $O(C) + (\log^* n)^{O(\log^* n)} D + poly(\log n)$ steps.

The periodic version We also consider the periodic version of the problem where one has to send an infinite stream of packets along every path. The problem can be stated as follows. There are paths P_i in the network with rates r_i. For a path P_i there is a new packet starting at the source s_i every r_i steps. The packet has to follow the path P_i.

This periodic packet scheduling problem is motivated by the ATM and similar standards. At a high level, communication in an ATM network is established by allocating a *virtual circuit* that reserves resources along a path connecting the communicating nodes. Recently, there has been significant progress in the design of algorithms for virtual circuit routing [3, 4], as well as their implementation in practice [9]. However, at a lower level, a virtual circuit is simply a stream of packets carrying encoded video or other transmissions. These packets have to be scheduled in a way that will facilitate timely and reliable communication to all concurrent users of the network.

This paper makes a first attempt of suggesting effective ways of dealing with packet scheduling in ATM networks. We consider the problem of scheduling regular streams of packets along paths in the network. This model of packet traffic is extremely simple, and by no means accurately reflects the situation in ATM networks.

In this periodic setting there are infinitely many packets that must traverse each edge, hence C as defined in the case of scheduling single packets, is not a reasonable measure of congestion. Here we measure the congestion on an edge e by the formula $\lambda(e) = \sum_{P_i \ni e} 1/r_i$, the sum of the rates of the packet streams that use the edge. Obviously, if there is an edge for which the rate is greater than one, no schedule can guarantee delivery of all packets within bounded delay.

We require that for some small constant $\epsilon > 0$, for every edge e, $\lambda(e) \leq 1-\epsilon$. We give a distributed algorithm that routes every packet to its destination with probability $1-p$ in $O(R)+(\log^* p^{-1})^{O(\log^* p^{-1})}D+poly(\log p^{-1})$ steps, where $R = \max_i r_i$ is the maximum distance between packets of the same stream, and we assume that the dilation D is bounded by p^{-1}. The constant hidden by the big-Oh notation depends on the parameter ϵ. Thus, achieving better throughput requires slowing down some packets more. Notice that $\max\{R, D\}$ is a worst case lower bound on the maximum delay of a packet, since as many as R packets from different streams might show up at once, wanting to traverse an edge, without creating congestion more than $1-\epsilon$. Hence, one of these will have to wait R steps before traversing the edge. In this abstract we ignore the issue of limiting queue sizes for simplicity.

Our schedule routes each individual packet to its destination with high probability, but may drop some of the packets. Dropping packets appears to be essential to maintaining simultaneously good response time and near maximum throughput for the network. There are many techniques known to limit the degradation of the quality of the transmission due to limited loss of packets (see, e.g., Albanese *et al* [1]).

Bursty traffic patterns The advantage of the ATM network over the IP networks using Synchronous Transfer Mode (STM) comes out most when packets do not arrive in regular intervals on each communication path, at times there are lots of packets (a burst of packets) and at other times there are very few. In an STM network time units are divided into subsets of size N for a large N, and each path through an edge must reserve some of the N "time-slots" on the edge. The STM network must allocate enough time slots for each path to support the maximum possible burst of traffic, and this leads to many empty time slots, and under utilized networks. In an ATM network allocation of packets to time slots is not done by such reservation, this allows greater utilization of the network, assuming that not too many of the paths would have their traffic burst at the same time (referred to as statistical multiplexing).

We consider a model of packet traffic for handling bursty communication. The model is motivated by the new adversarial model suggested by Borodin et al [6] in the context of studying stability of greedy queueing strategies in various networks. The model we study generalizes their model to allow short periods of overload. A parameter L upper bounds the length of period during which an edge may be overloaded. As in the periodic version, one has to send an infinite stream of packets along every path. Packets may be injected into the stream arbitrarily, subject to the following constraint: There is a (small) constant ϵ such that for any time period of length t which is at least L, for any edge e, the number of packets injected on streams that use e is at most $(1-\epsilon)t$.

Our results for this model parallel those for the periodic model. We give a distributed algorithm that routes every packet to its destination with probability $1-p$ in $O(L) + (\log^* p^{-1})^{O(\log^* p^{-1})}D + poly(\log p^{-1})$ steps, assuming $D \leq p^{-1}$. Notice that $\Omega(L)$ is a worst case lower bound on the maximum delay, because as many as $(1-\epsilon)L$ packets may be injected simultaneously along a single edge without violating our assumptions.

Tradeoffs Summarizing our results for scheduling infinite packet streams, we consider four parameters that define the packet traffic: the network throughput $1-\epsilon$, the burstiness as characterized by the longest overload period L, the packet loss probability p, and the maximum delay of a packet. We give algorithms that provide tradeoffs between the first three parameters and the delay. Such tradeoffs seem intuitively necessary in a distributed setting.

Our techniques The main idea of the algorithm follows the lines of the papers of Leighton, Maggs, Ranade, and Rao [12, 13, 11]. In the simple randomized algorithm of [12] each path selects a delay of at most $O(C/\log n)$ uniformly and independently at random. The main idea of the distributed algorithm is to show (using Chernoff bounds) that with high probability in the resulting schedule no more than $O(\log n)$ packets traverse any edge at a given time. Now a valid schedule is obtained by slowing down this schedule by a factor of $O(\log n)$. The resulting schedule takes $D \cdot O(\log n)$ time to traverse a path of length D, so with the initial $O(C)$ delay, the length of the resulting schedule is $O(C + D\log n)$.

The centralized algorithm of [12, 11] uses the idea of inserting an initial random delay iteratively. After the initial random delay they show that not only are there no more than $O(\log n)$ packets traversing any edge at any given time, but also the number of packets traversing an edge in any $\log n$ *time period* is at most $O(\log n)$ with high probability. Now they divide each path into $O(\log n)$ long subpaths, and add delays in front of each of the subpaths. In order to argue that these extra delays decrease the congestion to $O(\log\log n)$, they need to use the Lovász Local Lemma [12, 11]. Chernoff bounds used in the analysis of the initial larger delays would not yield a strong enough guarantee. Rather than showing the desired high probability bound, Chernoff bounds would imply that the probability of an edge getting congested is at most $1/poly(\log n)$. However, this would imply that among the m edges of the network many do get congested.

