Universal $O(\text{congestion} + \text{dilation} + \log^{1+\epsilon} N)$ Local Control Packet Switching Algorithms

Rafail Ostrovsky*
Bellcore

Yuval Rabani†
Technion

Abstract

In 1988, Leighton, Maggs and Rao proved a much celebrated result: that for any network, given any collection of packets with a specified route for each packet, there exists an "optimal" schedule for all these packets. That is, there exists a schedule of the motion of the packets such that at each step, every edge is crossed by at most one packet, and all the packets are delivered to their destinations in $O(C + D)$ steps, where C is the "congestion" (i.e., the maximum number of paths that share the same edge), and D is the "dilation" (i.e., the length of the longest path). The proof was non-constructive and relied on Lovász Local Lemma. In a followup paper, Leighton, Maggs, and Richa gave a centralized algorithm for finding the schedule. The original paper left open the question whether there exists a constructive distributed "on-line" algorithm with the same optimal performance. Last year, Rabani and Tardos presented a randomized local-control algorithm which with high probability delivers all packets in time $O\left(C + D \cdot \left((\log^* N)^{O(\log^* N)}\right) + + (\log N)^6\right)$.

In this paper, we show a nearly optimal local control algorithm for this long-standing open problem. That is, we show a randomized local control algorithm which for any network topology delivers all the packets to their destinations in time $O(C + D + \log^{1+\epsilon} N)$ with high probability, where N is the size of the problem, and $\epsilon > 0$ is arbitrary. Our result has implications to ATM (Asynchronous Transfer Mode) packet switching algorithms and other applications.

*Bell Communications Research, MCC-1C365B, Morristown, NJ 07960-6438, USA. Email: rafail@bellcore.com

†Computer Science Department, Technion — IIT, Haifa 32000, Israel. Part of this work was done while visiting Bell Communications Research. Work supported by grants from the Fund for the Promotion of Sponsored Research and from the Fund for the Promotion of Research at the Technion, and by a David and Ruth Moskowitz Academic Lecturship award. Email: rabani@cs.technion.ac.il

1 Introduction

Problem statement and motivation. With the tremendous growth of large-scale networks, and the ever-increasing demand for multimedia and other high-bandwidth applications, efficient packet-routing algorithms play an increasing role in current and future network performance [8]. An accepted international standard for packet routing is the so-called Asynchronous Transfer Mode (ATM) standard, where all messages are divided into "cells" (which we call packets[1]) and routed using a virtual circuit model, consisting of two steps: path selection and packet movement. During path selection, every connection request is assigned a fixed routing path (a "virtual circuit"), and then the stream of packets moves along this path. As any packet-routing algorithm can be viewed as a combination of path selection and packet movement decisions, both topics received considerable attention in the literature (e.g., see the survey by Leighton [9]).

In this paper, we concentrate on the second task, namely packet scheduling when routes are fixed. (For information on path selection algorithms, see, e.g., the survey by Plotkin [16].) Both specific and general network topologies were considered in the literature. In this paper, we consider arbitrary network topologies, modeled as arbitrary directed graphs, where the vertices are switches and the edges are communication links with incoming and out-going queues for packets. Every packet needs to traverse a pre-specified path. Routing is done in a synchronous "store-and-forward" manner, where at each step any edge can be crossed by at most one packet and packets traverse their pre-specified routes. In the simplest setting, all packets start at the same time, and we measure the number of steps it takes for all packets to reach their destinations. In this setting, two parameters seem to be crucial to the performance: the *congestion*, i.e., the maximum number of paths that go over the same edge (we denote this by C); and, the *dilation*, i.e., the length of the longest path any packet must travel (which we denote by D).

Related work. Considerable effort has been devoted to packet routing, including packet movement, in specific network topologies. We refer the reader to Leighton's survey [9] for more detail. In general topologies, two approaches to packet movement are prevalent in the literature. One approach, introduced by Leighton, Maggs, Ranade, and Rao [11], is based on assigning random ranks to the packets and using these ranks to resolve conflicts. For the special case of bounded degree *leveled* networks, Leighton, Maggs, Ranade, and Rao [11] give a randomized algorithm which with probability $1 - 1/poly(N)$ delivers all packets in $O(C + \text{depth} + \log N)$ steps.

[1] Asynchronous Transfer Mode (ATM) international standard makes a distinction between "packets" and "cells", where packets may have different sizes, but they are further decomposed into cells of 53 bytes each [3], where these "cells" are then treated as indivisible units. In this paper, we ignore this distinction and assume that packets and cells are equivalent and indivisible.

Meyer auf der Heide and Vöcking [13] modify this algorithm to get for arbitrary networks an $O(C + \text{diameter} + \log N)$ guarantee when routing only along shortest paths.

A different approach, that of inserting random delays, was first considered in the ground-breaking paper of Leighton, Maggs, and Rao [10]. They show using the Lovász Local Lemma that there exists a schedule which delivers all the packets to their destinations in time $O(C + D)$. (Moreover, they show that the schedule can be implemented with constant-size queues. We do not address the issue of queue sizes in this extended abstract.) Their non-constructive result was very significant for two reasons: (1) if all the packets start at the same time, and if at most one packet can cross an edge at a time, then clearly $\max\{C, D\}$ is a lower bound on the delivery time and hence $O(C + D)$ is optimal up to constant factors; (2) it establishes that path-selection (so as to minimize C and D) and scheduling of the packet movement can be done independently. (Indeed, very recently Srinivasan and Teo [18] announced a path selection algorithm that gets within a constant factor of the best $C+D$ value.)

In a followup paper, Leighton, Maggs, and Richa [12] show how to use Beck's constructive version of the Local Lemma [4] in order to construct a centralized algorithm. The algorithm, given a specification of the graph and the packet routes, searches through a polynomial number of alternative schedules and in polynomial time finds one which works in $O(C + D)$ time. While the schedule found is asymptotically optimal, to find such a schedule requires a centralized algorithm. On the other hand, we are interested in the distributed setting, where processors can only talk to their neighbors in the communication network. A special case of distributed routing algorithms are "local control" algorithms (also called "on-line" algorithms in the original terminology of [10], and "distributed" algorithms elsewhere), where at each step the switches make decisions on which packets to move forward along their paths based only on the routing information that the packets carry and on the local history of execution. (I.e., *no* additional communication is necessary to execute the routing algorithm.) An even more restrictive case are the local control algorithms where the switches need not know what the underlying topology of the network is, and in fact all execute identical code, where their state depends only on their local "view" so far. Such algorithms are sometimes called "universal local control" switching algorithms. The algorithms in [11, 13] essentially use local control.

For general-topology networks, the original paper of Leighton, Maggs, and Rao [10] gives an $O(C + D \cdot \log N)$ steps randomized local-control algorithm. It leaves as a major open problem the task of finding a local-control $O(C + D)$ algorithm. In fact, the authors even say: *"We suspect that finding such an algorithm (if one exists) will be a challenging task"* [10]. We note that an $O(C + D + \log N)$ guarantee would match the centralized algorithm guarantee in almost all proposed applications. Recently, Rabani and Tardos [17] made significant progress towards resolving this question: They show a randomized local-control algorithm which with probability $1 - 1/\text{poly}(N)$ delivers all packets in time $O\left(C + D \cdot \left((\log^* N)^{O(\log^* N)}\right) + (\log N)^6\right)$. Cypher, Meyer auf der Heide, Scheideler, and Vöcking [7] give a distributed bufferless algorithm that delivers all packets to their destinations in $O(C + D \log\log N + \log N \log\log N / \log\log(CD))$ steps w.h.p., provided that each edge can transfer $\Theta(\log(CD)/\log\log(CD))$ packets in a single step.

In addition to the simplest "one packet per stream" model, several dynamic models of packet routing have been considered. Perhaps the simplest is the following periodic traffic model: We are given a collection of packet streams. Packets are injected into stream i at a regular rate R_i (that is, every $1/R_i$ steps a packet is injected). The packets in a stream all follow the same path. On any edge, the rates of streams that use that edge sum up to less than 1. Other models considered in the literature include stochastic traffic models (most notably, Poisson arrivals), and various adversarial traffic models (see below).

