Online Time-Constrained Scheduling in Linear Networks

Joseph (Seffi) Naor† Adi Rosén Gabriel Scalosub
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
Email: {naor,adiro,gabriels}@cs.technion.ac.il

***Abstract*— We consider the problem of scheduling a sequence of packets over a linear network, where every packet has a source and a target, as well as a release time and a deadline by which it must arrive at its target. The model we consider is bufferless, where packets are not allowed to be buffered in nodes along their paths other than at their source. This model applies to optical networks where opto-electronic conversion is costly, and packets mostly travel through bufferless hops. The offline version of this problem was previously studied in [1]. In this paper we study the online version of the problem, where we are required to schedule the packets without knowledge of future packet arrivals. We use competitive analysis to evaluate the performance of our algorithms. We present the first deterministic online algorithms for several versions of the problem. For the problem of *throughput maximization*, where all packets have uniform weights, we give an algorithm with a logarithmic competitive ratio, and present some lower bounds. For other weight functions, we show algorithms that achieve optimal competitive ratios. We complete our study with several experimental results.**

I. Introduction

As technology advances, communication networks are constantly going through rapid change. The classic best-effort mechanisms are given up in favor of networks that are able to provide Quality-of-Service (QoS) guarantees. The growing use of multimedia applications motivate this transition. Such applications involve continuous transmission of data, which requires some guarantees as to its arrival time, bandwidth allocation etc. [2].

It is often the case that the overall number of packets destined to be transmitted through the network exceeds the network's capacity. In such cases, packets are either delayed or dropped. When considering streaming video or audio data, there is very little point in delaying such packets more than some predetermined period of time. Take, for example, a home user listening to the radio over the Internet. We can model such a transmission by considering every packet to have a certain deadline by which it must arrive at its destination. In such a setting, having the packet arrive after its deadline is of no use.

Real life applications vary in importance and value as well, thus rendering some packets more important than others. Consider, for example, the case of MPEG encoding, where some packets are more important than others when reconstructing the image at the target. This situation makes it vital to decide which packets to schedule at any given time, such that the decision will eventually result in a "best" possible set of packets, which are all delivered by their deadline.

When considering such packets with their corresponding deadlines, one would want to take into account both the packet's importance as well as its deadline when trying to determine which packet to route first. Additionally, packets can have different values, according to the end user's willingness to pay for an improved quality of service. In such a scenario, delivering valuable packets on time would mean more profit for the service provider, which should naturally be maximized. Time-constrained traffic is also the common case in real-time applications, such as avionics, industrial process control, and automated manufacturing, which necessitate coping with time constrained communication in interconnection networks [3].

In this paper we consider the problem of online scheduling a sequence of packets, each with a deadline constraint. The model underlying our work is a *bufferless* scheduling environment. In this model, a packet can only be stored at its source, and cannot be buffered in any node along its path. Once a packet has left its source, it must move along its designated path without interruptions or delays, until arriving at its destination. Any interruption or delay causes the packet to be dropped. This model is the common setting in optical networks, where trying to buffer packets in nodes along the path requires opto-electronic conversion of the signal, a prohibitively costly operation. This is the case in Wavelength Division Multiplexing (WDM) networks, where a packet is assigned a wavelength along which it is supposed to be transmitted throughout its path.

We restrict our attention to specific network topologies such as the line and the ring. The results of [4] motivate this focus, since under common complexity assumptions, for arbitrary graphs, no reasonable approximation can be obtained in polynomial time. Moreover, focusing on simple network topology like the line topology or the ring topology is motivated by considering electro-optical interconnection networks. In such networks, we might have a packet's path go through several long bufferless hops with very few nexus points, each enabling the expensive optical-electric conversion. This occurs for ex-

†This research is supported in part by a foundational and strategical research grant from the Israeli Ministry of Science, and by a US-Israel BSF Grant 2002276.

ample in a mesh network topology, employing a dimension-order routing policy. In such a case we can use a bufferless strategy along rows and columns, and perform a conversion to change dimensions (see [3]). Another advantage in considering simple topologies is the fact that they usually adhere to simple routing-path selection. In cases of less regularly structured networks, it is often the case that packets are routed along subnetworks of such simple topologies.

