Packet Forwarding Algorithms in a Line Network

Antonios Antoniadis[1]*, Neal Barcelo[1], Daniel Cole[1], Kyle Fox[2]**, Benjamin Moseley[3], Michael Nugent[1], and Kirk Pruhs[1]***

[1] University of Pittsburgh, Pittsburgh PA 15260, USA, antoniosantoniadis@gmail.com; ncb30, dcc20, mnugent, kirk@cs.pitt.edu
[2] University of Illinois at Urbana-Champaign, Urbana IL 61801, USA, kylefox2@illinois.edu
[3] Toyota Technological Institute at Chicago, Chicago IL 60637, USA, moseley@ttic.edu

Abstract. We initiate a competitive analysis of packet forwarding policies for maximum and average flow in a line network. We show that the policies Earliest Arrival and Furthest-To-Go are scalable, but not constant competitive, for maximum flow. We show that there is no constant competitive algorithm for average flow.

1 Introduction

The Internet Protocol (IP) layer of the TCP/IP suite is responsible for transporting (essentially fixed-sized) datagrams/packets from a source host, through intermediate routers, to a destination host specified by an IP address. The utilization of any imaginable economically-sustainable network will be sufficiently high so that routers will usually have a backlog of packets waiting to be forwarded. Thus routers need a policy that specifies which packets to forward first in the event of a backlog. Ideally the goal of this forwarding policy should be to provide the best possible quality of service (QoS) to the application layer clients, although this is a problematic goal as one consequence of the layering/encapsulation principle of the protocol suite design is that the overlying applications are generally hidden from the IP layer. Thus, a reasonable fallback goal for this forwarding policy would be to provide good QoS to the packets.

The most natural QoS measure for an individual packet j is the response/flow time $C_j - r_j$, the duration of time between the time r_j when the packet j is injected into the IP layer at the source host, until the time C_j when the packet j is ejected from the IP layer at the destination host. The most natural QoS measure

* With support by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).
** Research by this author is supported in part by the Department of Energy Office of Science Graduate Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract no. DE-AC05-06OR23100.
*** Supported in part by NSF grants CCF-1115575, CNS-1253218, and an IBM Faculty Award.

for a collection of packets is then to take a p-norm, for $p \in [1, \infty]$, of the flow time of individual packets. The ∞-norm, or maximum flow time, is usually the most mathematically tractable norm, and is the second most commonly considered norm in the systems literature. The 1-norm, or average flow time, is usually the second most mathematically tractable norm, and is the most commonly considered norm in the systems literature.

The goal of the research we report on in this paper is to initiate a competitive analysis of packet forwarding policies for these natural QoS measures. In this paper, we generally assume that the network topology is a line. Even for this simplest of topologies, we find the subtlety of the algorithm analysis and design process to be surprising.

There are three natural packet forwarding policies that play a central role in our findings:

Furthest-To-Go (FTG): The FTG policy always forwards a packet with the most hops left to go. It is not too difficult to see that FTG minimizes the makespan, the time that the last packet is delivered. We actually show in Section 5 the stronger statement that for every router i and for every time t, FTG has forwarded the maximum number of packets possible over router i by time t. Intuitively, this implies that FTG maximizes the amount of parallel processing possible in the network.

Earliest Arrival (EA): The EA policy always forwards the packet that was first injected into the network layer. For a one-edge network, it is well-known and obvious that the policy EA is optimal for maximum flow. For general line networks, it is obvious that EA is not optimal for maximum flow because there are situations where the optimal algorithm needs to forward a younger further-to-go packet. Earliest Arrival is also known in the literature as Longest-In-System.

Shortest-To-Go (STG): Shortest-To-Go always forwards the packet that is the fewest number of hops from its destination router. We show in Lemma 11 that STG achieves optimal average flow time if all packets are injected into the system at the same time. We recently learned that the same result was proven independently by Kowalski et al. and will appear in [1]. Shortest-To-Go is also known in the literature as Nearest-To-Go.