For the periodic model, Parekh and Gallager [14, 15] provide a simple distributed algorithm that delivers streams of packets injected at regular intervals, guaranteeing at delay of $O(D_i/R_i)$ for a packet in stream i, where D_i is the path length for this stream, and R_i is the rate of injection. Rabani and Tardos [17] provide generic reductions from the case of streams of packets to the batch delivery model discussed above. These reductions hold both for the periodic model, as well as for the adversarial models suggested by Borodin, Kleinberg, Raghavan, Sudan, and Williamson [5], and further developed by Andrews, Awerbuch, Fernández, Kleinberg, Leighton, and Liu [1]. (These papers consider an issue of related interest, namely the stability of various queuing disciplines under adversarial packet injection strategies. However, they are not concerned with worst-case bounds on the delivery delays.) Under some technical conditions, the reductions of [17] give the same guarantee for streams as for batch delivery, with the congestion C replaced by $\max\{1/R_i\}$ in the periodic model, and by the maximum overload period in the adversarial models. (See also Broder, Frieze, and Upfal [6] for related work.) Very recently, Andrews, Fernández, Harchol-Balter, Leighton, and Zhang [2] announced an $O(D_i + 1/R_i)$ centralized algorithm for the periodic traffic model.

Our results. The main result of this paper is to show a randomized local control algorithm that for any $\epsilon > 0$ with probability $1 - 1/\text{poly}(N)$ delivers packets in time

$$O\left(C + D + (\log^{1+\epsilon} N)\right)$$

where the hidden constant depends on ϵ. It happens to be the case that all known "universal local control" randomized routing algorithms (i.e. the $O(C + D\log N)$ algorithm of [10] and that of [17]) can be phrased as a complicated backoff protocol, where a packet tries to cross an edge, and if that is impossible, it goes to sleep for a while, then tries again. This is also the case for our protocol. The details are excluded from this extended abstract. Our result "beats" the [10] lower bound for a simple randomized strategy: Legithon, Maggs, and Rao have shown that a strategy of assigning to packets random priorities may yield an $\Omega\left(\left(\frac{\log N}{\log\log N}\right)^{3/2}\right)$ schedule for $C = D = O\left(\frac{\log N}{\log\log N}\right)$. Moreover, we also address the issue of handling "streams" of packets using the reductions in [17] which in turn has applications to ATM and other routing problems. The details are deferred to the full version of the paper.

Packet loss. So far, we have stated all randomized results in terms of a low probability of failure (at most $1/\text{poly}(N)$, where N is the size of the problem). Sometimes, in particular when routing streams of packets, it makes sense to drop some of them in order to speed up the delivery of the rest. In [17] this issue is addressed, and the results generalize to loss probability p, as long as $p \leq D^{-1}$ (simply replace N by p^{-1}). Similar claims hold for the results in this paper. The details are deferred to the full version of the paper.

Methods and approaches to the problem. The basic idea underlying our algorithm is the notion of refinement of an infeasible schedule. This follows the Leighton, Maggs, and Rao results [10]. An infeasible schedule, where several packets are allowed to cross an edge simultaneously, is divided up into time frames (i.e. intervals of time), where the congestion in each time frame is roughly proportional to the frame length. The schedule in each frame is then refined by inserting a random delay for each packet in the beginning of the frame. The idea is that if one considers the packets that cross an edge during the frame, their crossing times would be spread throughout the frame, so that the congestion restriction would now hold for much smaller time frames. The initial step of the [10] refinement procedure takes the trivial schedule, where all packets move without interruption, and inserts random initial delays distributed uniformly in $[0, \lambda C]$, for some constant λ. Leighton, Maggs, and Rao prove the following lemma using the Chernoff bound:

Lemma 1 (Leighton, Maggs, and Rao [10]) *For every constant ν_1 there is a constant ν_2 such that with probability at least $1 - N^{-\nu_1}$, the resulting schedule has the property that in every frame of length $\ell \geq \nu_2 \log N$ the congestion is at most ℓ.*

This lemma guarantees the high-probability success of the first refinement step. However, similar claims about further refinement steps do not hold w.h.p. In [10], this is overcome by using the Local Lemma. In the local control algorithm of Rabani and Tardos [17], they use the following large deviation bound to estimate the number of packets that fail the refinement step:

Lemma 2 (Rabani and Tardos [17]) *Let X_1, X_2, …, X_N be 0-1 random variables, such that each is independent of any combination of the rest excluding a fixed set of size at most k. Further suppose that for all i, $\Pr[X_i = 1] \leq p$. Let $\delta \geq 1$. Then, $\Pr[\sum X_i > 4e\delta pN] < 4ek2^{-\delta pN/k}$.*

The small number of packets that fail in a refinement step are moved into a "catch-up track". The catch-up track gets scheduled in parallel with the regular track (by alternating moves), and is refined recursively. The purpose of the catch-up track is to enable packets to catch up on their intended schedule, and to boost their success probability. This is possible because the congestion in the catch-up track is small compared with its length.

In our algorithm, we use catch-up tracks too. We overcome the inherent slowdown due to the recursive refinement procedure of [17] by collecting packets into much larger catch-up tracks, which have very small congestion relative to their length. Thus we are able to pack all the catch-up tracks together, and avoid recursion. The large catch-up tracks would introduce an additive term which is even larger than the one in [17]. To avoid it, we introduce an initial refinement process that is invoked a constant number of times, and avoids the large additive term. The idea follows the first step in Beck's algorithmic version of the Local Lemma [4]. The collection of failed packets is composed of small connected components (with respect to correlations), w.h.p. The small size of each component makes it behave, in some sense, like a single edge. Thus we are able to schedule each component separately by simply repeatedly trying to schedule them independently from each other until all packets succeed.

2 The Algorithm

First, let us present the overview of the local control algorithm. Our overall strategy builds on the work of [10, 12, 17], namely we have regular time-frames, which we refine iteratively (as introduced by [10]), as well as "catch-up" frames (introduced in [17]). We differ substantially from previous work in the way we move packets to catch-up frames, in the way we refine those frames, and in the way we schedule together the regular and catch-up frames. Also, we introduce an "initial refinement process" which allows us to improve the additive term in the delay guarantee through a different refinement process applied prior to the main one.

Time is divided (mod 3) into two modes, the "regular" mode, and the "catch-up" mode, where the protocol executes one step of a "regular" mode, followed by two steps of the "catch-up" mode, then repeats. Packets are "active" in either one of these two modes (but not both). Packets remember what mode they are currently in. All packets start in a "regular" mode, and then sometimes switch to "catch-up" mode or back to "regular" mode. If a packet is in a "regular" mode, it only moves (and counts steps) in "regular" steps and goes to sleep (i.e. stays put in the network) during "catch-up" steps. If a packet is in the "catch-up" mode it goes to sleep during "regular" time. Since by design there are two "catch-up" steps for every "regular" step, we talk about "catch-up time" going "twice as fast" as regular time.

Whenever packets cross edges along their route, they inform the edges (i.e. both vertices) what state they are in (and some other information concerning the specifics of their state). Edges keep counts as to how many packets they have seen so far path through them in various states. This information is used in order to determine when packets switch from "regular" mode to "catch-up" mode and back, as well as specifics of the main refinement process of both regular and catch-up modes.

In both modes, there are $O(\log^* N)$ "levels" of refinement, where each level of refinement requires certain random delays at regular intervals of time. Each packet keeps track of all these levels, and goes to sleep at appropriate times. Now we describe (in a top-down fashion) the main refinement process, both of the regular track and of the catch-up track.

We start with some notation. A path π is a pair (p, L), where p is a packet id and L an ordered list of pairs (t_0, q_0), (t_1, q_1), …, (t_ℓ, q_ℓ). In the pair (t_i, q_i), t_i is a time slot and q_i is a queue (edge). The pairs satisfy the following conditions: For all i, $t_{i+1} > t_i$, and q_{i+1} is adjacent to q_i (i.e., there is a vertex v such that q_i is an edge incoming to v and q_{i+1} is an edge outgoing from v).

A frame S is a pair (T, P), where T is a segment of time slots and P is a set of paths. For $T = [t, t']$, $\pi \in P$, $\pi = (p, \{(t_0, q_0), \ldots, (t_\ell, q_\ell)\})$, we have $t_0 = t$ and $t_\ell \leq t'$. For two frames $S = (T, P)$, $S' = (T, P')$, we denote $S \cup S' = (T, P \cup P')$. For two frames $S = (T, P)$, $S' = (T', P')$, such that $T = [t, t']$, $T' = [t' + 1, t'']$, we denote $S \circ S' = (T \cup T', P \circ P')$, where $P \circ P'$ is the set of paths derived from $P \cup P'$ by replacing pairs of paths of the same packet by their concatenation.

Our starting point for the regular track is the schedule derived by inserting random initial delays as in Lemma 1. Let the initial "time-frame" length be $T_0 = O(\log^2 N)$ and let the initial "relative congestion" be $r_0 = 1$. Then the guarantee of Lemma 1 can be restated as follows: W.h.p., in every frame of length $\ell \geq \sqrt{T_0}$, every edge is crossed by at most $r_0 \cdot \ell$ packets. In our main refinement process of the regular track we "refine" this schedule while we guarantee that for every $\sqrt{T_j}$ time-frame, the congestion during this time-frame is going to be at most $r_j \cdot \sqrt{T_j}$, for T_j and r_j as defined below. We proceed with this refinement until every constant number of steps only a constant number of packets need to cross a constant number of edges. At this point, we can just make the schedule feasible by simulating every step of the schedule by a constant number of actual steps.

