General Dynamic Routing with Per-Packet Delay Guarantees of $O($ distance + 1 / session rate $)$ *

Matthew Andrews† Antonio Fernández‡ Mor Harchol-Balter§ Tom Leighton
Lisa Zhang¶

Laboratory for Computer Science, MIT.
{andrews, anto, harchol, ftl, ylz}@theory.lcs.mit.edu

Abstract

A central issue in the design of modern communication networks is that of providing performance guarantees. This issue is particularly important if the networks support real-time traffic such as voice and video. The most critical performance parameter to bound is the delay experienced by a packet as it travels from its source to its destination.

We study dynamic routing in a connection-oriented packet-switching network. We consider a network with arbitrary topology on which a set of sessions is defined. For each session i, packets are injected at a rate r_i to follow a predetermined path of length d_i. Due to limited bandwidth, only one packet at a time may advance on an edge. Session paths may overlap subject to the constraint that the total rate of sessions using any particular edge is less than 1.

We address the problem of scheduling the sessions at each switch, so as to minimize worst-case packet delay and queue buildup at the switches. We show the existence of an asymptotically-optimal schedule that achieves a delay bound of $O(1/r_i + d_i)$ with only constant-size queues at the switches. We also present a simple distributed algorithm that, with high probability, delivers every session-i packet to its destination within $O(1/r_i + d_i \log(m/r_{\min}))$ steps of its injection, where $r_{\min}$ is the minimum session rate, and m is the number of edges in the network. Our results can be generalized to (leaky-bucket constrained) bursty traffic, where session i tolerates a burst size of b_i. In this case, our delay bounds become $O(b_i/r_i + d_i)$ and $O(b_i/r_i + d_i \log(m/r_{\min}))$, respectively.

1 Introduction

1.1 Motivation

Motivated by the need for quality-of-service guarantees, network designers today offer *connection-oriented* service in many networks, e.g. ATM (Asynchronous Transfer Mode) networks. In this medium, a user requests a particular share of the bandwidth and injects a stream of packets along one particular session at the agreed-upon rate. An important consequence of the user's predictability is that the network can, in return, guarantee the user an *end-to-end delay bound*, i.e. an upper bound on the time that any packet takes to move from its source to its destination. In order to provide this delay guarantee, the network must determine how to schedule the packets that contend for the same edge simultaneously. Apart from delay bounds, it is also important to guarantee small queues at each switch due to limited buffer size. In this paper we show, for the first time, that it is possible to design such a schedule that guarantees asymptotically-optimal per-session delay bounds as well as small queues.

1.2 Model and problem

Consider a network $\mathcal{N}$ of arbitrary topology and a set of sessions defined on this network. A session i is associated with a source node, a destination node, and a simple path from the source to the destination. (A path is *simple* if it uses each edge at most once.) Packets are injected to the network $\mathcal{N}$ in sessions. A packet injected in session i enters the system at the

*This research is supported by Army grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246. A full version of the paper may be found at `ftp://theory.lcs.mit.edu/pub/groups/algorithms`.

†Supported by NSF contract 9302476-CCR.

‡Supported by the Spanish Ministry of Education. Current address: Departamento de Arquitectura y Tecnología de Computadores, Universidad Politécnica de Madrid, Spain, anto@eui.upm.es.

§Supported by NSF Postdoctoral Fellowship in the Mathematical Sciences.

¶Supported by an NSF graduate fellowship.

source node of i, traverses the path associated with i, and then is absorbed at its destination. The length d_i is the number of edges on the path from the source to the destination of session i.

Each session i has an associated injection rate r_i. This rate constrains the injection of new packets from the session so that, during any interval of t consecutive steps, at most $tr_i + 1$ packets can be injected in session i, for any t.

We assume that all packets have the same size and all edges have the same bandwidth. We also assume a synchronized store-and-forward routing, where at each step at most one packet can traverse each edge. When two packets simultaneously contend for the same edge, one packet has to wait in a queue. During the routing, packets wait in two different kinds of queues. After a packet has been injected, but before it leaves its source, the packet is stored in an *initial queue*. Once the packet has left its source, during any time it is waiting to traverse an edge, the packet is stored in an *edge queue*. The *end-to-end delay* (delay for short) for a packet is the total time from the packet injection until it reaches its destination. This includes the total time the packet spends waiting in both types of queues, plus the time it spends traversing edges.

Our goal is to minimize both the end-to-end delay for each packet and the size of all edge queues. In order to achieve delay guarantees and bounded queue sizes, it is necessary to require that, for all edges e, the sum of the rates of the sessions that use edge e is at most 1. Throughout, we shall assume that the sum of the rates of the sessions using any edge e is at most $1 - \varepsilon$, for a constant $\varepsilon \in (0, 1)$.

Our paper focuses on the problem of timing the movements of the packets along their paths. A *schedule* specifies which packets move and which packets wait in queues at each time step. We address the problem of finding a schedule which, for each session, guarantees an asymptotically-optimal delay bound for packets of that session, while maintaining constant-size edge queues.