A. Our Results

We present the first online algorithm for bufferless scheduling of packets with deadline constraints in a linear network topology. Our goal is to maximize the total weight of packets delivered by their deadlines. A packet p contributes its weight to the overall weight gained by the algorithm only if it arrives at its target node by its deadline. We can further show that these results extend to a ring network topology, using arguments similar to those appearing in [1].

We present results for several special cases of the problem, determined by the weights given to the packets. In the *Throughput Maximization* problem the packets have uniform weights, i.e., for every packet p, its weight is equal to some constant w, where without loss of generality $w = 1$, and thus our goal is to maximize the number of packets scheduled successfully. In the *Maximum Network Utilization* problem the weight of each packet is defined to be its path length. The optimization problem in this case can be considered as trying to maximize the utilization of the network over time, where only packets scheduled successfully contribute to the network utilization. We further present results for the general case of arbitrary weights.

We analyze the performance of our online algorithm using *competitive analysis* (see [5], [6]), which compares the schedule produced by the algorithm to the optimal schedule produced by an algorithm with full knowledge of future incoming packets. This approach is robust in the sense that it makes no assumptions on the arrival sequence of packets. We assume that the algorithm has no knowledge about any packet until the packet is released at its source, at which point the algorithm learns its source, target, and deadline. A deterministic online algorithm for a maximization problem is said to be *c-competitive* if the ratio between its performance and the performance of an optimal schedule is at least $1/c$, for every possible request sequence.

In Section II we present an $O(\min\{\log\alpha, R\})$-competitive algorithm for the throughput maximization problem, where α is the ratio between the length of the longest path of a packet in the input sequence and the length of the shortest path, and R is the number of different path lengths appearing in the sequence. This reduces to an $O(\log n)$-competitive algorithm in the worst setting. Unlike the results of [7] and [8] for task scheduling on a single machine, our algorithm need not know the value of the parameter α beforehand. We give an example exhibiting our analysis to be tight up to a constant factor. We additionally show that no deterministic algorithm for the problem can achieve a competitive ratio better than 2.

In Section III we give a constant competitive algorithm for the problem of maximizing network utilization. This algorithm is an adaptation to our model of an algorithm given in [9]. We further derive an $O(\beta)$-competitive algorithm for arbitrary weights where β is the maximum ratio between any two packets' weight-to-length ratio. Due to the results of [10], this is the best possible, up to a constant factor.

In Section IV we show how our results can be applied to a ring network topology.

In Section V we present several experimental results, where we compare the performance of our suggested algorithm for the problem of maximum throughput with an offline 2-approximation algorithm, as well as with an online greedy algorithm. We test the performance of the algorithms on randomly generated input sequences, and show that the performance of our algorithm is very close to the performance of the offline 2-approximate schedule. Our results further show that our algorithm outperforms the greedy algorithm.

B. Previous Work

The offline version of our problem in the linear network topology was first considered by Adler *et al.* in [1]. They restricted their attention to the problem of throughput maximization and showed that it is NP-hard, and further provided a 2-approximation algorithm for the problem. Another model considered in [1] is the *buffered* model, where packets are allowed to be stored in a buffer of any node along their path. Adler *et al.* showed that allowing the packets to be buffered along their paths can increase the throughput by at most an $O(\log\gamma)$ factor, relative to the throughput obtained by a bufferless schedule, where γ is the minimum among the network size, the number of packets in the instance and the maximum slack a packet has.[1] Adler *et al.* devised a distributed online algorithm for the buffered case, which mimics the approximation algorithm given for the buffereless case. An extension of these results was later given by Adler *et al.* in [4], where they present algorithms for several versions of the time constrained scheduling problem, all in an offline setting. They first describe a 2-approximation algorithm for the bufferless case in a linear network, where packets are allowed to have arbitrary weights. They further consider the case where the underlying network topology is a tree or a mesh in the bufferless setting. For this problem they present constant-approximation algorithms for both the throughput maximization problem as well as for arbitrary weights. For the buffered case under the tree and mesh topologies, they devise an algorithm based on the algorithm for the bufferless case, with a logarithmic approximation guarantee.