We now define T_j and r_j. Let α, β be absolute constants whose value will be implicit in the proofs. We define two sequences $\{T_j\}$ and $\{r_j\}$, $j = 0, 1, \ldots, j_{\max}$, by:

$$T_j = \log^6 T_{j-1}, \tag{1}$$

$$r_j = \left(1 + \frac{\alpha}{\log T_{j-1}}\right) r_{j-1}, \qquad (2)$$

with the initial conditions $T_0 = O(\log^2 n)$ and $r_0 = 1$, and with the stopping condition $T_{j_{\max}} \leq \beta$.

Regular track refinement step. In the regular track refinement step j, the schedule is divided up into frames of length T_j. Each of those is refined separately. We insert for each path in each T_j frame a random initial delay uniformly distributed between 0 and $\sqrt{T_j}$. Now consider frames of size $\sqrt{T_{j+1}}$ after the random initial delays of step j have been fixed. We say that a packet *fails* in a $\sqrt{T_{j+1}}$ frame if it crosses an over-congested edge, where an edge is over-congested if more than $r_j \cdot \sqrt{T_{j+1}}$ packets cross it during this $\sqrt{T_{j+1}}$-frame. We remark that if a path fails any $\sqrt{T_{j+1}}$ frame, then we fail the entire T_j path which contains this $\sqrt{T_{j+1}}$ frame (in fact, we fail a longer path, of length $2T_j^k$ as discussed below). We move the failed path to the "catch-up track"also discussed below.

We pick k as a sufficiently large constant (to be fixed later) and "group" together T_j^{k-1} frames of length T_j each. Let $\mathrm{RTF}_j(i,j)$ denote the ith level j frame in the regular track. This is a frame of length T_j^k. Let $\mathrm{RTF}_{j+1}(i,j)$ denote the same frame after the level j initial delays of length $[0..\sqrt{T_j}]$ have been inserted. We further denote, for $k \in \{j, j+1\}$, $\mathrm{RTF}_k(i,j) = \mathrm{RTF}_k^1(1,i,j) \circ \cdots \circ \mathrm{RTF}_k^1(T_j^{k-1}, i, j)$, where $\{\mathrm{RTF}_j^1(f,i,j)\}$ are frames of length T_j, and $\{\mathrm{RTF}_{j+1}^1(f,i,j)\}$ are the same frames after insertion of the level j initial delays. Thus, the refinement step of the regular track has the following from:

$$\underbrace{\overbrace{\mathrm{RTF}_j^1(1,i,j)}^{\substack{\text{insert}\\ \text{delay} \in_U [0..\sqrt{T_j}]}} \circ \cdots \circ \cdots \circ \cdots \circ \overbrace{\mathrm{RTF}_j^1(T_j^{k-1},i,j)}^{\substack{\text{insert}\\ \text{delay} \in_U [0..\sqrt{T_j}]}}}_{RTF_j(i,j)}$$

contains T_j^{k-1} frames of length T_j each

$$\boxed{\Downarrow \ \text{level} \ \ j \ \ \text{regular tack refinement} \ \Downarrow}$$

$$\underbrace{\overbrace{\mathrm{RTF}_{j+1}^1(1,i,j)}^{\substack{\text{fixed}\\ \text{delays} \in [0..\sqrt{T_j}]}} \circ \cdots \circ \cdots \circ \cdots \circ \overbrace{\mathrm{RTF}_j^1(T_{j+1}^{k-1},i,j)}^{\substack{\text{fixed}\\ \text{delays} \in [0..\sqrt{T_j}]}}}_{\mathrm{RTF}_{j+1}(i,j)}$$

Notice that some of the paths in $\mathrm{RTF}_{j+1}(i,j)$ are discarded when the frame is partitioned to create the level $j+1$ frames $\{\mathrm{RTF}_{j+1}(i', j+1)\}_{i'}$, namely the paths that use over-congested edges. If a packet happens to cross the over-congested edge, we fail its entire path in $\mathrm{RTF}_j(i,j) \circ \mathrm{RTF}_j(i+1,j)$. This is a path of length at most $2T_j^k$.

Now we are ready to describe the catch-up track. Let $\mathrm{CTF}_j(i,j)$ denote the ith level j frame in the catch-up track. This frame has length $2T_j^k$. Before we explain how our catch-up schedule is designed and refined, let us first explain the structural and temporal correspondence between the regular track frames and the catch-up track frames of the same level:

SIMULATION VIEW:

$$\cdots \circ \overbrace{\mathrm{RTF}_j(a,j) \circ \mathrm{RTF}_j(b,j)}^{\mathrm{CTF}_{j(\overline{ab},j)}} \circ \overbrace{\mathrm{RTF}_j(c,j) \circ \mathrm{RTF}_j(d,j)}^{\mathrm{CTF}_{j(\overline{cd},j)}} \circ \mathrm{RTF}_j(e,j) \circ$$

$$\cdots \circ \mathrm{RTF}_j(a,j) \circ \underbrace{\mathrm{RTF}_j(b,j) \circ \mathrm{RTF}_j(c,j)}_{\mathrm{CTF}_{j(\overline{bc},j)}} \circ \underbrace{\mathrm{RTF}_j(d,j) \circ \mathrm{RTF}_j(e,j)}_{\mathrm{CTF}_{j(\overline{de},j)}} \circ \cdots$$

Recall, however, that the catch-up track goes twice as fast. Therefore we have the following

TEMPORAL VIEW:

$$\cdots \circ \mathrm{RTF}_j(a,j) \circ \overbrace{\mathrm{RTF}_j(b,j)}^{\mathrm{CTF}_{j(\overline{ab},j)}} \circ \mathrm{RTF}_j(c,j) \circ \overbrace{\mathrm{RTF}_j(d,j)}^{\mathrm{CTF}_{j(\overline{cd},j)}} \circ \mathrm{RTF}_j(e,j) \circ$$

$$\cdots \circ \mathrm{RTF}_j(a,j) \circ \mathrm{RTF}_j(b,j) \circ \underbrace{\mathrm{RTF}_j(c,j)}_{\mathrm{CTF}_{j(\overline{bc},j)}} \circ \mathrm{RTF}_j(d,j) \circ \underbrace{\mathrm{RTF}_j(e,j)}_{\mathrm{CTF}_{j(\overline{de},j)}} \circ \cdots$$

We now specify what gets into the catch-up track (we shall see that there are three different ways a packet can land in the level j catch-up track) and how the catch-up track is refined. Recall that we partitioned the regular track into frames $\mathrm{RTF}_j(i,j)$ of length T_j^k each. Every adjacent pair will have a catch-up track frame of length $2T_j^k$. If a packet fails in the refinement step of a regular frame $\mathrm{RTF}_j(i,j)$ then its entire path in this frame *and the next T_j^k frame* moves to the corresponding catch-up frame. We denote the frame of packets which go from regular frame $\mathrm{RTF}_j(i,j) \circ \mathrm{RTF}_j(i+1,j)$ *to the same level* catch-up track by $\mathrm{CTF}_{\rightarrow}(i,j)$. Informally, notice that if a packet manages to "catch-up" in the catch-up track then it is back "on schedule" and can go back to the "regular" track.

We will shortly specify an iterative refinement process for the catch-up track. However, before we proceed, we must specify what happens when in the iterative process, some path at level j of a catch-up track frame fails (which we will show will happen only with exponentially small probability in the length of the catch-up track). In this case, we *dump* the entire $2T_{j-1}^k$ path which contains the current $2T_j^k$ path to the *future* (i.e. temporal following the current one) level $j-1$ catch-up track frame $\mathrm{CTF}_{j-1}(i,j-1)$. We denote the frame of packets dumped this way by $\mathrm{CTF}_{\uparrow}(i,j-1)$.

Finally, since we are talking about the iterative refinement process of the catch-up track, it means that packets get into the level j catch-up track not only from the level j regular track, but also as a result of a refinement of the catch-up track itself, i.e., from the level $j-1$ catch-up track. We denote the frame of packets which go to the catch-up track frame $\mathrm{CTF}_j(i,j)$ from this source by $\mathrm{CTF}_{\downarrow}(i,j)$.