In this paper, we report on the progress that we made beyond these initial observations. Namely, we show that:

Maximum Flow on a Line: Our initial conjecture was that EA is $O(1)$-competitive.
 - In Section 3 we show that in fact EA is not $O(1)$-competitive. The lower bound instance results from a rather intricate recursive construction that increases the age of a packet by a fixed amount on each recursion. Intuitively, this shows that a competitive algorithm must take into account the path lengths of the packets.
 - In Section 4 we show that EA is however scalable, that is, $O(1)$-competitive with arbitrarily small speed augmentation. Intuitively, this shows that

EA should be reasonable until the network utilization is near the capacity of the network.
  - In Section 5 we show that FTG is also scalable.

Average Flow on a Line: Initially we had two competing intuitions as to the "right" policy for average flow time. One might reasonably think that as Shortest Remaining Processing Time is the optimal scheduling policy for average flow for arbitrary sized jobs on a single processor, that analogously the policy STG, which forwards a packets with the fewest hops to go, should be a good policy for average flow. Alternatively, one might reasonably think that FTG should be a good policy for average flow because it maximizes parallelism.
  - In Section 6 we show that there is no $O(1)$-competitive online algorithm. Intuitively in our lower bound construction, if the online algorithm initially forwards packets using STG, then there is a future in which FTG was the right initial policy, and if instead the online algorithm initially forwards packets using FTG, there is another future in which STG was the right initial policy (and there is no intermediate policy that is good in both futures). So in some sense, this shows that there is no possible resolution to the conflicting intuitions favoring STG and FTG.

Maximum Flow on a Tree: In Section 7 we show that there is no $O(1)$-speed $O(1)$-competitive deterministic local online algorithm. This shows that generalizing the problem to the second most simple network already makes the problem harder.

We then conclude by stating the two open problems "discovered" by this research that we find appealing.

1.1 Related Work

Previous work on routing algorithms under the adversarial model has, to the best of our knowledge, revolved around two distinct models. In the first one, stability is studied, i.e., whether the number of packets in the system will remain bounded as the system runs for an arbitrarily long period of time. In general this depends on the protocol studied, on the size and topology of the network and on the maximum rate at which the adversary is allowed to inject packets into the network. We refer to [2–13] for some representative papers under this model. In the second model, a subset of the packets has to be dropped, due to some restriction. For example, there might be a limit in the size of the buffer, a maximum delay (per packet) allowed by the system, or packets may come with a deadline. The objective is to maximize a function of the transmitted packets, for instance their number, size, or (weighted) value. Work in this model has mostly employed competitive analysis, see [14–25]. The problem has also been studied when the routers have shared but limited memory [26]. Our work significantly differs from both these models, since (1) instead of considering the stability of a network, we use competitive analysis with respect to specific objective functions, and (2) in our model every single packet has to be transmitted to its destination.

We would like to point out that [22] also studies the Furthest-To-Go algorithm on line networks, but with fixed-size buffers under the objective of maximizing the throughput. They show that for this model on a line of length k every greedy algorithm (including Furthest-To-Go) has a competitive ratio of $O(k)$, and also give a matching lower bound of $\Omega(k)$ on the competitive factor of Furthest-To-Go. Also, Angelov et al. [19] as well as Azar and Zachut [18], give centralized online algorithms with a polylogarithmic competitive ratio for the problem of maximizing throughput on the line. The special case of information gathering, where all packets have a common destination, was studied on the line by Rosén and Scalosub [25].

2 Preliminaries and Notation

We begin by introducing a formal model for the problems we consider and some notation. In the problems we consider there are k routers labeled 1 through k on a line network. The routers are ordered in increasing order on the line from left to right. Over time n packets arrive. We say that a packet j arrives at time r_j. We assume arrival times are integral. Each packet is associated with a path P_j. The path P_j consists of a set of routers that packet j must be processed on to be completed. This corresponds to sending a packet from its source to its destination. Since we are considering a line network, P_j will consist of a set of adjacent routers on the line. A router can process one packet at each time unit, which corresponds to sending this packet to the router to the right of this router on the line. A packet can only be processed if it is on a router and a packet j starts at the leftmost router in P_j. Note that a packet must be processed by every router in P_j and therefore $|P_j|$ is a lower bound on the amount of time a packet requires to be sent to its destination.