The hardness results appearing in [4] motivate the focus on particular network topologies as they show that for any $\varepsilon > 0$, there is no $k^{1-\varepsilon}$-approximation algorithm for the problem in general networks, unless NP=ZPP, where k is the number of

[1] For the definition of slack, see Section I-C.

packets in the instance. This hardness result is based on the hardness of MAX-INDEPENDENT-SET, and it holds even if the underlying topology is either a directed acyclic graph or a planar graph.

The only result regarding the online version of the problem is given in [4], where they show that no deterministic online algorithm can achieve a competitive ratio better than $\Omega(\log n)$ when the underlying graph is a tree, in both the bufferless and the buffered settings, where n denotes the size of the network. One can compare this result with our upper bound for the linear network topology, which is guaranteed to be $O(\log n)$-competitive.

Our problem is closely related to interval scheduling problems and other call control models, e.g., [9], [11], and [12]. In the online interval scheduling problem we are given a sequence of intervals to schedule on a line segment. In some cases the problem can be solved in polynomial time, e.g., the case where the intervals are given in non-decreasing order of their left end-point, all having uniform weights, and preemption is allowed. In other cases however there are lower bounds on the attainable competitive ratio of any online algorithm, e.g., the case where the weight of an interval is defined to be its length, even in a randomized setting [11], and the case where intervals have uniform weights in a deterministic setting [9]. These lower bounds apply to non-preemptive scheduling of the intervals. Our model however is not reducible in the general case to either of these. The main difference between our model and the ones mentioned above is the concept of time, which introduces further constraints on the scheduling problem. Further results related to our problem involve multiple bin-packing, dealt with in the context of call admission control and wavelength division multiplexing in optical networks [13], which were later adapted to the case were calls are allowed to be preempted [12].

Some results regarding online task scheduling on a single machine, where each job must terminate by a certain deadline, are also related to our problem. Baruah *et al.* show in [10] that when packets may have arbitrary weights, no deterministic online algorithm can achieve a competitive ratio better than $\Omega(\beta)$, where β is the ratio between the largest and the smallest weight-to-length ratio of the packets in the instance. In [7] Koren and Shasha present an online algorithm for the problem, whose guarantee is exactly that of the lower bound in [10]. Their algorithm need know the value of β in advance. A guarantee based on a different parameter is given by Garay *et al.* in [8] for the problem of throughput maximization. They present an algorithm that is guaranteed to be $O(1/\kappa)$-competitive, where κ is the minimum ratio between the slack and the processing time of all jobs in the request sequence. In this case as well, the algorithm has to be given the value of κ in advance.

C. The Network Model

Our main results will be described for the linear network. We model our problem by a digraph $G = (V, E)$, where $V = \{1, \ldots, n\}$, and $E = \{(i, i+1) | 1 \leq i \leq n-1\}$. An instance comprises additionally of a set of packets that are to be routed through the network. Each packet p is specified by a tuple $(s_p, t_p, r_p, d_p, w_p)$, where s_p and t_p denote the source and target nodes respectively, r_p is the packet's release time, i.e., the time at which the packet is available for routing, d_p denotes the packet's deadline, and w_p is the packet's weight. We denote by $|p| = t_p - s_p$ the *length* of packet p. The algorithm learns of packet p in time r_p. The above definitions make it natural to consider the concept of *slack* each packet has, also known as *laxity*, defined by $\ell(p) = d_p - r_p - |p|$. The slack of packet p captures the notion of the maximum amount of time a packet can wait at its source node if it is to arrive at its target node by its deadline. We denote by $\ell_t(p) = d_p - t - |p|$ the *residual slack* of packet p in time t. A packet can be scheduled to leave its source at any time t for which $\ell_t(p) \geq 0$. We consider a *synchronous* model, where at each time step at most one packet can be transmitted on any edge, and we focus our attention on the *bufferless* case. We make no restriction on the amount of storage available at any node. We further assume packets can be *preempted* but cannot be *rescheduled*. Preemption means that a packet on route to its destination can be stopped, in which case it is dropped and cannot be rescheduled, even if its residual slack allows it. Every packet arriving at its destination by its deadline contributes its weight to the overall weight obtained, and is considered successfully scheduled. Every other packet contributes 0 to the overall obtained weight. The goal is to maximize the weight obtained.