Note that $\mathrm{CTF}_j(i,j) = \mathrm{CTF}_{\rightarrow}(i,j) \cup \mathrm{CTF}_{\downarrow}(i,j) \cup$ $\cup \mathrm{CTF}_{\uparrow}(i,j)$, where $\mathrm{CTF}_{\rightarrow}(i,j)$ includes those paths in $\mathrm{CTF}_j(i,j)$ which were dumped from the level j regular track, $\mathrm{CTF}_{\downarrow}(i,j)$ includes the paths that were refined from the level $j-1$ catch-up track, and $\mathrm{CTF}_{\uparrow}(i,j)$ includes the paths that were dumped from the level $j+1$ catch-up track (we will explain how this is done, after we explain how the catch-up track is refined).

Catch-up track refinement step. In the catch-up track refinement step j, we will be guaranteed that the congestion in every $2T_j^k$ time frame is no more than $3T_j^6$. (We allow each of $\mathrm{CTF}_{\rightarrow}(i,j)$, $\mathrm{CTF}_{\downarrow}(i,j)$, $\mathrm{CTF}_{\uparrow}(i,j)$ to contribute T_j^6 to the congestion. Excess congestion is dumped to a higher level track.) We partition every path into segments of length T_j^{k-3}. Notice that there are exactly $2T_j^3$ such segments, and in order for a packet to travel its entire path, it must traverse (in order) each of these $2T_j^3$ segments of length T_j^{k-3} each. Let γ be a constant (to be defined later; γ determines the expansion of the catch-up track time during the main refinement process). We schedule the frame in $2T_j^3 + \gamma T_j^2$ frames of length T_j^{k-3} each. (That is, we allow γT_j^2 additional frames.) Now, in every $2T_j^k$ catch-up frame there is a collection of *sub-frames* to be refined (where the first sub-frame contains the first T_j^{k-3}-segment of each $2T_j^k$ path). We schedule all these segments by inserting random initial delays distributed uniformly between 0 and T_j^{k-4}. Then, we proceed to the next sub-frame which contains (for each packet) the next segment that has not succeeded yet. (A segment *fails* in the refinement at level $j+1$ if there is a frame F of size $2T_{j+1}^k = 2\log^{6k} T_j$ such that the segment crosses an edge with congestion greater than $\kappa \log T_j$ during F, for some constant κ to be defined later.) We say that a packet *succeeds* in the $2T_j^k$ catch-up frame if all $2T_j^3$ segments of length T_j^{k-3} have been scheduled (in $2T_j^3 + \gamma t_j^2$ tries).

We shall need some more notation in the analysis of the algorithm. Let $\mathrm{PATH}(p,k,i,j)$ denote the path of packet p in $\mathrm{CTF}_j(i,j)$ between time $t_0 + (k-1)T_j^{k-3}$ and $t_0 + kT_j^{k-3} - 1$, where t_0 is the time when the frame begins. Let $\mathrm{CTF}_j^1(s,i,j)$, $1 \le s \le 2T_j^3 + \gamma T_j^2$, denote the step s frame in the refinement process of $\mathrm{CTF}_j(i,j)$. I.e., $\mathrm{CTF}_j^1(s,i,j)$ has length T_j^{k-3} and contains a shift of $\mathrm{PATH}(p,k,i,j)$ for all p that appear in $\mathrm{CTF}_j(i,j)$ and do not use an over-congested edge. Let $\mathrm{CTF}_{j+1}^1(s,i,j)$ denote the same frame after insertion of the level j initial delays. Finally, denote $\overline{\mathrm{CTF}}_j(i,j) = \mathrm{CTF}_j^1(1,i,j) \circ \cdots \circ \mathrm{CTF}_j^1(2T_j^3 + \gamma T_j^2, i, j)$, and denote $\mathrm{CTF}_{j+1}(i,j) = \mathrm{CTF}_{j+1}^1(1,i,j) \circ \cdots \circ \mathrm{CTF}_{j+1}^1(2T_j^3 + \gamma T_j^2, i, j)$.

Finally we must specify how packets move between the catch-up track and the regular track. If a packet enters a catch-up track at level j and succeeds, put it back onto the next level j regular track frame, and if it fails, dump it to the next level $j-1$ catch-up track frame.

We call the algorithm we have just described the "main refinement algorithm". The following theorem states its delivery guarantees:

Theorem 3 *There is a constant k such that the following holds: Let $T = T(N)$ be a positive integer, and let $r > 0$ be an absolute constant. Suppose that the input to the main refinement algorithm is a schedule of the motion of the packets of length L, such that for every time interval $I = [t, t']$ with $|I| = t' - t + 1 \geq \sqrt{T}$, for every edge e, the number of packets that cross e during I is at most $r|I|$. Then, the main refinement algorithm produces a (random) feasible schedule of length $O(L+T^k)$. Furthermore, for any packet, the probability that it is excluded from the schedule is at most $L \cdot \exp(-T)$.*

We conclude from this

Corollary 4 *The main refinement algorithm can be used to guarantee w.h.p. the delivery of all packets within $O(C + D + poly(\log N))$ steps.*

Proof. Using Lemma 1, we can generate w.h.p. a schedule satisfying the conditions of Theorem 3 with $T = O(\log^2 N)$, $r = 1$, and $L = O(C + D)$. We apply our main refinement algorithm to this schedule. The theorem asserts that the main refinement algorithm provides the desired delay bounds w.h.p. (as $C, D \leq N$). ∎

The proof of theorem 3 appears in Section 3.

The drawback of the main refinement algorithm is the poly($\log N$) additive term in the delivery guarantee. The reason for this term is that at the top level, we schedule frames of length T_0^k, and by using Lemma 1 to initiate the schedule, we begin with $T_0 = O(\log^2 N)$. The secret to improving the additive term is to take the schedule of Lemma 1 and refine it differently so that we end up with a schedule on which the main refinement algorithm can be invoked with a much smaller T_0. Our task is further complicated because the packet success guarantee of Theorem 3 depends on T_0, so we will have to boost it up.

Initial refinement process. The initial refinement algorithm, works as follows. After invoking Lemma 1, we divide the schedule into frames of length $\log^{1+\epsilon} N$, for a constant ϵ as small as we wish. In the initial schedule, every frame of length $\nu_2 \log N$ has congestion at most its length. We refine each $\log^{1+\epsilon} N$ frame to get that in every frame of size $\log^{1/2k} N$ the congestion is at most its length. Then we invoke the main refinement algorithm on this schedule.

The initial refinement process has a regular track and a catch-up track too. (We shall refer to them as the regular i-track and the catch-up i-track.) The regular i-track refinement is simple: we insert random initial delays (distributed uniformly between 0 and $\lambda \log^{1+\epsilon} N$), remove paths that cross over-congested edges (i.e., where in a single $\log^{1/2k} N$ frame the congestion exceeds the length), then call the main refinement algorithm, and finally remove paths that failed in the main refinement algorithm. The removed paths end up in the catch-up i-track.

We introduce some additional notation: Let $\mathrm{RIF}_0(i,0)$ denote the ith regular i-frame prior to its refinement. This is a frame of length $\log^{1+\epsilon} N$. Let $\mathrm{RIF}_1(i,0)$ denote the ith regular i-frame after the insertion of the random initial delays. Let $\mathrm{RIF}_1(i,1)$ denote the frame derived from $\mathrm{RIF}_1(i,0)$ by removing packets that cross over-congested edges. Let $\mathcal{G}(i)$ denote the interference graph among the paths of packets in $\mathrm{RIF}_0(i,0)$ but not in the schedule produced by the main refinement algorithm with input $\mathrm{RIF}_1(i,1)$. These are the paths that are dumped onto the catch-up i-track.

There will be a catch-up i-frame for each regular i-frame, and of the same initial length. There is an important difference between the role of the catch-up track in the initial refinement process, and its role in the main refinement algorithm: In the main refinement algorithm, the catch-up track serves its purpose because the congestion there is considerably reduced. Here, we cannot guarantee a reduction in the congestion (dependencies are too big). On the other hand, we are guaranteed that w.h.p. $\mathcal{G}(i)$ is composed of small connected components. The advantage of a small connected component is that it behaves, in some sense, like a single edge, so we can succeed in refining it by simply trying several times. We will refine each connected component of $\mathcal{G}(i)$ separately (that is, the frames of different components do not necessarily coincide in time). We refine the frame gradually, in a constant number of steps. (The constant depends on ϵ.) In the jth step, the $\log^{1+\epsilon} N$ frame is divided into frames of length $\log^{1-j\epsilon} N$, where the congestion in each such frame is at most its length. We refine the frame by inserting random initial delays distributed uniformly between 0 and $\lambda \log^{1-j\epsilon} N$, and then check if the conditions for the next step hold. We repeatedly try this, till the entire connected component of $\mathcal{G}(i)$ succeeds. Then we move on to the next $\log^{1-j\epsilon} N$ frame. The idea is that though a single frame may fail repeatedly, the total number of failures is not too high w.h.p. The last step is special. We divide the $\log^{1+\epsilon} N$ frame into frames of length $\log^{k\epsilon/(k-1)} N$, and run the main refinement algorithm on each. Again, we repeat unless the entire connected component succeeds. The motivation is similar to that of the previous steps.