We will consider two different objective functions, namely total (average) flow time and maximum flow time. For some schedule, let C_j denote the time packet j is finished being processed. The flow time of j is $C_j - r_j$. For total flow time we are interested in minimizing $\sum_{j\in[n]} (C_j - r_j)$ and for maximum flow time we are interested in minimizing $\max_{j\in[n]}\{C_j - r_j\}$. We will be considering algorithms that possibly use resource augmentation. If an algorithm is given $s + 1/c$ speed the algorithm is allowed to send s packets every time step at a particular router and an additional packet every c time steps. Here s and c will be assumed to be integral. Note that we assume packets are sent only at discrete time steps. Therefore, a packet can only move one router in a time step.

We will compare our algorithms against a fixed optimal solution for a given objective and problem instance. We denote the optimal solution as OPT. For an algorithm A, we let $Q^A(t)$ be the packets alive at time t and $Q_i^A(t)$ as the packets available for processing on router i at time t. We let $p_i^A(t)$ denote the number of packets processed on router i by time t for A and, likewise, $p_i^O(t)$ for OPT. For a packet j and a fixed algorithm, which will be clear by the context, $P_j(t)$ is the remaining routers j needs to use to be completed at time t and $d_j(i,t)$ is the distance of packet j to router i. Note that $P_j(r_j) = P_j$. The value of $n_i^A(t', t)$ denotes the number of packets released by time t' that still need to use router i at time t for A and $n_i^A(t')$is short for $n_i^A(t', t')$. Note that the packets that contribute to $n_i^A(t', t)$ do not necessarily have to be in $Q_i^A(t)$. Let A_j be the total flow time for packet j for A and OPT_j be total flow time for packet j in the optimal schedule.

For an input I, let $A(I)$ and $\mathrm{OPT}(I)$ denote the final objective value for running I on A and OPT respectively. We may use A and OPT to denote the objective when I is clear from context. Finally, we say an algorithm A is s-speed c-competitive if $\frac{A(I)}{\mathrm{OPT}(I)} \leq c$ for any input I when A runs at s speed and OPT runs at unit speed.

Some proofs are ommitted due to space constraints.

3 Lower Bound for Earliest Arrival for Maximum Flow

We show that the EA policy is not constant competitive for maximum flow.

Theorem 1. *There exists an n_0 so that for each integer $L > n_0$, there exists an instance I with $OPT(I) = \Theta(L)$ and $\frac{EA(I)}{OPT(I)} \geq OPT(I)$. Furthermore, there exist instances $I_{n,k}$ with n packets and k routers so that $\frac{EA(I_{n,k})}{OPT(I_{n,k})} \geq n^{1/3}$, and $\frac{EA(I_{n,k})}{OPT(I_{n,k})} \geq k^{1/2}$.*

Let K and C be sufficiently large even integers such that $C < K/2 - 3$. Set an input with $C \cdot \frac{K}{2} - C + 3$ routers. We designate two sets of routers, the *stream-routers* and the *gap-routers*. The routers are defined as follows:

- There is a stream-router with index $R^S(0, 0) = 1$. At time $T^S(0, 0) = 0$ there are $K/2$ *short stream-packets* released to $R^S(0, 0)$ with destination 1 and $K/2$ *long stream-packets* released to $R^S(0, 0)$ with destination $K/2$.
- For each p and q with $1 \leq p \leq C$ and $0 \leq q \leq p - 1$ there is a stream-router with index $R^S(p, q) = p \cdot \frac{K}{2} - 2p + 2 + q$. At time $T^S(p, q) = p(K - 1) + \frac{K}{2}\left(\frac{p(p+1)}{2} - 1\right) - (p - 1)\left(\frac{K}{2} + 1\right) + q\left(\frac{K}{2} + 1\right)$ there are $K/2$ short stream-packets released to $R^S(p, q)$ with destination $p \cdot \frac{K}{2} - 2p + 2 + q$ and $K/2$ long stream-packets released to $R^S(p, q)$ with destination $(p + 1) \cdot \frac{K}{2} - p + 1$.
- For each p with $1 \leq p \leq C$ there is a gap-router with index $R^G(p, 0) = p \cdot \frac{K}{2} - p + 2$. At time $T^G(p, 0) = p(K - 1) + \frac{K}{2}\left(\frac{p(p+1)}{2}\right) + 1$ there are $K/2$ short gap-packets released to $R^G(p, 0)$ with destination $p \cdot \frac{K}{2} - p + 2$ and $K/2$ long gap-packets released to $R^G(p, 0)$ with destination $(p + 1) \cdot \frac{K}{2} - p + 1$.