In this paper most of the schedules obtained are *template based*. The schedule defines a fixed *template* for each edge in advance. A template of size M is a wheel with M slots, each of which contains at most one token. Each token is affiliated with some session. The wheel spins at the speed of one slot per time step. A session-i packet can traverse the edge if and only if a session-i token appears. For each session-i token, the session-i packet that uses it will be the one that has been waiting to cross the edge for the longest amount of time, i.e. the session-i packets use the session-i tokens in a First-Come-First-Served manner. The template size and associated tokens do not change over time.

1.3 Lower Bounds

Observe that d_i is always a lower bound on the delay for session i, since every session-i packet has to cross d_i edges. It is also easy to see that $\Omega(1/r_i)$ is an existential lower bound. For instance, consider n sessions, all of which have the same rate $r = (1-\varepsilon)/n$ and the same initial edge e. If a packet is injected in each session simultaneously, one of the packets requires $n = \Omega(1/r)$ steps to cross e.

Furthermore, for *any* given set of sessions, $\Omega(1/r_i)$ is a lower bound for some session i in template-based schedules. It can be proven that, in the template for an edge e where $\sum_e r_i = 1-\varepsilon$, there exist two session-i tokens separated by at least $(1-\varepsilon)/r_i$ slots, for some i. Then, an adversary can make sure a session-i packet arrives just after the first token has passed, thereby forcing the packet to wait $\Omega(1/r_i)$ steps.

If the schedule is not restricted to being template-based, the scheduler is more powerful. The scheduler does not have to decide on a fixed schedule in advance, but rather can make a new decision at each step, based on seeing the adversary's injections. In this case it is unknown if for *any* given set of sessions $\Omega(1/r_i)$ is a lower bound.

1.4 Previous Work

The problem of dynamic packet routing in the above setting is well studied. Until recently, the best delay bound known was $O(d_i/r_i)$ for packets of session i. It is tempting to believe that this is the best possible delay bound, since a session-i packet may need to wait $O(1/r_i)$ steps to cross each of the d_i edges on its route. However, this leaves a large gap between the upper bound of $O(d_i/r_i)$ and the lower bound of $O(1/r_i + d_i)$. The recent work seeks to close this gap.

In 1990, Demers, Keshav and Shenker [6] proposed a widely-studied routing algorithm called Weighted Fair Queueing (WFQ). WFQ is a packetized approximation of the idealized fluid model algorithm Generalized Processor Sharing (GPS). WFQ is simple and distributed. This same algorithm was proposed independently by Parekh and Gallager [11, 12] in 1992 under the name of Packet-by-Packet Generalized Processor Sharing (PGPS). Parekh and Gallager prove that the algorithm has an end-to-end delay guarantee of $2d_i/r_i$ [12, page 148] in the case when all packets have the same size.

In their 1996 paper, Rabani and Tardos [13] produce an algorithm that routes every packet to its destination with probability $1-p$ in time $O(1/r_{\min}) +$

$(\log^* p^{-1})^{O(\log^* p^{-1})} d_{\max} + poly(\log p^{-1})$, where $r_{\min} = \min_i r_i$ and $d_{\max} = \max_i d_i$. Ostrovsky and Rabani improve the bound to $O(1/r_{\min} + d_{\max} + \log^{1+\varepsilon} p^{-1})$ [10]. These bounds are not session-based, meaning that if one session has a small rate or a long path then the delay bounds for all sessions will suffer. The algorithms of Rabani et al. are distributed, where knowledge of the entire network is not assumed, but each packet carries some information.

The main technique of Rabani et al. is based on "delay-insertion". The intuition here is that if each packet receives a large random initial delay, then the packets are sufficiently spread out to ensure that they only need to wait $O(1)$ steps at each successive edge rather than $O(1/r_i)$ steps. This delay-insertion technique is used extensively by Leighton et al. in [8, 9] in the context of static routing. (In the *static* routing problem, all packets are present in the network initially.) Since our main result employs many techniques from [8], we give a detailed summary of [8] in Section 3.1.

A contrasting model, the *connectionless adversarial queueing model*, is also much studied, e.g. [3, 1]. Here the paths on which packets are injected can change over time giving the adversary more power. In the adversarial queueing model the best delay bound known is polynomial in the maximum path length [1].

1.5 Our Results

We first provide a randomized, distributed scheduler that achieves a delay bound of $O(1/r_i + d_i \log(m/r_{\min}))$ and a bound on the queue size of $O(\log(m/r_{\min}))$, where m is the number of edges in the network and $r_{\min} = \min_i r_i$. While this bound is not optimal, it nevertheless conveys some intuition for our main result.

The main contribution of this paper is an asymptotically-optimal schedule. We prove that a schedule exists for the dynamic routing problem such that the end-to-end delay of each session-i packet is bounded by $O(1/r_i + d_i)$. [1] Our result improves upon previous work in several aspects.