D. Terminology

We follow the geometric representation introduced in [1]. We define the concept of *waves* upon which we "mount" the packets to be scheduled. Consider a two dimensional array whose X-axis represents the linear network, numbered $1, \ldots, n$ to designate the network nodes, and its Y-axis represents time, numbered $1, 2, \ldots$ to designate discrete time steps. Given a packet p that was presented at time r_p with slack $\ell(p)$, in order for it to arrive at its destination by its deadline, it must be sent from its source at some time $t \in \{r_p, \ldots, r_p + \ell(p)\}$. Every such scheduling of p starting at t can be geometrically viewed as packing an interval of length $|p|$ on a SW-NE line starting at point (s_p, t) and ending in $(t_p, t + |p|)$. We call each such SW-NE line a *wave*. Every such wave represents the network resources used over time. Each packet has a set of *eligible* waves, defined according to the packet's parameters, where a packet can be mounted on any of its eligible waves. Figure 1 shows an example of the waves eligible for a packet p for which $\ell(p) = 4$, and the location in which it can be mounted in every one of them. For each packet p, we consider the waves eligible for packet p as ordered from earliest (crossing point (s_p, r_p)) to latest (crossing point $(s_p, r_p + \ell(p))$). A feasible schedule solution is a packing of the packets upon the waves, such that on any wave no two packets intersect, and every packet is scheduled on at most one wave.

Consider for example an instance where all packets have zero slack. In this case, every packet has only one eligible wave. We therefore seek to compute a maximum-independent set, in an online fashion, for each wave independently. Since preemption is allowed, for such instances this can be done optimally (in the case of uniform weights). To see this notice that when focusing on a single wave, the packets corresponding to this wave are given in increasing order of their left end-point. This is due to the fact that packet p is introduced in time r_p. We can therefore preempt a currently scheduled packet q on the wave in favor of a packet p for which $t_p < t_q$. This mimics exactly the behavior of an offline algorithm for finding a maximum independent set in an interval graph in these settings, which finds an optimal solution. If we allow packets to have positive slack, the plot thickens, as demonstrated in Section II-A.

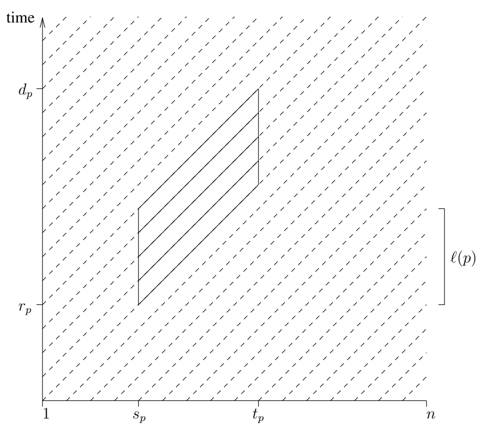

Fig. 1. Geometric representation of waves

In what follows we will use the following notation. Let $M = \max_p |p|$ and let $m = \min_p |p|$. We let α denote the ratio M/m and R is the number of different packet lengths appearing in the input. Define the *density* of packet p to be $\rho(p) = w_p/|p|$. Denote by $\rho_{\min} = \min_p \rho(p)$, $\rho_{\max} = \max_p \rho(p)$ and let $\beta = \rho_{\max}/\rho_{\min}$.

II. Throughput Maximization

We first consider the case where for every packet p, $w_p = w$ for some constant w. Without loss of generality we assume $w = 1$, and thus our goal is to maximize the number of packets scheduled.

A. Online Bufferless Lower Bound

Theorem 1: No deterministic online algorithm can achieve a competitive-ratio better than 2. This holds even if rescheduling is allowed.