Note that the instance created above consists of $k = \Theta(K^2)$ routers assuming $C = \Omega(K)$. Further, $n = \Omega(K^3)$. We have the following observation.

Note 2. The set of stream-routers and the set of gap-routers are disjoint. Further, for any two stream- or gap-routers i_1, i_2 with $i_1 < i_2$, the packets on router i_1 are released earlier than the packets for i_2.

We now compare the performance of the optimal schedule to EA for minimizing maximum flow time.

Lemma 3. *The maximum flow time for the optimal schedule is at most $K+C$ for the given instance.*

Lemma 4. *EA has total flow time of at least $(C+1)\cdot\frac{K}{2}$ on the given instance.*

We can finally prove Theorem 1.

Proof. By setting $C=\frac{K}{2}-4$, we have that on the above instance, $OPT=\Theta(K), EA=\Theta(K^2)$, and again, the number of routers is $k=\Theta(K^2)$ and the number of packets $n=\Omega(k^3)$. This proves the theorem. $\square$

4 Analysis of EA for Maximum Flow

We show that EA is scalable for maximum flow.

Theorem 5. *EA is $(1+\varepsilon)$-speed $4/\varepsilon$-competitive for any $\varepsilon>0$.*

We first prove a useful fact for the algorithm Earliest Arrival First. This fact is useful because it will give us an upper bound on how long it takes at time t for a packet j to be completed assuming no more packets arrive. We know that $|P_j(t)|$ is the remaining path length for packet j and we know packet j will need to wait at least this long to be completed. Further, $n_i^A(r_j,t)$ is the total number of packets with strictly higher priority than j that need to use router i. Thus, j may have to wait on all these packets and therefore j may need to wait $\max_{i\in P_j(t)}\{n_i^A(r_j,t)\}$ time. Intuitively, we would like to show that in fact j waits at most $|P_j(t)|+\max_{i\in P_j(t)}\{n_i^A(r_j,t)\}$ time to be completed assuming no more packets arrive. To do this, we would like to show that $|P_j(t)|+\max_{i\in P_j(t)}\{n_i^A(r_j,t)\}$ decreases each time step. Knowing that if this expression reaches 0 then j has reached its destination ($|P_j(t)|=0$), this would show that j waits at most this much time. We will not be able to show this directly, but rather will show a slightly more involved expression decreases in a similar manner. Fix an input I.

Lemma 6. *Let A be any algorithm (possibly with speedup). Let j be any packet alive at time t and suppose A processes the $\min\{s,Q_i^A(t)\}$ packets with earliest release time on each router i at time t. Then for any constant $c\geq s$*

$$\begin{aligned}\max_{i\in P_j(t+1)}&\{n_i^A(r_j,t+1)-c\cdot d_j(i,t+1)\}+c|P_j(t+1)|\\ &\leq \max_{i\in P_j(t)}\{n_i^A(r_j,t)-c\cdot d_j(i,t)\}+c|P_j(t)|-s.\end{aligned}$$

We can now prove Theorem 5.