In this paper we develop techniques to guarantee the high probability of delivering packets without resorting to the Lovász Local Lemma. Instead of the Lovász Local Lemma we use a simple version of Chernoff bounds for events with limited correlation. We introduce the idea of a special "catchup" time. In our schedule we reserve some special time for packets that are "far" behind where they are "supposed to be" in the schedule. This extra reserved time allows the delayed packets to catch up and continue on their paths, where they were "supposed to be."

2 Routing of single packets

First we review a centralized version of our algorithm. We develop an algorithm that schedules all packets with high probability (w.h.p.), *i.e.,* the probability of the schedule failing can be bounded by any inverse polynomial in n.

Following the general outline of Leighton, Maggs, and Rao [12] we consider schedules of packets that might not be feasible, *i.e.*, there might be more than a single packet traversing an edge at a time step. The *congestion* of the schedule is the maximum number of packets that traverse an edge at the same time. Notice that if the congestion of the schedule is bounded by λ then we can obtain a feasible schedule by expanding every time step into λ steps.

A *T-frame* is a sequence of T consecutive time steps. The *congestion of an edge in a T-frame* is the number of packets that traverse the edge during the T-frame. The *congestion of a T-frame* is the maximum congestion of an edge in this frame. Leighton, Maggs and Rao [12] obtain the $O(C + D)$ long schedule by repeated refinements. Given a schedule where the congestion of any T-frame is at most λT, for a small constant λ, they refine it to a schedule partitioned into T'-frames, where $T' \approx \log T$, so that the congestion of any T'-frame is at most $\lambda' T'$, for a constant λ' not much bigger than λ. When T, the size of the frames, is a constant the congestion of the frame implies that the congestion of any edge in any time step is at most λT, so a good schedule can be obtained by expanding every time step into λT steps.

Rather than refining the schedule iteratively, we have to refine it recursively. If $C > \log n$ then we start similarly to the first step of the distributed algorithm of Leighton, Maggs and Rao [12] to find a schedule with $T \approx \log n$. The main part of the algorithm is a recursive procedure. The procedure assumes we have a parameter $T \le O(\log n)$, and a non-feasible schedule divided into T-frames. We assume that the congestion of each T-frame is at most $\gamma^{\log^* T} T$. The frames define path segments: a path segment defined by frame f is the portion of a packet's path that is traversed during f in the input schedule. We schedule the path segments of each set of some consecutive $\bar{T} = poly(T)$ T-frames in a somewhat expanded interval of length $\gamma^{O(\log^* T)}\bar{T} \cdot T$ so that each path segment is scheduled with probability at least $1 - 1/exp(T)$. To get the best bounds we will maintain a schedule with a relatively high congestion: we use $\gamma = \log^* n$ to get the results claimed in the abstract.

The general overview of the recursive procedure is as follows. The procedure schedules consecutive T^6-frames independently. Each T^6-frame is scheduled in two phases. In the first phase, the schedule is *refined* and a *recursive* call is invoked with frame size $T' = O(\log T)$. Some packets may fail either in the refinement or in the recursive call. These get scheduled in the second phase (the *catchup track*), which also involves recursive calls. Some packets may fail both phases. These are reported to the calling procedure as dumped packets.

We will use the first step of the distributed algorithm of Leighton, Maggs and Rao [12] to create a schedule divided into $T \approx \log n$ size frames, so that all frames have congestion at most ν for a parameter ν that will be defined later. We introduce random initial delays $\in_U [0, C/\nu(1+\epsilon)]$ for every packet for some $\epsilon > 0$, where $\in_U$ denotes the fact that the random delays should be selected uniformly and independently. The next lemma states that this results in a schedule with the desired property with high probability. The proof uses Chernoff bounds and is exactly analogous to the proof of the distributed Leighton-Maggs-Rao [12] algorithm.

Lemma 1 *For any constant $\nu > 1$ and any constant $c > 0$ there is a constant α so that with probability at least $1 - n^{-c}$, we obtain a schedule that can be divided into T-frames with $T = \alpha \log n$, so that the congestion of any T-frame is at most νT.*

We call the recursive procedure with the resulting schedule. The procedure will schedule each packet with high probability as $exp(T) \geq poly(n)$ due to the choice of T. Notice that we need to start off with a frame size of $T \approx \log n$ even if both C and D are small, as otherwise the packets would not be routed with high enough probability. In the next section we will discuss a version of the algorithm where we do not insist that each packet is routed with high probability. In that version we can start with frame size below $\log n$ when C and D are both small.

2.1 An $O(C + D \log\log n + poly(\log n))$ Algorithm

To make the description cleaner we first describe the version of the algorithm that routes packets in $O(C + D \log\log n + poly(\log n))$ steps. This procedure gives the essence of our algorithm. The improved schedule is obtained by using the ideas in this simpler schedule recursively.

We start with using Lemma 1 with $\nu = \log\log n$ to create the initial schedule. Consider the schedule given by the lemma divided into T-frames for some $T = \alpha \log n$. Consider the schedule as a set of T^6-frames, by grouping T^5 consecutive T-frames into a T^6-frame. The algorithm will handle each T^6-frame independently. It will schedule each T^6-frame in $O(\log\log n)T^6$ steps. This results in a schedule of total length $O(C + D \log\log n + T^6 \log\log n)$ as desired, as the number of T-frames is bounded by $O(D/T + C/(T\nu))$ by having paths of length D and initial delays of at most $O(C/\nu)$.

We now describe more precisely how the procedure handles a T^6-frame F. Consider a T-frame f of F. We attempt to schedule the path segments defined by f in an interval of length $O(T \log T)$. First, we insert an initial random delay $\in_U [0, T]$ in front of every segment. The expected congestion of an edge at a time step is bounded by $\log T$, we would like to argue that with high probability the congestion of the edges is at most $O(\log T)$, and then we could expand the schedule by a factor of $O(\log T) = O(\log\log n)$ to obtain a feasible schedule as desired. However, the probability of an edge having congestion more than $\log T$ is not small enough. We use an analogue of Lemma 1 to bound this probability. We state the lemma here in a bit more general form so we can refer to it later.