Proof: Consider a linear network with 4 nodes $\{v_1, \dots, v_4\}$. We now describe an adversary. The adversary releases at time 0 a packet p with slack $= 1$, going from v_1 to v_4. If the algorithm schedules it on its first wave (i.e., it starts moving at $t = 0$), then the adversary releases at time $t = 2$ a packet q with zero slack, going from v_3 to v_4. If, on the other hand, the algorithm schedules p on its second wave (i.e., it starts moving at $t = 1$), then the adversary releases at time $t = 2$ a packet q with zero slack, going from v_2 to v_3. In either case, the algorithm can deliver at most one of the two packets, while an optimal solution delivers both. Notice that in both cases, if the algorithm preempts p in favor of q, then it cannot reschedule p on any other wave, because at the time of preemption p has a negative residual slack, i.e., it can no longer reach its target node by its deadline. We can repeat this procedure an arbitrary number of times, thus ensuring no deterministic online algorithm can achieve a competitive ratio better than 2. ■

B. Online Bufferless Upper Bound

1) A Simple Randomized Strategy: A simple greedy strategy can be used to devise a randomized $O(\log n)$-competitive non-preemptive algorithm for the problem. Consider a new packet just arrived. If it can be scheduled (considering the previously scheduled packets) on any wave, then schedule it. Otherwise, discard it. Since this algorithm is $(\alpha + 1)$-competitive when considered on any single wave, using the multiple-bin packing methodology appearing in [13], it follows that the above algorithm is $(\alpha + 2)$-competitive for our problem. We now introduce randomization: Consider a partition of the packets into $O(\log n)$ classes according to their length, where class i consists of all packets whose length falls in the interval $(2^i, 2^{i+1}]$, and we have $i = 0, \dots, \log n - 1$. Pick uniformly at random a class i, and use the greedy strategy described above to schedule only packets from class i. Denote by α_i the ratio between the maximum length to the minimum length of packets in class i. Since for every i we have $\alpha_i = 2$, using linearity of expectation, we conclude that the above randomized non-preemptive algorithm is $O(\log n)$-competitive.

2) The Deterministic Case: The non-preemptive simple strategy applied above will not do in the deterministic setting. To see this, consider an input sequence consisting of all zero slack packets. One packet which needs to traverse the entire network, followed by a sequence of $(n - 2)$ unit-path-length non-intersecting packets, each intersecting the path of the first packet on a different link. It follows that any non-preemptive deterministic algorithm can be $O(n)$-competitive at best. We apply a different method for the deterministic case to balance between "long" and "short" packets. We analyze in the Theorem 2 the competitive ratio guarantee of our algorithm, which we call MT (See Algorithm 1 below).

We say that packet p *evicts* packet q if the condition in line 7 holds and q is replaced by p. Let us first make sure that the algorithm is well defined, and indeed produces a feasible schedule.

Lemma 1: For any sequence of h packets, MT produces a feasible schedule.

Proof: Proof by induction on h. For $h = 0$, the claim clearly holds. Assume the claim is true for any sequence of

Algorithm 1 Algorithm MT

Given a new packet p just arrived,

1: **if** there exists a wave c eligible for p such that p doesn't intersect any currently scheduled packet on c **then**
2:     schedule p on c
3: **else**
4:     let c be the earliest eligible wave for p
5:     **while** c is still eligible for p and p is not yet scheduled **do**
6:         let q be the first (i.e., leftmost) packet scheduled on c which intersects p
7:         **if** $|p| \leq |q|/2$ and $t_p \leq t_q$ **then**
8:             replace q by p $\triangleright$ p evicts q
9:         **end if**
10:         $c \leftarrow c + 1$
11:     **end while**
12: **end if**

$h-1$ packets. Let p be the h'th packet introduced. If p is scheduled on a wave c such that p doesn't intersect any currently scheduled packet on c, then the schedule remains feasible. Otherwise, assume for every wave c eligible for p, there are scheduled packets intersecting p on c. If p is not scheduled by MT, then clearly the schedule remains feasible. Otherwise, let c be the wave on which MT schedules p. Let S_p be the set of packets intersecting p on c, and let $q \in S_p$ be the first packet (i.e., leftmost packet) which intersects p on c. Since p is scheduled on c, by the condition in line 7 it follows that $t_p \leq t_q$, hence $S_p = \{q\}$, and therefore since q is evicted in favor of p, all the packets intersecting p on c are evicted, which results in a feasible schedule. ■

We now turn to show the competitive ratio guarantee of MT.

Theorem 2: Algorithm MT is an $O(\min\{\log \alpha, R\})$-competitive algorithm, where α is the ratio between the longest and shortest packets, and R is the number of different packet lengths.

Proof: Let A be the set of packets scheduled by MT and O be the set of packets scheduled by some optimal schedule. Denote by N the set of packets never scheduled on any wave by MT. We distinguish between two types of packets in $O \setminus A$.