Proof. Assume the theorem is false for a contradiction. Let t be the earliest time such that there is some packet j with flow time greater than $\frac{4}{\varepsilon}\mathrm{OPT}$ so that $t - r_j > \frac{4}{\varepsilon}\mathrm{OPT}$. By Lemma 6, $\max_{i \in P_j(t)}\{n_i^A(r_j, t) - 2d_j(i, t)\} + 2|P_j(t)|$ decreases by 1 every time step except for every $1/\varepsilon$ time steps where it decreases by 2. Further, it does not reach 0 until j's completion. Therefore,

$$\begin{aligned}\max_{i \in P_j} n_i^A(r_j) + 2|P_j| &\geq \max_{i \in P_j(t)}\{n_i^A(r_j, r_j) - 2d_j(i, r_j)\} + 2|P_j(r_j)| \\ &> \frac{4}{\varepsilon}\mathrm{OPT} + 4\mathrm{OPT} - 1.\end{aligned}$$

Let i be a router in P_j that maximizes the value $n_i^A(r_j)$. As $|P_j| \leq \mathrm{OPT}$ and $\mathrm{OPT} \geq 1$, we have

$$n_i^A(r_j) > \left(\frac{4}{\varepsilon} + 1\right)\mathrm{OPT}. \tag{1}$$

Packet j is the first packet with flow greater than $\frac{4}{\varepsilon}\mathrm{OPT}$, so any packets contributing to $n_i^A(r_j)$ have age at most $\frac{4}{\varepsilon}\mathrm{OPT}$ at time r_j. These packets have arrival time between $r_j - \frac{4}{\varepsilon}\mathrm{OPT}$ and r_j. Further, the optimal schedule must complete them by time $r_j + \mathrm{OPT}$, so the total amount of time the optimal schedule can process them is $r_j + \mathrm{OPT} - (r_j - \frac{4}{\varepsilon}\mathrm{OPT}) = (\frac{4}{\varepsilon} + 1)\mathrm{OPT}$. The optimal schedule can only process one of these packets at a time on router i, so we observe $n_i^A(r_j) \leq (\frac{4}{\varepsilon} + 1)\mathrm{OPT}$. Finally, we combine the previous expression with (1) to yield

$$\left(\frac{4}{\varepsilon} + 1\right)\mathrm{OPT} > \left(\frac{4}{\varepsilon} + 1\right)\mathrm{OPT},$$

a contradiction based on our assumption that the theorem is false. □

5 Analysis of FTG for Maximum Flow

We show that FTG is scalable for maximum flow.

Theorem 7. *FTG is $(1+\varepsilon)$-speed $3/\varepsilon$-competitive for any $\varepsilon > 0$.*

We begin by proving a fact for the algorithm FTG that will prove useful later and is interesting in its own right. Fix an input I. The following lemma essentially states that this algorithm gets the maximum amount of 'parallelism' possible by showing that for this algorithm at any point in time each router will have processed the most number of packets possible.

Lemma 8. *Let A be any algorithm (possibly with speedup) for which each router i processes a packet with furthest final destination at least once every time step if any are available. We have $p_i^A(t) \geq p_i^O(t)$ at all routers i and times t.*

In order to prove Theorem 7, we need to formalize how efficiently routers are processing packets with extra speed. We say router i *fully processes* a set J of packets at time t if router i processes as many packets from J at time t as the speedup allows. We have the following lemma.

Lemma 9. *Run FTG with speed $1+\varepsilon$ on input I. Let j be a packet and let $i \in P_j$. Let t be any time step strictly before router i processes packet j. If $t \geq r_j + d_j(i, r_j) + 1$, then router i processes at least one packet at time t with destination at least as far as packet j's. Further, if $t \geq r_j + (d_j(i, r_j) + 1)/\varepsilon$, then router i fully processes packets with destination at least as far as j's at time t.*

We may now prove Theorem 7.

Proof. Assume the theorem is false for a contradiction. Let OPT be the maximum flow in the optimal schedule. Run FTG with speed $1 + \varepsilon$ and let t be the earliest time such that there is some packet j with flow time greater than $\frac{3}{\varepsilon}\mathrm{OPT}$. Let i be the router upon which j is queued (or being processed) at time t. Let t_0 be the earliest time such that FTG always processes at least one packet every step of the interval $[t_0, t)$. Let J be the set of packets processed by i during the interval $[t_0, t)$.