Consider a T-frame of a schedule for some T, assume that the congestion of the frame is bounded by some $C > T$ and let $C' < C$ and $\epsilon > 0$ constants so that $C'(1 + \epsilon) \leq C$. Refine the the T-frame by introducing random initial delays $\in_U [0, T\nu(1+\epsilon)]$ for every segment in the frame, where $\nu = C/C'$. We will say that an edge e is *blocked* in f' iff the congestion on e during f' is more than CT'/T.

We show using Chernoff bounds that the probability of an edge being blocked is small, bounded by inverse polynomial in T. This probability is not small enough to guarantee that none of the m edges gets blocked. Instead of arguing about the congestion of the whole T-frame, we want to argue about the success of individual path segments, as there are at most T edges involved in a path segment, and hence with good probability none of those T edges fail. A path segment p is *successful* during f iff it does not cross a blocked edge in any T'-frame. Otherwise, the path segment *fails*. Analogously to Lemma 1 we can prove the following:

Lemma 2 *For any constant $\epsilon > 0$ and any constant $c > 0$ there is a constant α so that if we consider T'-frames with $T' = \alpha \log T$, then the probability that a path segment defined by a T-frame fails is bounded by T^{-c}.*

Now consider the T^6 frame F. We do this for all T-frames f within F. We say that a path segment succeeds in F iff it succeeds if each of the T^5 T-frames. From Lemma 2 we get that the probability of a path segment succeeding can be bounded by an inverse polynomial in T.

However, we would like each path segment to succeed with probability of inverse exponential in T. This is where we will have to introduce the idea of "catchup" to help packets catch up with the schedule after they fail. The key feature of the "catchup" track is that the congestion there is much lower with sufficiently high probability, as the following analysis shows.

First we bound the number of failed path segments that cross any particular edge e. Notice that this number includes all segments that were supposed to cross e, and failed, including segments that failed due to some other edge e'. A complication in proving a bound is that the failure of different path segments through edge e is highly correlated.

The following simple lemma extends the Chernoff bounds to correlated events, correlated like the situation handled by the Lovász Local Lemma. Let X_1, X_2, …, X_N be 0-1 random variables, consider the *dependency graph* with the events as nodes, so that an event is independent of all events not adjacent to the node.

Lemma 3 *Assume that the maximum degree in the dependency graph of the random 0-1 variables X_1, X_2, …, X_N is at most k, and suppose that for all i, $E[X_i] \leq p$. Let $\delta \geq 1$. Then, $\Pr[\sum X_i > 4e\delta pN] < 4ek2^{-\delta pN/k}$.*

Proof. By Brooks' theorem [7] the dependency graph can be partitioned into at most $k+1$ independent sets, and therefore into $m \leq 4ek$ independent sets, each of size at most $N/2ek$. Let the set sizes be N_1, N_2, …, N_m. Let S_i denote the number of variables in set i that are 1. Set $\beta_i = \delta N/kN_i \geq 2e$. Using Chernoff bounds (see [2, Appendix A]), we get

$$\Pr[S_i > \beta_i p N_i] < \left(\frac{e}{\beta_i}\right)^{\beta_i p N_i}.$$

Summing over the sets, and using $\beta_i N_i = \delta N/k$, gives the lemma. □

Lemma 4 *Assume we are given a schedule divided into T-frames, so that the congestion of each T-frame is bounded by C, and $C \leq O(T \log T)$. For any constant $c > 0$ if T is sufficiently large, then the probability that more than T^5 path segments that cross e during F fail* *can be bounded by $exp(-cT)$.*

Proof. There are at most CT^5 path segments that want to cross e during F. Let $\textsc{Fail}(i,j)$ denote the event that path segment i failed in T-frame j. The total number of such events is CT^{10}. $\textsc{Fail}(i,j)$ is totally independent of any $\textsc{Fail}(i',j')$, if $j' \neq j$.

Now consider a T-frame j. First observe that the blocking of two edges are correlated iff they share a path. There are at most C paths though an edge and each path goes through at most T edges, so this implies that the blocking of an edge is correlated with at most CT other edges. Now two events $\textsc{Fail}(i,j)$ and $\textsc{Fail}(i',j)$ in a T-frame j are correlated iff there are two edges, one for each path, whose being blocked are dependent. A path goes through T edges, each of these edge is correlated with CT other edges, and each of these other edges carries at most C other paths. This gives a total of C^2T^2 paths whose failure correlates with the failure of a particular path. So, the dependency graph over the events $\textsc{Fail}(i,j)$ events has maximum degree C^2T^2.

We apply Lemma 3 to our setting. In our case, $N = CT^{10}$, the failure probability p can be made $1/T^{-c}$ for any constant c, and $k = C^2T^2$. We argue that the probability that more than $b = T^5$ events $\textsc{Fail}(i,j)$ occur is exponentially small in T (assuming T is large enough). The number of such events is clearly an upper bound on the number of failed path segments that cross an edge (we are over counting, since b bounds the number of times paths fail, counting multiplicities). □

So now we are ready to continue the schedule in each T^6-frame F. Successful path segments wait till the end of the frame. The rest of the scheduling process within F handles failed segments. An edge is *bad* iff more than $b = T^5$ failed path segments cross it. In the case we are considering now, when $T \approx \log n$ there are no bad edges with high probability. In the general case failed segments that cross a bad edge will be *dumped*, they will not participate in the refined schedule.

We schedule the failed segments as follows. Partition F into T consecutive T^5-frames. This partitions the failed path segments into T parts. We schedule these parts frame by frame by inserting random delays $\in_U [0, T^5]$. In this schedule blocked edges, and failed segments are defined similar to the above using $\nu = O(1)$ (only for the longer T^5-frames). If a path segment in a T^5-frame succeeds, the packet moves on to the next T^5-frame, trying to schedule the next path segment. If a path segment fails, it is moved to the next T^5-frame, where it tries again.