Packets scheduled. Consider a packet $p \in (O \setminus A) \cap \overline{N}$. Let c be the wave on which p was scheduled. Packet p was not successfully sent by A, and therefore was evicted from c by some packet q. Note that this can only happen if the condition of line 7 is met. Note that q is not necessarily in A either, since q might have been later evicted by a packet q'. However, notice that continuing this scenario eventually results in a packet that is successfully sent by A, since the line is of finite length, and in every time step we have a finite number of packets' arrivals. Denote any such maximal sequence by $q_1, \ldots, q_k$, where q_1 is a packet scheduled on c without evicting any other packet, and q_k is a packet eventually sent by A. We therefore have, by the condition of line 7, $|q_{i+1}| \leq |q_i|/2$ for all $i = 1, \ldots, k-1$.

Let us map each such p to its corresponding q_k. Each p that maps to a specific q_k, is mapped via one of the packets q_i that evicts it. Notice that the proof of Lemma 1 implies that any such q_i is responsible for evicting at most one packet from $(O \setminus A) \cap \overline{N}$. We therefore have a one-to-one correspondence between packets in $(O \setminus A) \cap \overline{N}$ and packets in any such maximal sequence. Let us turn to bound the size of each sequence:

$$m \leq |q_k| \leq 2^{-(k-1)}|q_1| \leq 2^{-(k-1)}M$$

which in turn yields

$$k \leq \log \alpha + 1. \tag{1}$$

It follows that $|(O \setminus A) \cap \overline{N}| \leq (k-1)|A|$ since for each sequence we have one packet that is eventually sent by A.

Packets never scheduled. Consider a packet $p \in (O \setminus A) \cap N$. Packet p was never scheduled because on each wave c eligible for p the condition of line 7 was not met. This specifically holds for the wave c on which O schedules p. Assume first that the condition was not met in wave c because of excess length of p, i.e., if q is the packet scheduled by A on c specified by the algorithm in line 6, preventing p from being scheduled on c, then $|p| > |q|/2$. Notice that each such q can be responsible for the "non-scheduling" of at most 2 packets in $(O \setminus A) \cap N$ that suffer from excess length. This is because all such packets are successfully scheduled on c in the optimal schedule, and therefore do not intersect on c. In addition, there might be at most one packet in $(O \setminus A) \cap N$ that isn't scheduled by A because of its end point being later to that of the conflicting packet q scheduled by A. This is again because O produces a valid schedule, so there is at most one packet using c on any edge, specifically at most one using the edge leaving the endpoint of q. The same maximal sequences identified in the analysis of the packets in $(O \setminus A) \cap \overline{N}$ occur here. There are at most k such packets, where each one is "responsible" for the non-scheduling of at most 3 packets. It follows that $|(O \setminus A) \cap N| \leq (3k-1)|A|$.

We can now conclude the proof of the theorem.

$$\begin{aligned}
|O| &\leq |(O \setminus A) \cap N| + |(O \setminus A) \cap \overline{N}| + |A| \\
&\leq (k - 1 + 3k - 1 + 1)|A| \\
&\leq (4 \log \alpha + 3)|A| = O(\log \alpha)|A|.
\end{aligned}$$

Note that by the condition of line 7, $k \leq R$, hence combining this with Eq. (1) gives a competitive ratio of $O(\min\{\log \alpha, R\})$, which completes the proof. ■

MT has running time of $O(\delta n)$ per packet, where δ is the maximal slack of any packet in the sequence, and n is the network size. Note that MT need not know the values of α or R in advance.

C. A Tight Example for MT

We now give an example showing that the above analysis is tight, up to a constant factor. I.e., the above algorithm cannot achieve a performance superior to $\Omega(\log n)$. Assume that $n =$

2^k and let $r = k/2 - 1$. Define $x_i = \frac{1}{2^i}$. We therefore have $x_{i+1} = x_i/2$.

We consider two series of packets: $P = \{p_1, p_2, \ldots, p_r\}$ and $P' = \{p'_1, p'_2, \ldots, p'_r\}$, all with zero slack, where each packet is defined by its release time and its path:

- Packet p_i: release time $s_i = n(1 - x_i)$ and path $[s_i, n]$.
- Packet p'_i: release time $s'_i = n(1 - x_i) + 1 + i$ and path $[s'_i, s'_i + nx_{i+1} + 1]$.