Each packet arrives at most distance $\mathrm{OPT} - 1$ from its destination. Therefore, every packet in J arrives no earlier than $t_0 - \mathrm{OPT}$. Otherwise, Lemma 9 implies router i processes a packet at time $t_0 - 1$. Further, every packet in J is released strictly before time t and completed by the optimal algorithm by time $t + \mathrm{OPT}$. Therefore, the optimal algorithm spends strictly less than $t - t_0 + 2\mathrm{OPT}$ time steps processing all packets in J on router i.

By assumption, $t > r_j + \frac{3}{\varepsilon}\mathrm{OPT} = r_j + \frac{\mathrm{OPT}}{\varepsilon} + \frac{2}{\varepsilon}\mathrm{OPT}$. Therefore, router i has fully processed packets over $\frac{2}{\varepsilon}\mathrm{OPT}$ consecutive time steps by Lemma 9. Over this time period, there are at least $2\mathrm{OPT}$ time steps where i processes 2 packets instead of 1. As i processes at least one packet every time step since t_0, we have $|J| \geq t - t_0 + 2\mathrm{OPT}$. The optimal algorithm spends strictly less than $|J|$ time steps to process every packet in J on router i, a contradiction. □

6 Average Flow on a Line

We now show that there is no constant competitive algorithm for average flow on a line.

Theorem 10. *Any randomized algorithm for packet routing on a line is $\Omega(k)$ competitive for average flow in the oblivious adversary model.*

Proof. We prove the theorem for any deterministic algorithm. The proof can be easily generalized to randomized algorithms. Let A be any deterministic algorithm for packet routing on a line. Consider the following input. For each $i \in \{1, \ldots, k/2\}$, we have k *short* packets arrive at time 0 with source i and destination i. In addition, k *long* packets arrive at time 0 with source 1 and destination k. The rest of the input is determined by A's behavior.

Suppose A processes fewer than $k/4$ packets on router $k/2$ at time k. Then there are at least $3k/4$ long packets that still need processing on router $k/2$. Assume they all wait at router $k/2$. At each time step from $2k - 1$ to some sufficiently large T, a packet arrives with source k and destination k. Algorithm A will still have $k/4$ long packets remaining at time $2k - 1$, so it will always have $k/4$

packets alive with destination k and total flow time $\Omega(kT)$. However, consider FTG which finishes all long packets by time $2k-1$ and has only one packet pending at each time step after $3k-1$. The optimal total flow time is $O(T)$, and the competitive ratio of A is $\Omega(k)$.

Now, suppose A processes at least $k/4$ packets on router $k/2$ at time k. Then each router $i \in \{1, \ldots, k/2\}$ has at least $k/4$ short packets remaining, for a total of $k^2/8$ short packets remaining. For each time step from k to some sufficiently large T and for each router $i \in \{1, \ldots, k\}$, a packet with source i and destination i arrives (so k packets in total arrive at each time step). Algorithm A has at least $k^2/8 + k$ packets alive at each time step, so it has total flow time $\Omega(k^2 T)$. However, consider the algorithm that always schedules a packet with nearest destination. This algorithm finishes all the short packets by time k, so it has $2k$ packets remaining at each time step after k. The optimal flow time is $O(kT)$, and the competitive ratio of A is $\Omega(k)$. $\square$

We show that STG achieves optimal average flow time if all packets are injected into the system at the same time.

Lemma 11. *STG is optimal for the objective of average flow if all packets are released at time* 0.

7 Maximum Flow on Trees

We briefly explore extending our ideas to work with networks of routers that do not necessarily lie on a directed line. It turns out the problem of minimizing maximum flow time becomes much more difficult in this setting, unless we allow routers some way to communicate with one another.