In the following definitions, $\mathcal{C}$ is a connected component of $\mathcal{G}(i)$. Let $\mathrm{CIF}_0(\mathcal{C},0)$ denote the catch-up i-frame containing the paths in $\mathcal{C}$. This is a frame of length $\log^{1+\epsilon} N$. Let $j_{\max} = \epsilon^{-1} - (2k-2)^{-1}$ and let $\delta = \epsilon/(2k-2)$. For $0 \le j \le j_{\max}$, let $\mathrm{CIF}_j(\mathcal{C},0)$ be the level j refinement of $\mathrm{CIF}_0(\mathcal{C},0)$. We further denote $\mathrm{CIF}_j(\mathcal{C},0) = \mathrm{CIF}^1_j(1,\mathcal{C},j) \circ \cdots \circ \mathrm{CIF}^1_j(s_{\max},\mathcal{C},j)$, where the following conditions hold: (a) If $j < j_{\max}$, each $\mathrm{CIF}^1_j(s,\mathcal{C},j)$ is a frame of length $\log^{1-j\epsilon} N$; (b) Each $\mathrm{CIF}^1_{j_{\max}}(s,\mathcal{C},j_{\max})$ is a frame of length $\log^{2k\delta} N$; (c) $s_{\max} = s_{\max}(\mathcal{C}, j)$ is a random variable. Finally, let $\mathrm{CIF}^1_{j+1}(q, s, \mathcal{C}, j)$ denote the qth attempt at refining $\mathrm{CIF}^1_j(s, \mathcal{C}, j)$, and let $q_{\max} = q_{\max}(s, \mathcal{C}, j)$ denote the random variable whose value is the number of times we attempt to refine $\mathrm{CIF}^1_j(s, \mathcal{C}, j)$.

The main idea of the analysis is to show a high probability bound on the $\sum_s q_{\max}(s, \mathcal{C}, j)$. The following theorem is the main result of this paper. It states the guarantee provided by the initial refinement algorithm invoking the main refinement algorithm:

Theorem 5 *Let $\epsilon > 0$ be an absolute constant. Suppose that the input to the initial refinement algorithm is a packet switching problem with congestion C and dilation D. Then, w.h.p. the switching algorithm delivers all packets to their destinations within $O(C + D + \log^{1+\epsilon} N)$ steps.*

The proof of this theorem appears in Section 4.

3 Analysis of the Main Refinement Process

In this section we prove that our main refinement algorithm guarantees w.h.p. the delivery of each packet within $O(C + D + \mathrm{poly}(\log N))$ steps. In the next section we analyze the initial refinement process, improving the additive term.

We now proceed to lay the ground for the proof of Theorem 3. We shall use the notation for frames from the previous section. Notice that all these frames are random variables. Throughout the proof, when we say that such a frame is *fixed*, we mean the following:

1. All the random delays which determine the value of this frame are fixed;

2. The properties of events discussed in that context are the conditional properties, conditioned on the choice of those random delays;

3. The claims regarding these properties hold no matter what the choice of those random delays is.

Let $A(e, f, i, j)$ denote the following event: In $\mathrm{RTF}^1_{j+1}(f, i, j)$ there exists a frame F of size $\ell \geq \sqrt{T_{j+1}} = \log^3 T_j$ such that the congestion on e during F is greater than $r_j \ell$.

Lemma 6 *For all e, f, i, j, $\Pr[A(e, f, i, j) \mid RTF_j(i, j)] \leq T^{-c_1}$, where c_1 is an absolute constant which can be taken arbitrarily large by increasing α appropriately.*

Proof. It is sufficient to check frames of size between $\log^3 T_j$ and $2\log^3 T_j$. Consider such a frame F of size ℓ. In $\mathrm{RTF}^1_{j+1}(f, i, j)$ the packets that cross e during F are a subset of the packets that in $\mathrm{RTF}^1_j(f, i, j)$ crossed e during a frame of size at most

$\sqrt{T_j}+\ell$. The number of such packets is at most $r_{j-1}(\sqrt{T_j}+\ell)$. Each has a probability of $\ell/\sqrt{T_j}$ of falling into F. Thus the expected number of packets that cross e during F is at most $r_{j-1}(\ell+\ell^2/\sqrt{T_j})$, and the probability that this number exceeds $r_{j-1}(\ell+\alpha\log^2 T_j)\le r_j\ell$ is bounded using Chernoff bounds by T_j^{-c}, where c is a constant that depends on α. There are at most $T_j+\sqrt{T_j}$ frames of size ℓ in $\mathrm{RTF}^1_{j+1}(f,i,j)$, and $\log^3 T_j+1$ different sizes. Summing up the probabilities for all these frames gives the total of $T_J^{-c_1}$, where $c_1\ge c-2$. ∎

Let $B(p,f,i,j)$ denote the event that during $\mathrm{RTF}^1_{j+1}(f,i,j)$ packet p crosses an edge e for which $A(e,f,i,j)$ occurs.

Corollary 7 *For all p,f,i,j,* $\Pr[B(p,f,i,j)\mid RTF_j(i,j)]\le T_j^{-c_1+1}$.

Proof. Sum up the bound in Lemma 6 over at most T_j edges that p crosses during $\mathrm{RTF}^1_{j+1}(f,i,j)$. ∎

Lemma 8 *Fix i,j and $RTF_j(i,j)$. For all p,f, $B(p,f,i,j)$ is independent of any combination of similar events which excludes a fixed subset of events of cardinality at most $r_{j-1}^2T_j^4$.*

Proof. Consider the interference graph among the paths in $\mathrm{RTF}^1_j(f,i,j)$. We claim that any event $B(p,f,i,j)$ is independent of any combination of similar events corresponding to packets whose paths are at distance at least three from p's path in this graph. The lemma follows immediately from this claim.

To prove the claim, consider the set of edges used by p's path. Let e be such an edge. $A(e,f,i,j)$ is independent of any combination of similar events, unless this combination contains an event $A(e',f,i,j)$ such that there is a packet p'' which uses both e,e' during $\mathrm{RTF}^1_j(f,i,j)$. ∎

Let $C(e,i,j)$ denote the following event: In $\mathrm{CTF}_{\rightarrow}(i,j)$ the number of packets that cross e is more than T_j^6.

Lemma 9 $\Pr[C(e,i,j)\mid RTF_j(i,j)]\le 2^{-\epsilon T_j^2}$*, for some constant $\epsilon<1$.*

Proof. Let k be the number of packets that cross e in $\mathrm{RTF}_j(i,j)$ or $\mathrm{RTF}_j(i+1,j)$. $k\le 2r_{j-1}T_j^k$. Let $p_1,p_2,\ldots,p_k$ denote these packets. The events that determine which packets get dumped are $B(p_t,f,i,j)$, $t=1,2,\ldots,k$, $f=1,2,\ldots,T_j^{k-1}$. Instead of counting how many packets get dumped, we count how many such events occur. This is clearly an over-estimate, since a packet may fail for more than one value of f. The total number of B-events to consider is $kT_j^{k-1}\le 2r_{j-1}T_j^{2k-1}$. By Corollary 7, the probability that a B-event occurs is at most $T_j^{-c_1+1}$. $B(p_t,f,i,j)$ is independent of all $B(p_{t'},f',i,j)$ for $f\ne f'$. By Lemma 8,

$B(p_t, f, i, j)$ depends on at most $r_{j-1}^2 T_j^4$ events $B(p_{t'}, f, i, j)$. Let $X_{t,i}$ be the indicator variable for the event $B(p_t, f, i, j)$. Using Lemma 2, we have

$$\begin{aligned} \Pr[C(e,i,j)] &\leq \Pr[\sum X_{t,i} > T_j^6] \\ &< 4er_{j-1}^2 T_j^4 2^{-T_j^6/4er_{j-1}^2 T_j^4} \\ &\leq 2^{-\epsilon T_j^2}. \quad \blacksquare \end{aligned}$$

Let $D(p, i, j)$ denote the event that packet p appears in $\mathrm{CTF}_{\rightharpoondown}(i, j)$, and that during this frame it crosses an edge e for which $C(e, i, j)$ occurs.

Corollary 10 $\Pr[D(p, i, j) \mid RTF_j(i, j)] \leq 2^{-T_j}$.