- We provide a session-based delay guarantee. That is, packets from sessions with short paths and high injection rates reach their destinations fast. This is a big improvement over the previous bounds, which are stated in terms of $r_{\min} = \min_i r_i$ and $D = \max_i d_i$. We also guarantee that every packet always reaches its destination within the delay bound, without dropping any packets.
- We guarantee constant-size edge queues. This is interesting because edge queues are much more expensive than initial queues in practice.
- A consequence of our result is a packet-based bound, which improves upon the $O(c+d)$ bound in [8] for the static problem. (See Section 3.1 for the problem and parameter definitions.) We show that if packet p_i follows a route P_i, then p_i can be routed to its destination within $O(c_i + d_i)$ steps, where c_i is the maximum congestion along P_i and d_i is the number of edges on P_i. This result trivially follows from our result by creating a different session i for each packet p_i, and defining $r_i = (1-\varepsilon)/c_i$.

The asymptotically-optimal schedule is template-based. Even if the computation of the schedule is time-consuming, it only needs to be done once. Packets can then be scheduled indefinitely as long as the sessions do not change.

Leaky-bucket injection model Our results above can be generalized to bursty traffic streams that are *leaky-bucket regulated*. Here, each session i has a maximum burst size (or bucket size) of $b_i \geq 1$ and an average arrival rate of r_i. During any t consecutive time steps at most $r_i t + b_i$ session-i packets are injected. Leaky-bucket regulated traffic is widely used in the literature, e.g. [4, 5, 7, 11, 12, 14].

Leaky-bucket regulated injections allow traffic shaping. When session-i packets are injected, they first enter the session-i bucket at the source. These packets then leave the bucket one at a time at the rate of r_i. In this way, the end-to-end delay is separated into two components, delay in the bucket and delay in the network. Since delay in the bucket is at most b_i/r_i, the end-to-end delay is increased by at most b_i/r_i steps, and the size of the edge queues is unchanged.

The rest of the paper is divided into sections as follows. We first describe a simple distributed scheduler that has a delay bound of $O(1/r_i + d_i \log(m/r_{\min}))$ in Section 2. In Section 3, we overview the main techniques employed to achieve a bound of $O(1/r_i + d_i)$ and constant-size edge queues. The proof details are presented in the full version of the paper.

2 A Preliminary Result

Before presenting our main result, we first present a simple centralized schedule and a simple distributed

[1] In this paper, we concentrate on proving the existence of such a schedule. However, the proof can be made constructive using ideas of Leighton, Maggs and Richa [9] that are based on Beck's algorithm [2]. For details, see [15].

schedule that achieve delay bounds of $O(1/r_i + d_i \log(m/r_{\min}))$ with high probability. In addition, the centralized schedule has a maximum queue size of $O(\log(m/r_{\min}))$ with high probability. These preliminary results are substantially simpler to prove because of the relaxed bounds on delay and queue sizes. Nevertheless, they illustrate the basic ideas of the main result.

In this section, for ease of presentation, we omit floors and ceilings where they are necessary. We shall also assume that $1/r_i$ is an integer for all i and there is a constant k such that $k/r_{\min}$ is a multiple of $1/r_i$ for all i.[2]

2.1 Template-Related Definitions

Throughout much of this paper we are considering template-based schedules. Specifying a template-based schedule reduces to the problem of assigning tokens to the slots on the templates. Our usual strategy is to assign tokens in *token sequences*. A token sequence for session i consists of d_i tokens, one for each edge along session i. For each method of assigning tokens we initially provide bounds on delay and queue size for a *token-sequence schedule*. In this schedule, packets use tokens for the initial edge in a First-Come-First-Served manner. However, once a packet has used one token from a particular token sequence then for subsequent edges it *only* uses tokens from that token sequence. Below we state a theorem showing that token-sequence bounds give us bounds for the corresponding *template-based schedule* in which session-i packets use session-i tokens in a First-Come-First-Served manner on all edges. The proof is contained in the full version. If no ambiguity arises, we shall often refer to scheduling packets rather than assigning tokens in token sequences.

Let $\kappa_1, \ldots, \kappa_{d_i}$ be a token sequence for session i. If κ_{j+1} appears x_j steps after κ_j, then x_j is the *token lag* for these two tokens and $\sum_{j=1}^{d_i-1} x_j$ is the end-to-end delay for this token sequence. The token sequences for each session i form a partition of all the session-i tokens. In the following we state that, in any template-based schedule, bounding the delay for token sequences is sufficient to bound the packet delays and that bounding the token lag is sufficient to bound the edge queues.

Theorem 1 *If the end-to-end delay for each session-i token sequence is bounded by X, then in the* template-based schedule, *each session-i packet reaches its destination within X steps after it obtains an initial token. If the token lag is bounded by x for* all *token sequences for* all *sessions, then in the* template-based schedule *the edge queue size is also bounded by x.*

Note that in the template-based schedule the session-i packets reach their destination *in order*.

2.2 A Simple Centralized Schedule

We now describe the centralized schedule which we call TEMPLATE. Let $M = k/r_{\min}$ where k is the constant chosen above. Each template has size M. We first place $r_i M$ *initial tokens* on the template for the first edge of session i, spaced $1/r_i$ slots apart.