Notice that the successes of the packets at different iterations are dependent. To simplify the analysis consider the experiment when at all iterations we try *all* of the T different T^5-segments of all paths under consideration whether we need that segment or not. This way the success of the segments at different iterations are independent, and hence this analysis provides a bound on the success probability that is independent of other events.

Notice that since the total number of failed path segments that cross any particular edge is at most T^5, in any attempt to schedule T^5-frame path segments less than T^5 paths cross any particular edge. We bound the failure probability of a path segment:

Lemma 5 *Let p be a path segment attempted to be scheduled in an iteration of the above scheduling process. For any constant c the probability of its failing can be made to be at most T^{-c} independent of all other events.*

Proof. The proof is similar to the proof of Lemma 2. □

Once a packet succeeds T times (i.e., T consecutive T^5-frame path segments of this packet have been scheduled), it stops and waits till the end of the schedule for F. Its schedule in this frame has been completed. The total number of times we try to schedule these path segments is $2T$.

Lemma 6 *If T is sufficiently large, then the probability that a packet will succeed less than T times after $2T$ tries is exponentially small in T.*

Proof. Consider a failed path of length T^6. Each iteration a new T^5 segment of this path gets scheduled with probability at least $1-1/poly(T)$, independent of events in previous iterations due to Lemma 5. So the expected number of successes in $2T$ tries is close to $2T$. Therefore, using Chernoff bounds, we see that the probability that there will be less than T successes is exponentially low in T. □

In the primary case we consider now, when $T \approx \log n$, then path segments in F do not fail with high probability. In the general case (discussed below), the paths that fail this step are also dumped, together with those that cross bad edges. We now study the distribution of dumped path segments.

Lemma 7 *If T is sufficiently large, then the probability that a path segment p in F is dumped can be bounded by an inverse exponential function in T.*

Proof. By Lemma 4, the probability that an edge is bad is exponentially small, and a path segment in F crosses at most T^6 edges. By Lemma 6, a path segment that does not cross any bad edge has also a probability exponentially small in T to be dumped. So the total probability of an edge being dumped is also exponentially small in T. □

Putting together what we have so far: we use the Lemma 1 to get an initial schedule with frame sizes $T \approx \log n$, and congestion for each frame is bounded by $T \log\log n$. Then we refine this schedule, again by introducing local delays at random. The probability of failing at this stage is polynomial in $T \approx \log n$, hence over an n-node graph many edges fail. Next we schedule the failed segments using the catchup track. The catchup track small total congestion, so there we can afford to repeatedly schedule failed segments till they succeed. This boosts up the probability of failure from inverse polynomial in T to inverse exponential in T. With $T \approx \log n$ this implies that with high probability no segments fail, and we have a schedule for each T^6-frame of length $O(T^6)$ with congestion of any $\log\log n$ frame at most $\approx \log\log n$. We get the schedule of length

$O(C+D\log\log n+poly(\log n))$ by expanding every step of this schedule to $\log\log n$ steps.

In order to analyze the recursive version of the algorithm, we will need to analyze the probability that too many path get dumped by a phase. To do so we need to bound the dependency among the events that path segments get dumped. We first give such an analysis ignoring the effect of the recursive calls. Lemma 10 will later use the following lemma in the full analysis of dependencies.

Lemma 8 *The event that a path segment in F is dumped is independent of all but at most $T^{O(1)}$ similar events.*

Proof Sketch. Observe that the events that two edges e and e' are blocked in a T-frame j is correlated iff there is a path segment through both e and e'. The events that two path segments i and i' fail in a T-frame are correlated if there are two edges, one for each path, whose being blocked are dependent. Consider the graph $G(F)$ whose nodes are the path segment in a T^6-frame. We see that if the events that two path segments i and i' fail in a T-frame are correlated, then the corresponding two nodes are at most 2 apart in $G(F)$. Similarly, it is not hard to show that the events that two path segments i and i' get dumped in a T^6-frame can only be correlated if the distance between the corresponding two nodes in $G(F)$ is at most 4. The lemma follows (as was done in Lemma 4) by counting the number of path segments that can satisfy this condition. $\square$

2.2 The Recursive Algorithm

Next we consider the recursive version of the algorithm. Recall from our initial discussion that the recursive version depends on a parameter $\gamma > 1$. We will give a schedule of length $(1+(1/\gamma))^{O(\log^* n)}C + \gamma^{O(\log^* n)}D + poly(\log n)$.

The initial step of the algorithm is to use Lemma 1 to create a schedule (with high probability) divided into T-frames with $T \approx \log n$ so that the congestion of each frame is bounded by $C = \gamma^{\log^* T}T$. We use the recursive procedure on the resulting schedule. The recursive procedure is analogous to our algorithm creating the $O(C+D\log\log n+poly(\log n))$ long schedule. The main differences are that:

1. We no longer assume that $T \approx \log n$. As a result there will be dumped path segments. Each path segment will be dumped with probability inverse exponential in T. Dumped path segments do not participate in the final schedule.
2. We assume that the input to the procedure is a schedule divided into T-frames so that the congestion of each frame is bounded by $C = \gamma^{O(\log^* T)}T$. The procedure schedules sets of consecutive T^5 T-frames independently, one T^6-frame is schedules in an interval of length $(1+(1/\gamma))^{O(\log^* T)}C + \gamma^{O(\log^* T)}D + poly(\log T)$, and the probability that a path segment is dumped from the schedule is bounded by an inverse exponential in T.
3. The refine steps using Lemma 2 use the parameter $\nu = \gamma$ rather than $\nu = \log\log n$.
4. When applying a step of our algorithm to a T-frame, we prepare input for our recursive procedure with frame size $T' = O(\log T)$ using refinement.
5. The catchup track handles packets dumped by the recursive calls as well as failed packets.