To make the idea of communication concrete, define a *local* algorithm to be one where each router i processes packets based only on the existence of the packets currently queued at router i, their arrival times, and their distance to their destination. Define a *router network* as a directed graph $G = (\{1, \ldots, m\}, E)$ with arbitrary edge set. An input for a router network G can only move a packet from router i_1 to router i_2 in one processing step if $i_1 i_2$ is an edge in G. A *tree network* is a router network where the undirected edges form a spanning tree. We have the following theorem.

Theorem 12. *No deterministic local algorithm for packet routing on a tree network is s-speed c-competitive for any constants s and c.*

Proof. Fix a deterministic local algorithm A and let k be a sufficiently large integer multiple of s. We define a tree network and associated input for every $L \in \{0, \ldots, k-1\}$ recursively as follows. The construction is based on identifying routers of one or more paths of length k, where the routers within a single path are indexed 1 through k. We maintain the invariant that algorithm A queues a packet on the $L+1$st router of some path P in network L at time kL/s when $L < k$, and A completes a packet at time kL/s when $L = k$. If $L < k$, then we refer to the router mentioned in the invariant as a *shared router*.

For $L = 0$, we use one path P of length k. At time 0, we have 1 packet arrive with a source of the first router in P and a destination of the kth router in P. The invariant trivially holds.

For $L > 0$, we create k copies of network $L - 1$ and its associated input. As A is deterministic and local, we know which path P maintains the invariant for $L-1$ in each of these copies. Identify the k shared routers that are guaranteed by the invariant. It takes k/s time steps for A to process the k packets on the now common shared router, so either the $L + 1$st router in some path receives a packet at time kL/s or A completes a packet at time kL/s.

We see immediately that the maximum flow time for A in network k is k^2/s. However, an optimal schedule will always give precedence to the one packet at each router that will later need processing on a shared router as described above. There are never more than k packets that need to use a single router, and all but one will will not go to another shared router, so that optimal maximum flow time is $2k - 1$ for a competitive ratio of $\Omega(\frac{k^2}{ks})$. Setting k sufficiently high proves the lemma. □

8 Conclusions

We initially found it surprising that there was no prior literature on the packet forwarding problems considered in this paper as they seem quite natural. Part of the explanation may be that even for a line network, the problems are surprisingly challenging. The two most natural open problems "discovered" by our research are:

- Is there an $O(1)$-competitive algorithm for maximum flow on a line? The authors are in disagreement amongst themselves about which answer is most likely, and there is a modest wager on the outcome. As evidence that there might be an $\omega(1)$ lower bound, even finding a reasonable candidate policy seems challenging. We are able to show that all of the following policies are not $O(1)$-competitive:
  - *c-EA/FTG* - Forward using EA every c steps, and FTG the rest of the time, where $c > 0$ is any constant.
  - *c-OPT/FTG Threshold* - If we know OPT, the value of the optimal solution, forward the furthest-to-go packet that has age at least $c \cdot \text{OPT}$, otherwise send the furthest-to-go packet, where $c > 0$ is any constant.
  - $\frac{1}{c}$*-Local [Global] FTG Threshold* - Forward the furthest-to-go packet with age at least $1/c$ of the maximum age of any packet at the current router [or in the network], where $c > 0$ is any constant.
- Is there an $O(1)$-speed $O(1)$-competitive (or even scalable) policy for average flow on a line? In this case, the authors all conjecture that the answer is yes, for the traditional reason that we were not able to prove otherwise. Here candidate policies are abundant, but it is not clear how to do the analysis. STG is a good algorithm when all release times are the same, which suggests that it is a good candidate for an $O(1)$-speed $O(1)$-competitive algorithm,

as processor-sharing is for one processor [27]. Also running STG and FTG simultaneously is another obvious candidate algorithm. The main issue with the analysis is that neither the standard potential function approach [28] nor the standard application of linear programming duality seem to work.