Lemma 2 *Each session-i packet will obtain a session-i token at most $2/r_i$ steps after its injection.*

Proof: Suppose that packet p is injected at time t but has not obtained an initial token by time $t + 2/r_i + 1$. Let t' be the last time before $t + 2/r_i + 1$ that there were no session-i packets waiting for initial tokens. (Note that $t' < t$.) Between times t' and t at most $(t - t')r_i + 1$ session i packets are injected. However, at least $(t - t' + 2/r_i)r_i - 1 = (t - t')r_i + 1$ initial tokens for session i appear between times t' and $t + 2/r_i$. By the definition of t', each of these tokens was used by a packet. Hence none of the packets injected between t' and t (and in particular packet p) can still be waiting for an initial token at time $t + 2/r_i + 1$. $\square$

Once the initial session-i tokens are placed, we delay each of them by an amount chosen uniformly and independently at random from $[L+1, L+1/r_i]$, where $L = \frac{\alpha}{2}\log(mM)$ and α is a constant. The intuition is that the random delays would spread out the tokens. After the tokens have been delayed we can be sure that each packet obtains an initial token within $L + 3/r_i$ steps. We now create the token sequences. (Recall the definition of token sequence from Section 2.1.) For every session-i token a placed in the template corresponding to the jth edge, we place a session-i token b on the template corresponding to the $(j+1)$st edge such that b appears exactly $2L$ steps after a.

We observe that two different session-i token sequences have their initial tokens in different slots, and therefore two session-i tokens can never be in the same slot. Unfortunately, tokens from different sessions may be placed in one slot, which would cause packets from

---

[2] In the main text, we choose to assume the existence of k so as to avoid obscuring the main ideas. If there is no such constant k, we can always show the existence of $\hat{r}_i$ for each session i such that the following holds. i) $\hat{r}_i$ is a fraction s_i/ℓ_i, where s_i, ℓ_i are integers and $\ell_i = \Theta(1/r_i)$ is a power of 2; ii) $r_i \le \hat{r}_i$; iii) $\sum_i \hat{r}_i \le 1 - 5\varepsilon/6$ for all edges. Hence, if we choose the template size $M = \max_i \ell_i$, then $M = \Theta(1/r_{\min})$ is a multiple of all ℓ_i's. Then in the token placement process of Section 2.2, we place s_i tokens in one slot every ℓ_i slots.

different sessions to cross the same edge simultaneously. The following lemma shows that the tokens are not clustered to any great extent.

Lemma 3 *There are at most L tokens in any consecutive L slots on any template with probability $1 - 1/(mM)$, where $L = \frac{\alpha}{2}\log(mM)$ and α is a sufficiently-large constant.*

Proof: Since the initial tokens for session i are spaced $1/r_i$ apart and each is delayed by an amount chosen independently and uniformly at random from $[L+1, L+1/r_i]$, the expected number of session-i tokens in a single slot is r_i. For a particular interval of L consecutive slots on a particular template, let the random variable X equal the number of tokens in these slots. By linearity of expectations, $E[X] \leq \sum_i r_i L \leq (1-\varepsilon)L$. (Note also that $\sum_i \lfloor r_i L \rfloor \leq X \leq \sum_i \lceil r_i L \rceil$.) Whether or not a token lands in these L slots is a Bernoulli event. Since the delays to the initial tokens are chosen independently and all session paths are simple, these Bernoulli events are independent. Since $E[X] \leq (1-\varepsilon)L$, we have the following by a Chernoff bound.

$$\begin{aligned} \Pr[\, X > L \,] &\leq \Pr[\, X > (1+\varepsilon)(1-\varepsilon)L \,] \\ &\leq e^{-\varepsilon^2(1-\varepsilon)L/3}. \end{aligned}$$

In m templates there are at most mM intervals of L consecutive slots. Therefore, by a union bound the probability that more than L tokens appear in *any* L consecutive slots is bounded by,

$$\begin{aligned} mM \Pr[\, X > L \,] &\leq mMe^{-\varepsilon^2(1-\varepsilon)L/3} \\ &= mMe^{-\varepsilon^2(1-\varepsilon)\alpha\log(mM)/6}. \end{aligned}$$

By choosing a sufficiently large constant α, we can bound the above probability by $1/(mM)$. □

Lemma 3 is not sufficient to guarantee one token per slot. We solve this problem by partitioning each template into intervals of L consecutive slots[3] and "smoothing out" each interval as follows. We take the at most L tokens from these slots and rearrange them arbitrarily so that there is at most one token in each slot. We have,

Lemma 4 *Consider a packet p. Let $\mathcal{K}^p$ be the token sequence that contains the initial token used by p* before *the smoothing process. Let κ_j^p be the token for the jth edge in this token sequence. Then,* after *the smoothing process, the packet p can use the token κ_j^p to cross its jth edge. Therefore, p reaches its destination within $O(1/r_i + d_i \log(m/r_{\min}))$ steps of its injection.*