The input to the recursive version of the procedure consists of a schedule of the path divided into T-frames, so that the congestion of a T-frame is at most $\gamma^{O(\log^* T)}T$. The algorithm uses the refinement of Lemma 2 with parameter γ to refine the scheduled in each T-frame. Path segments that *fail* in refinement move to the catchup track. We invoke a recursive call with frame size T' with the successful path segments as input. The recursive call refines the schedule in each of the $(T')^6$-frames. (We assume that T is big enough so that $(T')^6 < T$.) Some path segments might get dumped by the recursive call, for such dumped segments the T^6-frame path containing the segment is moved to the catchup track. Now consider the catchup tracks. An edge is *bad* if more than T^5 paths through it were moved to the catchup track, and path segments through bad edges are *dumped*. In the catchup tracks for in each T^5-frame in turn, we prepare input for the recursive procedure with frame size $T' = O(\log T)$ using refinement, and invoke a recursive call. A path segments fails this step if it either failed the refinement, or got dumped by the recursive call. In the next T^5 frame we consider the next segment of successful paths, along with the failed segments of the failed paths. Path segments that do not get at T successfully scheduled path segments get dumped by the call.

We have to consider two issues in analyzing the resulting procedure (1) the length of the schedule created, and (2) the probability that a path gets scheduled. We start our analysis with (1).

Lemma 9 *One T^6-frame is scheduled by the recursive procedure in an interval of length $\gamma^{O(\log^* T)}T^6$.*

Proof. We prove the lemma by induction. If T is small then the lemma obviously holds, otherwise, we create a schedule by using refine, the catchup tracks, and recursive calls. The first step is to introduce a delay that is at most γT for each segment of a T-frame. This initial delay increase the length of each T-frame by a factor of $\gamma+1$, then the recursive calls are applied to smaller frames of size $T' \approx \log T$. By the induction hypothesis we schedule the segments of a T-frame that do not fail in T-frame in an interval of length $\gamma^{O(\log^* T')}T$. Similar analysis applies in the catchup track that consists of $2T$ frames of size T^5 each. So the final schedule will be of length $3(\gamma+1)\cdot\gamma^{O(\log^* T')}T$. Now the lemma follows by observing that $T' \approx \log T$, so $\log^* T' = \log^* T - c$ for some constant $1 > c > 0$ assuming that T is sufficiently large. $\square$

Before we can analyze the probabilities of being dumped we need to bound the amount of dependency among the events that paths are dumped. Consider a call to the recursive procedure with frame size T, let i denote a path segment in a T^6-frame of the input schedule, and let $\textsc{Dump}(i)$ denote the event that the path segment is dumped by the procedure call.

Lemma 10 *Let i be a path segment in a T^6-frame F handled by the recursive procedure called with frame size T. Then,* DUMP(i) *is independent of all but at most $T^{O(\log^* T)}$ similar events.*

Proof. We would like to prove the lemma by induction, however, the task of bounding the number of dependencies is complicated by the fact that different choices for delays in the T-frames cause different dependencies due to recursive calls. Instead of proving by induction the bound on the *number* of correlated DUMP(j) events, we prove by induction the structure of what other paths are correlated with paths i.

Recall the correlation analysis in the proof of Lemma 8. Consider the graph $G(F)$ whose nodes are the path segments in a T^6-frame, two of them connected by an edge if they share a network edge. Lemma 8 shows that in one level of recursion we get that DUMP(i) events for two path segments in the T^6-frame are correlated in the non-recursive version of the algorithm only if they are of distance at most 4 in this graph. More generally we can show by induction that two DUMP(i) and DUMP(i') events for two path segments i and i' can be correlated by the recursive procedure only if their distance in the graph $G(F)$ is at most 4^k, where k is the depth of the recursion.

The lemma follows by observing that the depth of the recursion is $O(\log^* n)$, and counting the number of path segments that can satisfy the above condition. □

We can now analyze the distribution of the number of dumped path segments. Path segments get dumped by two events (1) an edge having too many paths through failing in the first phase of the schedule, and (2) in the catchup track the path segments repeatedly fails to be scheduled.

Lemma 11 *Let F be a T^6-frame and let e be an edge. If T is sufficiently large, then the probability that a path segment p in F is dumped can be bonded by an inverse exponential function in T.*

Proof. To prove the lemma we have to go through the same analysis as was done in the non-recursive case, and prove the analogues of Lemmas 2, 4, 5, 6.

Let us consider the analogue of Lemma 2. We need to show that the probability that a path segment in a T-frame fails can be bounded by an inverse polynomial in T. A path segment can fail either by failing in the refinement step, or by being dumped by the recursive call. Lemma 2 bounds the probability of the former, and the induction hypothesis bounds the probability of the later event.

Now the analogue of Lemma 4 follows from the above analogue of Lemma 2; Lemma 10 bounding the degrees in the dependence graph of the FAIL events; and Lemma 3.

The analogue of Lemma 5 follows along the same lines as the analogue of Lemma 2. Finally, the analogue of Lemma 6 follows from the previous ones.

A path segment i that fails the first phase might be dumped for two reasons, either because it passes through a bad edge in the catchup track, or it fails to be scheduled successfully in the catchup track sufficiently often. The previous arguments show that both events have probability exponentially small in T. □

Lemmas 9,11 together with the initial scheduling based on Lemma 1 imply Theorem 12.

Theorem 12 *For any constant $\gamma > 1$, and any packet routing problem with congestion C and dilation D we can construct a schedule in which all packets reach their destinations with high probability in $(1+1/\gamma)^{O(\log^* n)}C + \gamma^{O(\log^* n)}D + poly(\log n)$ steps.*

When $C \approx D$ then we get the best schedule by using a constant γ. The version of the theorem claimed in the abstract can be obtained by using $\gamma = \alpha \log^* n$ for some constant α. In fact, we can push the constant in front of the congestion C arbitrarily close to 1 at the expense of making the constant multiplying the dilation D somewhat worse. This version of the result will be used in the next section.

We remark that a similar variant of the Leighton, Maggs, Rao [12, 11] schedule can be obtained: for any constant $\epsilon > 0$ there is a schedule for the periodic packet routing problem where all packets get delivered to their destination in $(1+\epsilon)C + O(D + \log n)$, where the constant hidden in the big-Oh depends on ϵ.

3 The Periodic Packet Routing Problem

In this section we consider the periodic version of the problem where one has to send an infinite stream of regularly arriving packets along every path. The problem can be stated as follows. A *stream i* of packets is defined by a paths P_i and an integer rate r_i. In the schedule a new packet starting at the source s_i of the path P_i appears every r_i steps. The packet has to follow the path P_i. The problem is to deliver each of these packets to their corresponding sinks in a small number of steps.