Proof. Sum up the bound in Lemma 9 for at most $2T_j^k$ edges that p crosses during $\mathrm{CTF}_{\rightharpoondown}(i, j)$. This upper bounds the probability that p crosses an edge for which $C(e, i, j)$ occurs, and therefore it also upper bounds $D(p, i, j)$ (which further requires that p is dumped). $\blacksquare$

Let $E(e, s, i, j)$, $1 \leq s \leq 2T_j^3 + \gamma T_j^2$, denote the following event: In $\mathrm{CTF}_{j+1}^1(s, i, j)$ there exists a frame F of size $2\log^{6k} T_j = 2T_{j+1}^k$, such that the number of packets that cross e during F is greater than $\kappa \log T_j$. Let $\mathcal{H}_s$ denote the history of the refinement process for $\mathrm{CTF}_j(i, j)$ up to step s.

Lemma 11 $\Pr[E(e, s, i, j) \mid \mathcal{H}_{s-1}] \leq T^{-c_2}$, *where c_2 is an absolute constant which can be taken arbitrarily large by increasing κ appropriately.*

Proof. Consider a particular frame F. The number of packets which may cross e during this frame is at most $3T_j^6$. Consider one of these packets. Of the T_j^{k-4} choices for initial delay in step s of the process, at most $2\log^{6k} T_j$ would cause the packet to cross e during F. Thus, the expected number of packets that end up crossing e during this frame is at most $6\log^{6k} T_j / T_j^{k-10} \ll 1$. Using Chernoff bounds we get that the probability that the number of such packets is more than $\kappa \log T_j$ is less than T_j^{-c} for some constant c which depends on κ. Summing this probability over all the relevant frames F completes the proof. $\blacksquare$

Let $F(p, s, i, j)$ denote the event that packet p appears in $\mathrm{CTF}_{j+1}^1(s, i, j)$, and during that frame it crosses an edge e for which $E(e, s, i, j)$ occurs.

Corollary 12 $\Pr[F(p, s, i, j) \mid \mathcal{H}_{s-1}] \leq T_j^{-c_2+k-3}$.

Proof. There is at most one k such that $\mathrm{PATH}(p, k, i, j)$ uses e. This path has length at most T_j^{k-3}. Summing up the bound in Lemma 11 over all the edges used by this path gives an upper bound on the desired probability. $\blacksquare$

Let $G(p, i, j)$ denote the event that packet p appears in $\overline{\mathrm{CTF}}_j(i, j)$ but does not appear in the final schedule for this interval.

Lemma 13 $\Pr[G(p,i,j) \mid \overline{CTF}_j(i,j)] \leq e^{-T_j}$.

Proof. The proof is by induction on $j' = j_{\max} - j$. For $j' = 0$ the claim is trivial, since $\Pr[G(p,i,j_{\max})] = 0$. So, assume the claim is true for $j+1$. Consider a frame $\mathrm{CTF}_j^1(s,i,j)$ and some path $\mathrm{PATH}(p,k,i,j)$ that is scheduled during $\mathrm{CTF}_j^1(s,i,j)$. $\mathrm{PATH}(p,k,i,j)$ fails in step s of the refinement process if either $F(p,s,i,j)$ occurs, or $G(p,i',j+1)$ occurs for some i' such that $\mathrm{CTF}_{\downarrow}(i',j+1) \subset \mathrm{CTF}_{j+1}^1(s,i,j)$. By Corollary 12,

$$\Pr[F(p,s,i,j) \mid \mathcal{H}_{s-1}] \leq T_j^{-c_2+k-3}. \tag{3}$$

By the induction hypothesis, $\Pr[G(p,i',j+1) \mid \overline{\mathrm{CTF}}_{j+1}(i',j+1)] \leq e^{-T_{j+1}} = e^{-\log^6 T_j} \ll T_j^{-c}$, where the constant c can be taken arbitrarily large by increasing β appropriately. Thus, averaging over all possible values of $\overline{\mathrm{CTF}}_{j+1}(i',j+1)$ gives that $\Pr[G(p,i',j+1) \mid \overline{\mathrm{CTF}}_j(i,j)] \ll T_j^{-c}$. Summing up this bound for at most T_j^{k-3} relevant values of i' gives

$$\Pr[\exists i',\ G(p,i',j+1) \mid \overline{\mathrm{CTF}}_j(i,j)] \leq T_j^{-c}. \tag{4}$$

The constant c is determined by β and k. Summing up the two bounds 3 and 4 gives that the probability that $\mathrm{PATH}(p,k,i,j)$ fails in step s of the refinement process is upper bounded by $T_j^{-c'}$ for some constant c' which depends on β, k, c_2.

Recall that p advances in step s of the refinement process if there exists k such that p advanced exactly $k-1$ times in the previous steps, and $\mathrm{PATH}(p,k,i,j)$ does not fail in step s. Thus, p's fate in the refinement process gives rise to the following Doob martingale: Let X_s, $s = 0,1,\ldots$, be the random variable whose value is the expected number of steps in which p succeeds, the expectation taken after the first s steps of the process. We count as success steps where p did not fail as well as steps where p did not appear (because it succeeded sufficiently many times). Notice that $X_0 = \left(1 - T_j^{-c'}\right)\left(2T_j^3 + \gamma T_j^2\right) > 2T_j^3 + \gamma' T_j^2$, for some $\gamma' < \gamma$. We are interested in $\Pr[X_{2T_j^3+\gamma T_j^2} < 2T_j^3]$. Since $X_{s-1} = E[X_s \mid \mathcal{H}_{s-1}]$, the sequence $\{X_s\}$ is a martingale, and since $|X_s - X_{s-1}| \leq 1$ we may use Azuma's inequality to conclude that

$$\begin{aligned}
\Pr[X_{2T_j^3+\gamma T_j^2} < 2T_j^3] &\leq \Pr[X_{2T_j^3+\gamma T_j^2} - X_0 < -\gamma' T_j^2] \\
&< e^{-(\gamma')^2 T_j^4/(4T_j^3+2\gamma T_j^2)} \\
&\leq e^{-T_j}. \qquad \blacksquare
\end{aligned}$$

Let $H(p,i,j)$ denote the event that packet p appears in $\mathrm{CTF}_{\uparrow}(i,j)$. Let $I(e,i,j)$ denote the event that in $\mathrm{CTF}_{\uparrow}(i,j)$ the number of packets that cross e is more than T_j^6. Let $J(p,i,j)$ denote the event that packet p appears in $\mathrm{CTF}_{\uparrow}(i,j)$, and that during this frame it crosses an edge e for which $I(e,i,j)$ occurs.

Lemma 14 *(i)* $\Pr[H(p,i,j) \mid RTF_j(i,j)] \le T_j^{-c_3}$*, where* c_3 *is an absolute constant which can be taken arbitrarily large by increasing* β *appropriately.*

(ii) $\Pr[I(e,i,j) \mid RTF_j(i,j)] \le 2^{-T_j^2}$.

(iii) $\Pr[J(p,i,j) \mid RTF_j(i,j)] \le 2^{-T_j}$.

Proof. By induction over $j' = j_{\max} - j$. For $j' = 0$ all three claims are obvious since $\mathrm{CTF}_\uparrow(i, j_{\max})$ is empty. So, assume the claims hold for $j' + 1$.

We begin by proving the first claim. $H(p,i,j)$ occurs if there is i' such that $\mathrm{CTF}_\downarrow(i', j+1) \subset \mathrm{CTF}_{j+1}(i-1, j)$, and one of the following holds: Either $D(p,i',j+1)$ occurs, or $G(p,i',j+1)$ occurs, or $J(p,i',j+1)$ occurs. By Corollary 10, $\Pr[D(p,i',j+1) \mid \mathrm{RTF}_{j+1}(i', j+1)] \le 2^{-T_{j+1}} = T_j^{-\log^5 T_j}$. Averaging over all possible values of $\mathrm{RTF}_{j+1}(i', j+1)$, we get $\Pr[D(p,i',j+1) \mid \mathrm{RTF}_j(i,j)] \le T_j^{-\log^5 T_j}$. By Lemma 13, $\Pr[G(p,i',j+1) \mid \overline{\mathrm{CTF}}_{j+1}(i',j+1)] \le e^{-T_{j+1}} \le T_j^{-\log^5 T_j}$. Again, by averaging, $\Pr[G(p,i',j+1) \mid \mathrm{RTF}_j(i,j)] \le T_j^{-\log^5 T_j}$. By the induction hypothesis (third claim), $\Pr[J(p,i',j+1) \mid \mathrm{RTF}_{j+1}(i',j+1)] \le 2^{-T_{j+1}} = T_j^{-\log^5 T_j}$. Once again, by averaging, $\Pr[J(p,i',j+1) \mid \mathrm{RTF}_j(i,j)] \le T_j^{-\log^5 T_j}$. The first claim follows by summing up the bounds on these probabilities for all the relevant values of i'.