Proof: It is sufficient to show that, after the smoothing process, token κ_1^p (the initial token) appears after the injection of p and κ_{j+1}^p appears after κ_j^p. A token is shifted by at most $L-1$ steps by the smoothing process. Before the smoothing, κ_1^p appears at least L steps after the injection of p and κ_{j+1}^p appears exactly $2L$ steps after κ_j^p. The lemma follows. □

Hence, we have presented a schedule TEMPLATE that assigns tokens on the templates with at most one token per slot with high probability. Note that if the first execution of TEMPLATE assigns more than one token per slot, TEMPLATE can be executed again until the condition of one token per slot is satisfied. We have already bounded the delay experienced by packets. We now show that the queue size is small.

Lemma 5 *Every session-i packet waits at most $O(\log(m/r_{\min}))$ steps to cross each edge. Therefore, the queue size is $O(\log(m/r_{\min}))$.*

Proof: Suppose that packet p uses token κ_j^q from token sequence $\mathcal{K}^q$ to cross its jth edge. Then packet p can use κ_{j+1}^q to cross its $j+1$st edge. Thus token κ_{j+1}^q appears at most $4L = O(\log(m/r_{\min}))$ steps later than κ_j^q. The result follows. □

Therefore,

Theorem 6 *With high probability, the randomized centralized schedule* TEMPLATE *has a delay bound of $O(1/r_i + d_i \log(m/r_{\min}))$ and a queue size of $O(\log(m/r_{\min}))$.*

2.3 A Simple Distributed Schedule

The above scheme TEMPLATE is centralized since the session-i tokens on one template are dependent on the previous template. However, it suggests the following simple *distributed* strategy for scheduling packets so as to achieve small delay. As with the centralized schedule, we place initial tokens on the first edge of session i and then delay each token by an amount chosen independently and uniformly at random from $[L+1, L+1/r_i]$. Suppose that a packet now has its initial token at time T. Then for the kth edge on this packet's path the packet is given a "deadline" of $T + 2L(k-1) + L$, where $L = \frac{\alpha}{2}\log(mM)$. Whenever two or more packets contend for the same edge simultaneously, the packet with the earliest deadline

[3] Here we assume M is a multiple of L. This can be achieved by choosing M and L to be powers of 2. (See the previous footnote.)

moves. We call this scheme EARLIEST-DEADLINE-FIRST (EDF).

Lemma 7 *For any edge, there are at most L deadlines in any consecutive L time steps with probability at least $1-1/(mM)$, where $L=\frac{\alpha}{2}\log(mM)$ and α is a sufficiently large constant.*

Proof: The proof is almost identical to that of Lemma 3. □

Lemma 8 *If for any edge, there are at most L deadlines in any consecutive L time steps, then each packet crosses every edge by its deadline.*

Proof: For the purpose of contradiction, let D be the first deadline that is missed. This implies that all deadlines earlier than D are met. Let p be the packet that misses deadline D for edge e. Since packet p meets its previous deadlines, p must have crossed its previous edge by time $D-L$, or else e must be p's first edge and p must have obtained its initial token by time $D-L$. Hence, at every time step from time $D-L+1$ to D packet p is held up by another packet with deadline no later than D. Furthermore, these deadlines must be later than $D-L$ since all deadlines earlier than D are met. Therefore, at least $L+1$ packets have deadlines for edge e from time $D-L+1$ to D. This contradicts the assumption of the lemma. □

Lemmas 7 and 8 imply,

Theorem 9 *With high probability, the randomized distributed schedule* EARLIEST-DEADLINE-FIRST *achieves a delay bound of $O(1/r_i + d_i\log(m/r_{\min}))$.*

Note that EDF does not generate a template for each edge. Instead, it generates a list of r_iM initial deadlines for the first edge of session i, and gives them in order to the session-i packets injected.

3 Summary of the Main Result

Our main result for the dynamic routing problem parallels an earlier result on static routing. In Section 3.1 we review the method used for solving the static case, and in Section 3.2 we give an overview of the additional complexities that need to be addressed in the dynamic case.

3.1 A Bound of $O(c+d)$ for Static Routing

Leighton, Maggs and Rao consider the static routing problem for arbitrary networks in [8]. For static routing, all packets are present in the network initially. Each packet is associated with a source, a destination, and a route. The *congestion* on each edge is the total number of routes that require that edge, and the *dilation* of a route is the number of edges on the route. Leighton et al. show that for any set of routes with maximum congestion c (over all edges) and maximum dilation d (over all routes), there is a schedule of length $O(c+d)$ and edge queue size $O(1)$. In this schedule, at most one packet traverses each edge at each time step. A packet waits $O(c+d)$ steps initially before leaving its source, and it waits $O(1)$ steps to cross each edge thereafter.