In this periodic setting there are infinitely many packets that must traverse each edge, hence, C as defined in the case of scheduling single packets, is not a reasonable measure of congestion. Here we measure the congestion on an edge e by the formula $\lambda(e) = \sum_{E \in P_i} 1/r_i$, the sum of the rates of the packet streams that use the edge. Obviously, if there is an edge for which the rate is greater than one, no schedule can guarantee delivery of all packets within bounded delay.

A worst case lower bound on the maximum delay of a packet is obtained by $\max\{R, D\}$, where D is the maximum dilation, and $R = \max_i r_i$ is the maximum rate. To see why, notice that as many as R packets from different streams might show up at once, wanting to traverse an edge, without creating congestion more than 1. Hence, one of these will have to wait R steps before traversing the edge.

The main result of this section uses a variant of the schedule developed in Section 2 to obtain a schedule with analogous delivery times assuming that the maximum congestion is at most $1 - \epsilon$ for a constant ϵ.

Before discussing different scheduling algorithms we need the following lemma. In essence the lemma shows that it is no loss of generality to assume that all packets have the same rate R.

Lemma 13 *Given positive constants $\epsilon > \epsilon'$, and a periodic scheduling problem with congestion at most $1 -$*

ϵ, and the maximum rate R, there is another periodic scheduling problem with maximum congestion at most $1-\epsilon'$ where all streams have common rate $R' = O(R)$, such that any schedule for the new problem provides a schedule for the old problem, where each packet is delivered in at most R' more steps than the corresponding packet in the new problem.

Proof. We will replace a stream with rate r_i along paths P_i with $n_i = \lceil R'/r_i \rceil$ different streams each of rate R' along paths P_i. We choose R' so that the congestion of the resulting schedule is at most $1-\epsilon'$. The congestion of the new schedule is

$$\sum_{i:e\in P_i} n_i/R' \leq \sum_{i:e\in P_i} (1/r_i + 1/R') \leq (1-\epsilon) + R/R',$$

where the last inequality follows from the fact that with maximum rate R there can be at most R different paths through the edge e. Now if $R' > (\epsilon - \epsilon')^{-1}R$ then we get that $\sum_{i:e\in P_i} n_i \leq (1-\epsilon')R'$ as desired. □

Before we get to the main result, we give a simpler initial schedule based on the on-line algorithm of Leighton, Maggs, and Rao.

Theorem 14 *If the congestion is bounded by $1-\epsilon$ for a constant ϵ, and the maximum rate is R, then there exists a schedule in which all packets along paths of length d_i are routed to their destination in $O(R + d_i \log n)$ steps.*

Proof. Using Lemma 13 we see that it is no loss of generality to assume that all streams have the same rate R. We can further assume that $R \geq \alpha \log n$, for any constant α (otherwise, replace each stream by several streams with the desired rate). The schedule will be periodic with period R. In the schedule of an edge there will be a time slot allocated to each path through the edge every R steps. The edge will forward a packet from the scheduled path along the edge if there is one in the queue of the edge. We will carefully coordinate the slots assigned to a stream so that after an initial delay of at most R steps, the packet will pass a new edge every $\log n$ steps.

To create the schedule select a value k so that $R/k \approx \alpha \log n$ where α is a large constant (dependent on ϵ), and divide the time period of $[1..R]$ into k roughly equal size slots of consecutive time steps. We will refer to these sets of consecutive time steps as *buckets*. We first give an initial assignment of each edge on a stream's path to buckets. Each packet will traverse exactly one edge during a bucket until it reaches its destination. Each stream selects an initial delay $\in_U [1..R]$, and starting at the that time reserves a step in the corresponding time bucket for its first edge, a step in the next bucket in the next edge, etc. Notice that time is understood `mod R` for now.

We argue using Chernoff bounds that for any edge and bucket the number of streams that have a reserved time slot in the bucket of the edge is at most R/k, the size of the bucket. Consider an edge e and a bucket j. Each path using this edge has a probability of $1/k$ of being assigned to this bucket, so the expected number of streams assigned buckets is at most $(1-\epsilon)R/k$, and then the choice of $R/k \approx a \log n$ and Chernoff bounds imply that with high probability there are no more than R/k assigned streams.

Now we schedule the reserved time slots in each bucket arbitrarily. In the resulting schedule a packet might have to wait up to R steps in the source queue due to the initial random delay, but then it waits at most $R/k \approx \alpha \log n$ steps between traveling on adjacent edges on its path, so the packet will be delivered in $R + d_i \log n$ steps. □

The main theorem of this section is obtained by a construction that is similar to Theorem 14, but it uses a variant of the algorithm of Theorem 12. The idea is that the (non-periodic) packet scheduling algorithm developed in Section 2, that delivers all packets to their destination in $(1+\delta)C + g(n)D + f(n)$ steps (where C is the congestion, and D the maximum dilation, as in Section 2), can be used to design a periodic schedule that delivers all packets to their destination in $O(R + g(n)D + f(n))$ steps, assuming that the maximum congestion is bounded by a constant smaller than $1/(1+\delta)$.

Theorem 15 *Assuming there is a scheduling algorithm that delivers all packets to their destination in $(1+\delta)C + g(n)D + f(n)$ steps for some constant $\delta > 0$, then there is a periodic scheduling algorithm that delivers all packets to their destination in $O(R + g(n)D + f(n))$ steps assuming that the congestion is bounded by a constant smaller than $1/(1+\delta)$ and $f(n) \geq \log n$.*

Proof. Using Lemma 13 we may assume without loss of generality that all streams have the same rate R, and the congestion of the schedule is bounded by $(1-\delta')$ for a constant δ', so that $(1-\delta')(1+\delta) < 1$. We may also assume that $R \geq \alpha g(n)$ for any constant α, using the same argument as in the proof of Theorem 14.

As in the proof of Theorem 14 the schedule will be periodic with period R. We divide the interval of time $[1..R]$ into k approximately equal size buckets for some k. This time we make the size of the bucket $R/k \approx \alpha g(n)$, where α is a large constant (dependent on δ and the constant in the big-Oh of the assumed single packet scheduling algorithm).