To prove the second claim, consider an edge e. A packet might cross e during $\mathrm{CTF}_\uparrow(i,j)$ only if it crosses e during $\mathrm{RTF}_j(i,j)$ or $\mathrm{RTF}_j(i+1,j)$. Given these two frames, the number of such packets is at most $2r_{j-1}T_j^k$. The fate of each such packet p is determined by occurance of the events $D(p,i',j+1)$, $G(p,i',j+1)$, $J(p,i',j+1)$, for at most T_j^k values of i'. The total number of relevant events is therefore at most $6r_{j-1}T_j^{2k}$. Instead of counting the number of packets which cross e and end up in $\mathrm{CTF}_\uparrow(i,j)$, we shall count the number of such events that occur. This is clearly an upper bound on the number of packets, since the dumping of a packet onto $\mathrm{CTF}_\uparrow(i,j)$ may be triggered by more than one such event. The conditional probabilities of these events are given in the above analysis. We now analyze the dependencies among them. To do that, we fix the frames $\mathrm{RTF}_{j+1}(i',j+1)$ and $\mathrm{CTF}_\downarrow(i',j+1)$ for the relevant values of i'. (We prove the desired large deviation bound under these conditions, and the claim in the lemma follows from averaging over the values of the fixed frames.) Notice that once these frames are fixed, the events for i' and for i'' where $i' \ne i''$ are independent. So consider a particular i'. The events $D(p,i',j+1)$, $G(p,i',j+1)$, $J(p,i',j+1)$ are determined by the congestion on the edges of p's path in $\mathrm{RTF}_{j+1}(i',j+1)$ during the refinement process. Therefore, each such event $\mathcal{E}(p,i',j+1)$ is independent of any other event $\mathcal{E}'(p',i',j+1)$, unless p crosses an edge e and p' crosses an edge e', such that the congestion on e depends on the congestion on e' (at some stage of the refinement process). In fact, $\mathcal{E}(p,i',j+1)$ is independent of any combination of such events, unless the condition holds for at least one of them. Now, at any stage in the refinement process, the congestions on e and e' are independent unless there is a path that crosses both. Such a path may exist only if it exists in either $\mathrm{RTF}_{j+1}(i', j+1)$ or $\mathrm{CTF}_{\downarrow}(i', j+1)$. Therefore, the number of packets p' such that an event $\mathcal{E}(p, i', j+1)$ may depend on an event $\mathcal{E}'(p', i', j+1)$ is bounded by $r_j^2 T_{j+1}^{4k} + 2\kappa r_j T_{j+1}^{2k+1/4} \leq 2r_j^2 T_{j+1}^{4k}$. Therefore, each event $\mathcal{E}(p, i', j+1)$ is independent of any combination of similar events which does not intersect a fixed subset of at most $6r_j^2 T_{j+1}^{4k}$ events. Notice that the expected number of these events that occurs is $\ll 1$. Therefore, using Lemma 2, we have that the probability that among at most $6r_{j-1}T_j^{2k}$ relevant events more than T_j^6 occur is $\leq 24er_j^2 T_{j+1}^{4k} 2^{-T_j^6/24er_j^2 T_{j+1}^{4k}} \ll 2^{-T_j^2}$.

The third claim follows from the second claim by summing up the bound for at most $2T_j^k$ edges crossed by p during $\mathrm{CTF}_{\uparrow}(i, j)$. ■

We are now ready for the

Proof (of Theorem 3). The probability that a packet p is lost is bounded by

$$\sum_i \left(\Pr[J(p, i, 0) \mid \mathrm{RTF}_0(i, 0)] + \Pr[G(p, i, 0) \mid \overline{\mathrm{CTF}}_0(i, 0)] \right)$$

$$\ll L(2^{-T_0} + e^{-T_0}),$$

as there are $\lceil L/T_0^k \rceil$ values that i can take.

So we turn to the length of the schedule. This length is determined by the number of regular track frames $\mathrm{RTF}_{j_{\max}}(i, j_{\max})$ and by the number of catchup track frames $\mathrm{CTF}_{j_{\max}}(i, j_{\max})$. Denote the former by $n_1(j_{\max})$ and the latter by $n_2(j_{\max})$. The length of the resulting schedule is at most $r_{j_{\max}} \beta^k (n_1(j_{\max}) + 2n_2(j_{\max})) = O(n_1(j_{\max}) + n_2(j_{\max}))$. $n_1(j_{\max})$ and $n_2(j_{\max})$ are given by the following recurrence:

$$\begin{aligned}
n_1(j) &= \left\lceil \frac{\ell_1(j)}{T_j^k} \right\rceil, \\
n_2(j) &= \left\lceil \frac{\ell_2(j)}{2T_j^k} \right\rceil, \\
\ell_1(j+1) &= \left(1 + \frac{1}{\sqrt{T_j}}\right) n_1(j) T_j^k, \\
\ell_2(j+1) &= \left(1 + \frac{1}{T_j}\right)\left(1 + \frac{\gamma}{2T_j}\right) n_2(j) 2T_j^k,
\end{aligned}$$

with the initial conditions $\ell_1(0) = L$, $\ell_2(0) = 2L$. The theorem easily follows. ■

4 Analysis of the Initial Refinement Process

In this section we show how to obtain a feasible schedule of length $O(C+D+\log^{1+\epsilon} N)$. Here ϵ is a constant. It can be made as small as we wish by increasing the constant hidden by the big-Oh notation (in a rather nasty way). The rest of this section is devoted to the proof of Theorem 5.

Let $K(e,i)$ denote the following event: In $\mathrm{RIF}_1(i,0)$ there exists a frame F of size $\ell \geq \log^{1/2k} N$ such that the number of packets that cross e during F is more than ℓ.

Lemma 15 *For all e, i, $\Pr[K(e,i) \mid RIF_0(i,0)] \leq 2^{-3\log^{1/2k} N}$.*

Proof. It is sufficient to check frames of size between $\log^{1/2k} N$ and $2\log^{1/2k} N$. Let F be such a frame. The number of packets that cross e during $\mathrm{RIF}_0(i,0)$ is at most $\log^{1+\epsilon} N$. the probability that after the insertion of the initial delays such a packet crosses e during F is at most $2\log^{1/2k} N/\lambda \log^{1+\epsilon} N$. Therefore, the expected number of packets that cross e during F is at most $\frac{2}{\lambda}\log^{1/2k} N$. By the Chernoff bound, the probability that more than $\log^{1/2k} N$ packets end up crossing e during F is less than $(\lambda/2e)^{-\log^{1/2k} N}$. Summing this probability over $1+\log^{1/2k} N$ possible sizes and over at most $(\lambda+1)\log^{1+\epsilon} N$ frames of any given size, completes the proof. ■

Let $L(p,i)$ denote the event that during $\mathrm{RIF}_1(i,0)$ packet p crosses an edge e for which $K(e,i)$ occurs.

Corollary 16 *For all p, i, $\Pr[L(p,i) \mid RIF_0(i,0)] \leq 2^{-2\log^{1/2k} N}$.*

Proof. Sum up the bound in Lemma 15 over at most $\log^{1+\epsilon} N$ edges that p crosses during $\mathrm{RIF}_0(i,0)$. ■

Lemma 17 *Fix i, $RIF_0(i,0)$. W.h.p., every connected component of $\mathcal{G}(i)$ has size at most $\log^6 N$.*

Proof. A packet p appears in $\mathcal{G}(i)$ either because $L(p,i)$ occurs or because p did not get scheduled by the main refinement algorithm (given $\mathrm{RIF}_1(i,1)$ as input). Denote the event that p ends up in $\mathcal{G}(i)$ by $\mathcal{E}(p)$. By Corollary 16 and Theorem 3, $\Pr[\mathcal{E}(p) \mid \mathrm{RIF}_0(i,0)] \leq 2^{-2\log^{1/2k} N} + (\lambda+1)(\log^{1+\epsilon} N)2^{-\log^{1/k} N} \leq 2^{-\log^{1/2k} N}$

Furthermore, $\mathcal{E}(p)$ and $\mathcal{E}(p')$ are clearly independent if p and p' are at distance 3 or more in the interference graph among the paths of $\mathrm{RIF}_0(i,0)$. So, consider the dependency graph *Dep* among the events $\mathcal{E}(p)$. The nodes of *Dep* are the events, and every independent set in *Dep* is a set of mutually independent events. Clearly, the maximum degree in *Dep* is at most $\log^{4(1+\epsilon)} N$.