We summarize here the techniques in [8]. The strategy for constructing an efficient schedule is to make a succession of *refinements* to an initial schedule $\mathcal{S}^{(0)}$. In $\mathcal{S}^{(0)}$, each packet moves at every step until it reaches its destination. This schedule has length d, but as many as c packets may traverse the same edge at the same step. Each refinement brings the schedule closer and closer to the requirement that at most one packet uses one edge per time step.

A *T-frame* is a time interval of length T. The *frame congestion*, C, in a T-frame is the largest number of packets that use any edge during the frame. The *relative congestion* in a T-frame is the ratio C/T. The frame congestion (resp. relative congestion) on an edge e during a T-frame is defined to be the frame congestion (resp. relative congestion) associated with edge e.

It is obvious that the initial schedule $\mathcal{S}^{(0)}$ has relative congestion at most 1 for any c-frame. A *refinement* transforms a schedule $\mathcal{S}^{(q)}$ with relative congestion at most $c^{(q)}$ in any frame of size $I^{(q)}$ or larger into a schedule $\mathcal{S}^{(q+1)}$ with relative congestion at most $c^{(q+1)}$ in any frame of size $I^{(q+1)}$ or larger. The resulting frame size $I^{(q+1)}$ is much smaller than $I^{(q)}$, whereas the relative congestion $c^{(q+1)}$ is only slightly bigger than $c^{(q)}$. In particular, $I^{(q+1)}=\log^5 I^{(q)}$ and $c^{(q+1)}=(1+o(1))c^{(q)}$. After a series of $O(\log^* c)$ refinements, a schedule $\mathcal{S}^{(\zeta)}$ is obtained where the relative congestion is $O(1)$ for any $O(1)$-frame. A final schedule, in which at most one packet at a time crosses each edge, can be constructed by replacing each step of $\mathcal{S}^{(\zeta)}$ by a constant number of steps. Each refinement is achieved by inserting delays to the packets. It is the central issue in [8] to show that a set of delays always exists satisfying the criteria in Table 1.

3.2 A Bound of $O(1/r_i+d_i)$ for Dynamic Routing

Our result for the dynamic routing problem is parallel to that in [8]. For an arbitrary network where paths (sessions) are defined, we show that there is a schedule such that every session-i packet reaches its destination within $O(1/r_i+d_i)$ steps of its injection,

Schedule	Frame size	Relative congestion
$\mathcal{S}^{(q)}$	$I^{(q)}$	$c^{(q)}$
Refinement	$\log^5 I^{(q)}$	$(1+o(1))c^{(q)}$
$\mathcal{S}^{(q+1)}$	$I^{(q+1)}$	$c^{(q+1)}$

Table 1: Frame-refinement for static routing in [8].

where r_i and d_i are the injection rate and path length for session i, respectively. A session-i packet waits $O(1/r_i + d_i)$ steps initially before leaving its source, and it waits $O(1)$ steps to cross each edge afterwards.

To achieve a session-based, end-to-end delay bound of $O(1/r_i + d_i)$ for our dynamic routing problem, we adopt the general approach in [8]. However, there are three major problems in transforming the solution for the static problem into a solution for the dynamic problem. In the following we present these three problems and their solutions.

Problem 1: Infinite time

In [8] all the packets to be scheduled are present initially. In the dynamic model, packets are injected over an infinite time line. We would like to partition the infinite time line into finite time intervals which can be scheduled independently of each other. We divide time into intervals of length $\mathcal{T}$, where $\mathcal{T} = \Theta(1/r_{\min} + d_{\max})$. We then independently schedule the time intervals $[0, \mathcal{T})$, $[\mathcal{T}, 2\mathcal{T})$, $[2\mathcal{T}, 3\mathcal{T})$, etc.

We associate each session i with a quantity $\mathcal{T}_i = \Theta(1/r_i + d_i)$. For any integer $k \geq 0$ consider all the session-i packets that are injected during interval $[k\mathcal{T} - \mathcal{T}_i, (k+1)\mathcal{T} - \mathcal{T}_i)$. We provide a schedule in which all these packets leave their sources no earlier than time $k\mathcal{T}$ and reach their destinations before time $(k+1)\mathcal{T}$. (See Figure 1.) From now on, we concentrate on scheduling the arrivals that would be serviced during interval $[\mathcal{T}, 2\mathcal{T})$.

Problem 2: Session-based delay guarantees

Once we restrict ourselves to the interval $[\mathcal{T}, 2\mathcal{T})$, it seems that the dynamic routing problem is similar to the static problem. However, we cannot simply proceed with the successive refinements as in Section 3.1, since some sessions need tighter delay bounds than others. Session-i packets can only tolerate a delay proportional to $1/r_i + d_i$. We group sessions according to their associated $1/r_i + d_i$ value. We start by inserting delays to sessions having large values of $1/r_i + d_i$,

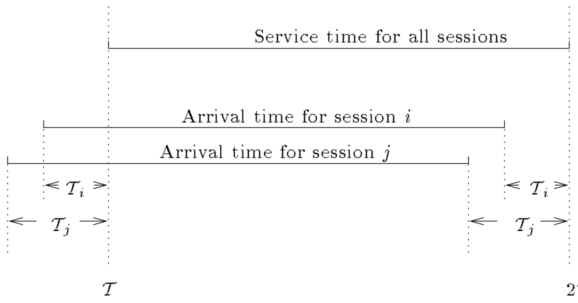

Figure 1: All the session-i packets that arrive during $[k\mathcal{T} - \mathcal{T}_i, (k+1)\mathcal{T} - \mathcal{T}_i)$ are serviced during $[k\mathcal{T}, (k+1)\mathcal{T})$. In this figure, $k = 1$.

reducing the frame size and bounding the relative congestion. When the frame size becomes small enough, sessions with smaller $1/r_i + d_i$ join in.