References

1. Kowalski, D., Nussbaum, E., Segal, M., Milyeykovsky, V.: Scheduling problems in transportation networks of line topology. Optimization Letters, to appear (2013)
2. Aiello, W., Kushilevitz, E., Ostrovsky, R., Rosén, A.: Adaptive packet routing for bursty adversarial traffic. J. Comput. Syst. Sci. **60**(3) (2000) 482–509
3. Andrews, M., Awerbuch, B., Fernández, A., Leighton, F.T., Liu, Z., Kleinberg, J.M.: Universal-stability results and performance bounds for greedy contention-resolution protocols. J. ACM **48**(1) (2001) 39–69
4. Andrews, M.: Instability of FIFO in the permanent sessions model at arbitrarily small network loads. ACM Transactions on Algorithms **5**(3) (2009)
5. Andrews, M., Zhang, L.: The effects of temporary sessions on network performance. SIAM J. Comput. **33**(3) (2004) 659–673
6. Borodin, A., Kleinberg, J.M., Raghavan, P., Sudan, M., Williamson, D.P.: Adversarial queuing theory. J. ACM **48**(1) (2001) 13–38
7. Broder, A.Z., Frieze, A.M., Upfal, E.: A general approach to dynamic packet routing with bounded buffers. J. ACM **48**(2) (2001) 324–349
8. Leighton, F.T., Maggs, B.M., Rao, S.: Packet routing and job-shop scheduling in O(Congestion + Dilation) steps. Combinatorica **14**(2) (1994) 167–186
9. Ostrovsky, R., Rabani, Y.: Universal $O(\text{Congestion} + \text{Dilation} + \log^{1+\text{epsilon}} N)$ local control packet switching algorithms. In: STOC. (1997) 644–653
10. Rabani, Y., Tardos, É.: Distributed packet switching in arbitrary networks. In: STOC. (1996) 366–375
11. Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Adversarial queuing on the multiple access channel. ACM Transactions on Algorithms **8**(1) (2012) 5
12. Gamarnik, D.: Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks. SIAM J. Comput. **32**(2) (2003) 371–385
13. Díaz, J., Koukopoulos, D., Nikoletseas, S.E., Serna, M.J., Spirakis, P.G., Thilikos, D.M.: Stability and non-stability of the FIFO protocol. In: SPAA. (2001) 48–52
14. Srinivasan, A., Teo, C.P.: A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. SIAM J. Comput. **30**(6) (2000) 2051–2068
15. Awerbuch, B., Azar, Y., Plotkin, S.A.: Throughput-competitive on-line routing. In: FOCS. (1993) 32–40
16. Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. Algorithmica **43**(1-2) (2005) 63–80
17. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies for QoS switches. In: SODA. (2003) 761–770
18. Azar, Y., Zachut, R.: Packet routing and information gathering in lines, rings and trees. In: ESA. (2005) 484–495
19. Angelov, S., Khanna, S., Kunal, K.: The network as a storage device: Dynamic routing with bounded buffers. Algorithmica **55**(1) (2009) 71–94
20. Adler, M., Rosenberg, A.L., Sitaraman, R.K., Unger, W.: Scheduling time-constrained communication in linear networks. Theory Comput. Syst. **35**(6) (2002) 599–623

21. Gordon, E., Rosén, A.: Competitive weighted throughput analysis of greedy protocols on DAGs. ACM Transactions on Algorithms **6**(3) (2010)
22. Aiello, W., Ostrovsky, R., Kushilevitz, E., Rosén, A.: Dynamic routing on networks with fixed-size buffers. In: SODA. (2003) 771–780
23. Chin, F.Y.L., Chrobak, M., Fung, S.P.Y., Jawor, W., Sgall, J., Tichý, T.: Online competitive algorithms for maximizing weighted throughput of unit jobs. J. Discrete Algorithms **4**(2) (2006) 255–276
24. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM J. Comput. **33**(3) (2004) 563–583
25. Rosén, A., Scalosub, G.: Rate vs. buffer size-greedy information gathering on the line. ACM Transactions on Algorithms **7**(3) (2011) 32
26. Kesselman, A., Mansour, Y.: Harmonic buffer management policy for shared memory switches. Theor. Comput. Sci. **324**(2-3) (2004) 161–182
27. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. ACM Transactions on Algorithms **8**(3) (2012) 28
28. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News **42**(2) (2011) 83–97