We first give an initial assignment of each edge on a stream's path to buckets. As in the schedule of Theorem 14 each stream picks a delay in $[1..R]$ uniformly at random, and reserves adjacent buckets for adjacent edges of it path (as before time is understood `mod R` for now). We will consider T-frames of buckets in this initial schedule, consisting of $T = f(n)/g(n)$ consecutive buckets.

Similar to the proof of Theorem 14 we argue using Chernoff bounds that for any edge and T-frame of buckets the number of streams that have a reserved time slot in one of the T consecutive buckets of the edge is at most $(1-\delta'')\alpha f(n)$, for some constant $\delta'' > 0$ so that $(1+\delta)(1-\delta'') < 1$.

Now we schedule the reserved time slots in each bucket using the assumed scheduling algorithm for each T-frame. We consider the scheduling problem defined by the path segments of a T-frame of buckets. The maximum dilation (length of a path) is at most $D = f(n)/g(n)$ by construction, and the maximum congestion is at most $C = (1-\delta'')\alpha f(n)$ by the Chernoff bound argument above. Therefore the assumed algorithm finds a schedule that delivers all packets to the end of their path segment in $(1+\delta)C + g(n)D + f(n)$

steps, which is $(1+\delta)(1-\delta'')\alpha f(n) + 2f(n) \leq \alpha f(n)$, where the inequality can be ensured by the choice of α. Notice that $\alpha f(n)$ is exactly the number of time slots in the T-frame of buckets, so this schedule can be completed within the T consecutive buckets.

The resulting schedules of the time buckets are combined into a single cyclic schedule for the problem. Each packet follows the schedule of frame i, and waits till the beginning of the next frame before continuing along the rest of its path. In the resulting schedule each packet might wait up to $O(R)$ steps in its initial queue at the source, due to the initial random delay (of k buckets $\approx R$ time steps), and then succeeds in making $f(n)/g(n)$ steps in each time frame of size $af(n)$, so the packet is delivered within $O(R+g(n)d_i+f(n))$ time steps. □

Corollary 16 *For any periodic packet scheduling problem with maximum congestion bounded by $1-\epsilon$ for some constant $\epsilon > 0$, and maximum rate R, there is a schedule that delivers all packets to their destination in $O(R+D+\log n)$ steps, and the scheduling algorithm of Section 2 can be used to find a schedule delivers all packets to their destination in $O(R) + (\log^* n)^{O(\log^* n)}D + poly(\log n)$ steps.*

Notice that when R and D are both small, then the additive $poly(\log n)$ term dominates the length of the schedule. For this case we can do better by dropping some packets.

First we need to prove a lossy version of Theorem 12. Recall the outline of the schedule of Theorem 12: we use Lemma 1 to get an initial schedule with frame sizes $T \approx \log n$, and then we use the the recursive algorithm of Section 2.2. The first phase uses Lemma 1 and no catchup tracks. Packets in this phase have a failure probability of $1/exp(T)$. In the resursive algorithm packets have a failure probability of $1/exp(T)$ in each frame of size T, for the *initial frame size* T. However, for $T \approx \log n$, the $1/exp(T)$ probability is small enough, and using catchup tracks at this point would cause large extra delays.

Let p denote the acceptable probability of packet loss and assume that $D \leq p^{-1}$. The lossy version of the algorithm starts with a similar special first phase, not using catchup tracks. This first phase generates a schedule with frames of size $T \approx logp^{-1}$. The probability of loosing a packet due to congestion at this phase is bounded by $1/exp(T) = 1/poly(p^{-1})$. After this initial phase we use the recursive algorithm of Section 2 with parameter $\gamma \approx \log^* p^{-1}$ with frames of size T. The recursive algorithm schedules each packet with probability $1 - 1/exp(T)$.

We use Theorem 15 to convert this algorithm for scheduling single packets to an algorithm for scheduling streams of packets with low probability packet loss.

Theorem 17 *For any periodic packet scheduling problem with maximum congestion bounded by $1-\epsilon$ for some constant $\epsilon > 0$, maximum rate R and maximum paths length D, and a probability p such that $D \leq p^{-1}$ there is a schedule that delivers each packet to its destination with probability $1-p$ in $O(R) + (\log^* p^{-1})^{O(\log^* p^{-1})}D + poly(\log p^{-1})$ steps.*

One drawback of this algorithm is that any individual stream is not scheduled at all with probability p. A possible solution to this problem is to repeat the random choices every now and then. If this is done frequently enough, any individual stream might lose a small segment with low probability. The trouble is that packets scheduled according to the new random choices might conflict with previous packets scheduled according to the old random choices. This requires flushing the network (dropping all packets) whenever the schedule is determined anew. Thus, it can be done only after a large (constant) number of cycles had passed. Some of the residual capacity ϵ can be used for the flushing, or else the amount of packet loss is somewhat increased. The details will appear in the full version of the paper.

4 Bursty Traffic

In this section we sketch how our methods can be used to handle irregular traffic patterns. We say that the packet traffic is L-bursty with congestion $1-\epsilon$ iff for any time period of length t which is at least L, for any edge e, the number of packets injected on streams that use e is at most $(1-\epsilon)t$.

To illustrate our results, set $L' = \max\{L, D\epsilon^{-1}\alpha \log n\}$, for some constant α. Consider the following simple algorithm. Insert a delay $\in_U [0, L']$ for each packet. Consider the following schedule. Partition time into consecutive frames of length $\alpha \log n$. A packet makes a move on the first available slot in the frame following its arrival. Similar to previous arguments, the probability that too many packets have to move in any particular $\alpha \log n$ frame is at most n^{-c} for some constant c which depends on α. The reason why this is true is that the set of packets which may fall into a particular frame is exactly the set of packets injected in some time interval of length $(1+\epsilon)L'$ (these are the only packets that could have made it to the relevant edge in time, according to the relation between L' and D), and there are at most $(1-\epsilon)(1+\epsilon)L' = (1-\epsilon^2)L'$ packets injected during that interval. The probability that a packet fails on any of the $\leq D$ edges along its path is at most $Dn^{-c} \leq n^{-c+1}$. Thus, we get results analogous to the Leighton-Maggs-Rao distributed algorithm: Each packet is delivered w.h.p. after at most $O(L + D\log n)$ steps.