More precisely, we introduce the concept of *integral* and *fractional* sessions. When session i is *integral*, packets of size 1 are injected at rate r_i. When session i is *fractional*, a packet of size $\hat{r}_i$ is injected at every time step, where $\hat{r}_i$ is a value slightly larger than r_i. A packet from a fractional session always crosses one edge at a time, whether or not other packets are crossing the edge at the same time. Therefore, a fractional packet from session i always contributes exactly $\hat{r}_i$ to the congestion. Integral sessions are those to which we can afford to insert delays in order to bound the congestion. Fractional sessions are those to which we cannot insert delays. However, congestion due to a fractional session i is only $\hat{r}_i$, which is small.

As before, $\mathcal{S}^{(q)}$ represents the schedule in the qth iteration. The set of integral sessions for $\mathcal{S}^{(q)}$ is denoted by $A^{(q)}$. For the initial schedule $\mathcal{S}^{(0)}$, all the sessions are fractional and we show that the relative congestion is less than 1. For schedule $\mathcal{S}^{(q)}$ we inductively assume that the relative congestion due to the current integral and fractional sessions is at most $c^{(q)}$ for any frame of size $I^{(q)}$ or larger. To create a schedule $\mathcal{S}^{(q+1)}$ from schedule $\mathcal{S}^{(q)}$ we carry out a frame-refinement step and a conversion step.

The frame-refinement step reduces the frame size

Schedule	Integral sessions	Frame size	Relative congestion
$\mathcal{S}^{(q)}$	$A^{(q)}$	$I^{(q)}$	$c^{(q)}$
Refinement	$A^{(q)}$	$\log^5 I^{(q)}$	$(1+o(1))c^{(q)}$
Conversion	$A^{(q)} \cup B^{(q+1)}$	$\log^5 I^{(q)}$	$(1+o(1))^2 c^{(q)}$
$\mathcal{S}^{(q+1)}$	$A^{(q+1)}$	$I^{(q+1)}$	$c^{(q+1)}$

Table 2: Refinement and conversion for dynamic routing.

from $I^{(q)}$ to $I^{(q+1)} = \log^5 I^{(q)}$, while slightly increasing the relative congestion from $c^{(q)}$ to $(1+o(1))c^{(q)}$. This step is achieved by delaying the integral packets by up to $\Theta\left((I^{(q)})^2\right)$ steps. We make sure that if session i is in $A^{(q)}$ then $1/r_i + d_i \geq (I^{(q)})^2$, and therefore the delays inserted can be tolerated. The conversion step converts some sessions from fractional to integral, while maintaining the frame size of $I^{(q+1)}$ and slightly increasing the relative congestion to $c^{(q+1)} = (1+o(1))^2 c^{(q)}$. These newly-converted sessions form a set $B^{(q+1)}$ and have associated values $1/r_i + d_i \geq (I^{(q+1)})^2$. This bound is chosen so that the sessions in $A^{(q+1)}$, which is $A^{(q)} \cup B^{(q+1)}$, will be able to tolerate the delays inserted during the next iteration of frame-refinement. During the conversion step we delay the packets in $B^{(q+1)}$ by up to $\Theta(1/r_i + d_i)$ steps. We are able to show the existence of "good" delays for both frame-refinement and conversion steps. Table 2 summarizes our approach.

At the termination of our algorithm we have a schedule $\mathcal{S}^{(\zeta)}$ in which every session is integral and the relative congestion is at most 1, for all frames of size larger than a certain constant. In $\mathcal{S}^{(\zeta)}$ all session-i arrivals during $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ are serviced during $[\mathcal{T}, 2\mathcal{T})$. Furthermore, all session-i packets reach their destination within $O(\mathcal{T}_i)$ steps of their injections.

Problem 3: Constant-factor stretching in the final schedule

As discussed above, we repeat the process of refinement and conversion until we have a schedule, $\mathcal{S}^{(\zeta)}$, in which all sessions are integral and in which the relative congestion is 1 for all frames of size larger than a certain constant w. In the static problem, a final schedule can easily be obtained by stretching $\mathcal{S}^{(\zeta)}$ by a constant factor. However, we cannot afford to have a constant blowup in our final schedule for the dynamic problem. This is because we need to independently schedule all time intervals $[0, \mathcal{T})$, $[\mathcal{T}, 2\mathcal{T})$, etc, and a constant blowup would make these time intervals overlap.