Define a $(2,3)$-tree to be an independent set *Ind* of *Dep* with the following property: If we connect the nodes of *Ind* whose distance in *Dep* is 2 or 3, we get a connected graph over *Ind*. The number of different size-s $(2,3)$-trees of *Dep* is easily upper bounded at $N(e\log^{12(1+\epsilon)} N)^s$. Given a $(2,3)$-tree, the probability that all its nodes end up in $\mathcal{G}(i)$ is at most $2^{-s\log^{1/2k} N}$. Thus, if $s \gg \log N$, then the probability that any $(2,3)$-tree of size s has all its nodes in $\mathcal{G}(i)$ is smaller than any inverse polynomial in N. The lemma follows from the observation that every connected subgraph of *Dep* of size at least $\log^6 N$ contains a $(2,3)$-tree of size $\gg \log N$. (The greedy algorithm for maximal independent set finds one.) ■

Let $\mathcal{C}$ be a connected component of $\mathcal{G}(i)$ of size at most $\log^6 N$. Let $\mathcal{H}_q$ denote the event that the first q tries at refining $\mathrm{CIF}^1_j(s,\mathcal{C},j)$ failed.

Lemma 18 *(i) For all $j < j_{\max}$, for all $\mathcal{C}, s, q$, $\Pr[\mathcal{H}_q \mid \mathcal{H}_{q-1}] \le 2^{-\log^{1-(j+1)\epsilon} N}$; and (ii) For $j = j_{\max}$, for all $\mathcal{C}, s, q$, $\Pr[\mathcal{H}_q \mid \mathcal{H}_{q-1}] \le 2^{-\log^{2\delta} N}$.*

Proof. *Claim i:* Consider an edge e. In $\mathrm{CIF}^1_j(s,\mathcal{C},j)$ there are at most $\log^{1-j\epsilon} N$ paths that cross e. The expected number of paths that cross e during any particular frame of length $\log^{1-(j+1)\epsilon} N$ in $\mathrm{CIF}^1_{j+1}(q,s,\mathcal{C},j)$ is at most $\frac{1}{\lambda}\log^{1-(j+1)\epsilon} N$. The probability that this number exceeds $\log^{1-(j+1)\epsilon} N$ is at most $\exp(-\log^{1-(j+1)\epsilon} N)$ by the Chernoff bound. Summing this bound over at most $(\lambda+1)\log^{1-j\epsilon} N$ relevant frames, and over at most $\log^7 N$ edges affected by $\mathcal{C}$ completes the proof for this case.
Claim ii: Theorem 3 guarantees that any individual packet is excluded from the final schedule of $\mathrm{CIF}^1_{j_{\max}}(s,\mathcal{C},j_{\max})$ with probability at most $\exp(-\log^{2\delta} N)$. Summing over at most $\log^6 N$ packets that appear in $\mathcal{C}$ completes the proof. ■

Assuming $\mathcal{C}$ is as above, we now have

Lemma 19 *(i) For all $j < j_{\max}$, for all $\mathcal{C}$, $\sum_s q_{\max}(s,\mathcal{C},j) < c_4 \log^{(j+1)\epsilon} N$ w.h.p., where c_4 is an absolute constant whose increase raises the high probability guarantee. (ii) For $j = j_{\max}$, for all $\mathcal{C}$, $\sum_s q_{\max}(s,\mathcal{C},j_{\max}) < c_4 \log^{1-2\delta} N$ w.h.p.*

Proof. The proof of both claims is similar, so we show in detail only the first claim.

Notice that $q_{\max} > 0$ only for values of s such that $\mathrm{CIF}^1_j(s,\mathcal{C},j)$ is not empty. The number of such paths is at most $w = (\lambda+1)^j \log^{(j+1)\epsilon} N$. The process of refinement of these frames can be represented as a time homogeneous Markov chain $\mathcal{M}$ over a state space with $w+1$ states. We denote the states by $1, 2, \ldots, w, \mathit{fin}$. The possible transitions are: In each state $1, 2, \ldots, w$ the chain stays there with probability $\Pr[\mathcal{H}_q \mid \mathcal{H}_{q-1}] \le 2^{-\log^{1-(j+1)\epsilon} N}$; otherwise it moves on to the next state (the state following w is *fin*). In *fin*, there is a self-loop with probability 1. Now $\sum_s q_{\max}(s,\mathcal{C},j)$ is exactly the number of steps it takes to get from the initial state 1 to the final state *fin*. It is not hard to show that the number of steps needed is at most $c_4 \log^{(j+1)\epsilon} N$ w.h.p., for a sufficiently large constant c_4. Here's a simple argument that proves this claim: w.h.p., we do not remain in any individual state (excluding *fin*) for more than $O(\log^{(j+1)\epsilon} N)$ steps. Therefore, excluding runs whose total probability is negligible, there are at most $w^{O(\log^{(j+1)\epsilon} N)} \ll N$ different runs. Now w.h.p. a run does not use more than $O(\log^{(j+1)\epsilon} N)$ stationary transitions in all states. ■

We conclude that

Lemma 20 *For all $\mathcal{C}$ of size at most $\log^6 N$, the length of the final schedule for the packets in $\mathcal{C}$ is $O(\log^{1+\epsilon} N)$ w.h.p.*

Proof. The length of $\mathrm{CIF}_{j_{\max}}(\mathcal{C}, 0)$ is at most $\sum_j c_4(\log^{(j+1)\epsilon} N)(\lambda + 1)(\log^{1-j\epsilon} N) = O(\log^{1+\epsilon} N)$ w.h.p., following to Lemma 19. In applying the main refinement algorithm to $\mathrm{CIF}_{j_{\max}}(\mathcal{C}, 0)$, according to Theorem 3, each non-empty frame of size $\log^{2k\delta} N$ is expanded by a constant factor. Following Lemma 19 again, this happens at most $c_4 \log^{1-2\delta} N$ times w.h.p. Therefore, the total addition to the final schedule due to the use of the main refinement algorithm is $O(\log^{2k\delta} N \log^{1-2\delta} N) = O(\log^{1+\epsilon} N)$ w.h.p., so the lemma follows. ∎

Now we may have the

Proof (of Thereom 5). The first step of the switching algorithm (inserting initial delays as in Lemma 1) expands the trivial $\max\{C, D\}$ schedule by a constant factor at most. Let ℓ_0 denote the length of this (infeasible) schedule. Now, if n_0 denotes the number of frames $\mathrm{RIF}_0(i, 0)$, then the length of the resulting feasible schedule is at most $n_0\left((\lambda+1)\log^{1+\epsilon} N + O(\log^{1+\epsilon} N)\right)$ w.h.p., where the second term comes from Lemma 20. As $n_0 = \left\lceil \ell_0 / \log^{1+\epsilon} N \right\rceil$, the theorem follows. ∎

References

[1] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, F.T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *FOCS '96*.

[2] M. Andrews, A. Fernández, M. Harchol-Balter, F.T. Leighton, and L. Zhang. General dynamic routing with per-packet delay guarantees of $O($ distance $+ 1 /$ session rate $)$. Unpublished manuscript, February 1997.

[3] ATM UNI Specification Version 3.0, ATM Forum, September 1993.

[4] J. Beck. An Algorithmic Approach to the Lovász Local Lemma. *Random Structures and Algorithms*, 2(4):367-378, 1991.

[5] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D.P. Williamson. Adversarial queueing theory. In *Proc. STOC '96*.

[6] A.Z. Broder, A.M. Frieze, and E. Upfal. A general approach to dynamic packet routing with bounded buffers. In *STOC '96*.

[7] R. Cypher, F. Meyer auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. In *STOC '96*.

[8] I. Gopal. Multimedia Networking: Applications and Challenges. In *PODC'94*.

[9] F.T. Leighton. Methods for message routing in parallel machines. Invited survey paper in *STOC '92*.

[10] 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 (preliminary version in *FOCS '88*).

[11] 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.

[12] F.T. Leighton, B.M. Maggs, and A.W. Richa. Fast algorithms for finding O(congesting+dilation) packet routing schedules. Technical report CMU-CS-96-152, School of Computer Science, Carnegie-Mellon University, 1996. *Combinatorica*, to appear.

[13] F. Meyer auf der Heide and B. Vöcking. A packet routing protocol for arbitrary networks. In *STACS '95*.

[14] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Trans. Networking*, 1(3):344–357, 1993.

[15] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple-node case. *IEEE/ACM Trans. Networking*, 2(2):137–150, 1994.

[16] S. Plotkin. Competitive routing of virtual circuits in ATM networks. *IEEE J. Selected Areas in Communications*, 1995.

[17] Y. Rabani and É. Tardos. Distributed packet switching in arbitrary networks. In *STOC '96*.

[18] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria. In these proceedings.