Arguing along the lines of previous sections, we improve this by the following

Theorem 18 *Assuming there is a scheduling algorithm for the single packets problem that delivers all packets to their destination in $(1+\delta)C + g(n)D + f(n)$ steps for some constant $\delta > 0$, then there is a scheduling algorithm for the L-bursty, $1-\epsilon$ congested, traffic problem that delivers each packet to its destination w.h.p. in $O(L+g(n)D+f(n))$ steps, assuming that $f(n) \geq \log n$.*

Proof. Set $L' = \max\{L, \epsilon^{-1}\alpha[Dg(n)+2f(n)]\}$ for some constant α. Partition time into consecutive frames of length $\alpha f(n)$ each. Consider the following "schedule." When a packet is injected, wait till the begining of the next frame, then insert a delay $\in_U [0, L']$, finally, move the packet one step every $\alpha g(n)$ time steps. Consider any edge e and any particular frame. Any packet that moves across e during that frame must have been generated within a time interval of length at most $\alpha f(n) + L' + \alpha Dg(n) + \alpha f(n) \leq (1+\epsilon)L'$. There are at most

$(1-\epsilon)(1+\epsilon)L' = (1-\epsilon^2)L'$ such packets. Thus, w.h.p. the number of packets that are actually scheduled to cross e during the frame is at most $(1-\delta')\alpha f(n)$, where δ' is a function of ϵ. The number of steps a packet takes during the frame is at most $f(n)/g(n)$. Therefore, using the assumed algorithm, we can schedule all the path segments that fall into this frame so that each packet is delivered within $(1+\delta)(1-\delta')\alpha f(n) + 2f(n) \leq \alpha f(n)$ steps, where the inequality is true under the assumption that α is sufficiently large. This schedule can be completed within the frame. □

5 Distributed Execution

In this section, we sketch the ideas used in implementing the above algorithms in a distributed fashion. The detailed algorithms will be presented in the full version of the paper.

We assume that packets may carry a small amount of extra routing data. Individual nodes have access to the queues of the edges adjacent to them. They can decide to forward packets from incoming queues into outgoing queues, or leave packets in the incoming queues. Aside from data stored in the packets, no other data is communicated between nodes. At any given time, a node does not have access to data stored in packets that have not yet reached it.

The distributed implementation is straightforward. Packets carry their path id and data used by the algorithm. We assume a global clock, so the partition into frames at all levels of the recurrence is known to all nodes. Other than the path, the only data needed for each packet is its current delay (given as the clock tick it's waiting for), and the level of the recurrence it's at.

The algorithm run by each node is the following. At each time step, the node examines its incoming queues. Only packets that were waiting for the current clock tick need to be handled. There are two cases:
Case 1: The packet is at the bottom level of the recurrence. In this case, the packet has to move to an adjacent node according to its path. Let's assume for a moment that indeed the node performs this move.
Case 2: Otherwise, the packet's schedule has to be further refined. In this case, the node tosses the needed dice, and determines a further delay of the packet. If the refined level is the bottom of the recurrence, the packet is delayed till the first available slot in the (constant size) frame.

The node maintains counters for each outgoing queue, for each level of recurrence. If the number of packets that has to move along a certain edge during a single frame of some level exceeds the upper bound allowed by the algorithm, then additional packets are not allowed to move along that edge during that frame. Such a packet is dumped. This is done by delaying it sufficiently, and changing the recurrence level, if needed.

Notice that blocked and bad edges might be detected not in the beginning of a frame, but later on. Also, a packet whose path crosses a blocked or bad edge during a frame might begin its move in the frame, and detect the problem only when it reaches the edge. This simply means that the packet managed to move ahead some steps despite the failure. It does not introduce additional failures (though perhaps prevents some), and the node can compute the required delay in case of failure given its access to the global clock.

Implementing the periodic algorithm is similar. The leading packet in each stream determines the schedule. Nodes must store the scheduling information for each stream (how long to delay packets arriving from that stream). The schedule can be redone every now and then at regular intervals by handling new leading packets. These ideas can also be used to implement the bursty traffic algorithm.

References

[1] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. In *Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science*, November 1994, pages 604-612.

[2] N. Alon and J.H. Spencer. *The Probabilistic Method.* Wiley, 1992.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. of the 23rd Ann. ACM Symp. on Theory of Computing*, San Diego, May 1993.

[4] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput competitive on-line routing. In *Proc. of the 34th Ann. Symp. on Foundations of Computer Science*, Palo Alto, November 1993.

[5] J. Beck. An algorithmic approach to the Lovász Local Lemma. *Random Structures and Algorithms*, 2(4):367–378, 1991.

[6] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D.P. Williamson. Adversarial queueing theory. In these proceedings.

[7] R.L. Brooks. On colouring the nodes of a network. *Proc. Cambridge Philos. Soc.*, 37:194–197, 1941.

[8] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal et al., eds., *Infinite and Finite Sets.* North Holland, 1975, pages 609–628.

[9] R. Gawlick, A. Kamath, S. Plotkin, and K. Ramakrishnan. Routing and admission control in general topology networks. Technical report no. STAN-CS-TR-95-1548, Stanford University, 1995.

[10] F.T. Leighton. Methods for message routing in parallel machines. In *Proc. of the 24th Ann. ACM Symp. on the Theory of Computing*, May 1992, pages 77–96.

[11] F.T. Leighton and B.M. Maggs. Fast algorithms for finding O(congestion + dilation) packet routing schedules. In *Proc. of the 28th Hawaii International Conference on System Sciences*, January 1995, vol 2, pages 555–563.

[12] F.T. Leighton, B.M. Maggs, and S.B. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[13] F.T. Leighton, B.M. Maggs, A.G. Ranade, and S.B. Rao. Randomized routing and sorting on fixed-connection networks. *J. Alg.*, 17(1):157–205, 1994.

[14] J. Spencer. *Ten Lectures on the Probabilistic Method.* SIAM, 1987.