To overcome this problem, we first devise a schedule for a new network $\mathcal{M}$ that is constructed from the original network $\mathcal{N}$ as follows. Each edge e of $\mathcal{N}$ is replaced by $2w$ consecutive edges $e_1, \ldots, e_{2w}$, where w is the constant introduced above. The rates and routes of the sessions are unaffected. In $\mathcal{M}$, session i has length $D_i = 2wd_i = O(d_i)$.

All the techniques described earlier are applied to the network $\mathcal{M}$. We carry out successive conversion and refinement steps for $\mathcal{M}$ and obtain a schedule $\mathcal{S}^{(\zeta)}$, where the relative congestion is 1 for any frame whose size is larger than w. We then "smooth" $\mathcal{S}^{(\zeta)}$ and convert it to a schedule for $\mathcal{N}$ where only one packet at a time traverses any edge.

The idea behind the smoothing process is as follows. In $\mathcal{S}^{(\zeta)}$, more than one packet may require some edge of $\mathcal{M}$ during a given time step, but at most w packets can require any given edge f in $\mathcal{M}$ within w time steps. This means we can shuffle each packet that requires edge f by at most w time steps, so that exactly one packet traverses f at any step. Unfortunately, this shuffling in time can lead to an illegal schedule for $\mathcal{M}$, in which a packet can be scheduled to traverse the edges on its path out of order (timewise). However, one can prove that if we consider the schedule with respect to the packets traversing edge e_{2w}, for all e, then this schedule *is* legal, i.e. the packets cross *these* edges in order. Hence, we schedule edge e in $\mathcal{N}$ in exactly the same way that the corresponding edge e_{2w} is scheduled in $\mathcal{M}$.

Figure 2 is a schematic picture of our overall approach.

The main theorem of this paper is stated below. A detailed proof is contained in the full version of the paper.

Theorem 10 *Consider an arbitrary network in which sessions are defined. Each session i is associated with an injection rate r_i and path length d_i. Packets are injected to the network along these sessions subject to the injection rates. If the total rate on each edge is at most $1-\varepsilon$ for a constant $\varepsilon \in (0,1)$, then there exists a*

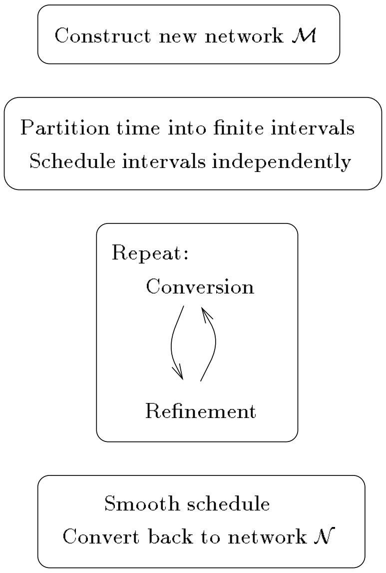

Figure 2: An overview of our approach for the dynamic routing problem.

template-based schedule such that each session-i packet reaches its destination within $O(1/r_i + d_i)$ *steps of its injection and at most one packet crosses an edge at each time step. This schedule also maintains constant edge queues.*

Acknowledgments

The authors wish to thank Bruce Maggs, Greg Plaxton and Salil Vadhan for many helpful comments.

References

[1] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 380 – 389, Burlington, VT, October 1996.

[2] J. Beck. An algorithmic approach to the Lovasz Local Lemma I. *Random Structures and Algorithms*, 2(4):343 – 365, 1991.

[3] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson. Adversarial queueing theory. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 376 – 385, Philadelphia, PA, May 1996.

[4] R. L. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, pages 114 – 31, 1991.

[5] R. L. Cruz. A calculus for network delay, Part II: Network analysis. *IEEE Transactions on Information Theory*, pages 132 – 141, 1991.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking: Research and Experience*, 1:3 – 26, 1990.

[7] A. Elwalid, D. Mitra, and R. H. Wentworth. A new approach for allocating buffers and bandwidth to heterogeneous, regulated traffic in an ATM node. *IEEE Journal on selected areas in communications*, pages 1115 – 1127, 1995.

[8] 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.

[9] F. T. Leighton, B. M. Maggs, and A. W. Richa. Fast algorithms for finding O(congestion + dilation) packet routing schedules. Technical report CMU-CS-96-152, Carnegie Mellon University, 1996.

[10] R. Ostrovsky and Y. Rabani. Local control packet switching algorithm. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, May 1997.

[11] 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 Transactions on Networking*, 1(3):344 – 357, 1993.

[12] 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 Transactions on Networking*, 2(2):137 – 150, 1994.

[13] Y. Rabani and E. Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, Philadelphia, PA, May 1996.

[14] J. S. Turner. New directions in communications, or which way to the information age. *IEEE Communications Magazine*, pages 8 – 15, 1986.

[15] L. Zhang. *An Analysis of Network Routing and Communication Latency*. PhD thesis, MIT, 1997.