Figure 2 shows an outline of the above sequence.

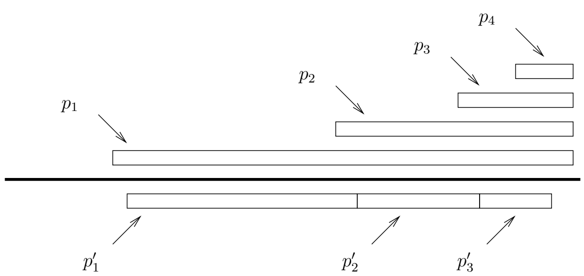

Fig. 2. Outline of the sequence showing MT is $\Omega(\log n)$-competitive.

Observation. For all i, $s_i < s'_i < s_{i+1}$. The first inequality follows from the definition, whereas the second follows from the fact that

$$\begin{aligned} s_{i+1} - s'_i &= -nx_{i+1} + nx_i - 1 - i \\ &= nx_{i+1} - 1 - i \\ &= 2^{k-(i+1)} - (i+1) > 0 \end{aligned}$$

for all $i \leq k/2 - 1 = r$.

Lemma 2: For every i, if p_i is scheduled by MT at the end of time s_i, then p'_i is rejected by MT.

Proof: Assume p_i is currently scheduled by MT. By the previous observation, the next packet in the sequence is p'_i. Since

$$|p_i| = nx_i = 2nx_{i+1} < 2(nx_{i+1} + 1) = 2|p'_i|,$$

then by the condition in line 7, p'_i is rejected by MT. ∎

Lemma 3: For every i, if p_i is scheduled by MT at the end of time s_i, then upon the arrival of p_{i+1}, MT preempts p_i and schedules p_{i+1} instead.

Proof: Assume p_i is scheduled by MT at the end of time s_i. By Lemma 2, p'_i, which is the next packet in the sequence, is rejected. The following packet is p_{i+1}, for which we have $|p_i| = x_i \geq 2x_{i+1} = 2|p_{i+1}|$, and in addition p_{i+1} doesn't terminate after p_i. By the condition in line 7, p_i is preempted by MT and p_{i+1} is scheduled in its place. ∎

Lemma 4: MT finishes scheduling only one packet from P, while there exists a scheduling that schedules all the packets in P'.

Proof: Since MT starts by scheduling p_1, then by Lemmas 2 and 3 it finishes scheduling only p_r. On the other hand notice that we can schedule all the packets in P'. Since the end point of p'_i is

$$\begin{aligned} s'_i + nx_{i+1} + 1 &= n(1 - x_i) + 1 + i + nx_{i+1} + 1 \\ &= n(1 - x_{i+1}) + 1 + (i+1) \\ &= s'_{i+1}, \end{aligned}$$

its path does not intersect with that of p'_{i+1}'s. ∎

Since $|P'| = \Omega(\log n)$, this example shows our analysis is tight up to a constant factor.

III. Non-Uniform Weights

A. Maximum Network Utilization

Assume that every packet p has weight $w_p = |p|$, and recall that our goal is to maximize the sum of the weights of delivered packets. This setting corresponds to optimizing network utilization. Unlike the case of uniform weights, the idea here is to prefer longer packets, which give a better utilization of the network. Let ϕ denote the golden ratio [2]. Consider the following algorithm for the problem, which we call MNU (see Algorithm 2 below).

Algorithm 2 Algorithm MNU

```
Given a new packet p just arrived,
 1: if there exists a wave c eligible for p such that p doesn't
        intersect any currently scheduled packet on c then
 2:     schedule p on c
 3: else
 4:     let c be the earliest eligible wave for p
 5:     while c is still eligible for p and p is not yet scheduled
            do
 6:         let S_p be the set of packets scheduled on c which
                intersects p.
 7:         if |p| ≥ φ · max_{q∈S_p} |q| then
 8:             replace S_p by p                    ▷ p evicts S_p
 9:         end if
10:         c ← c + 1
11:     end while
12: end if
```

MNU is an adaptation to our model of the algorithm given by Garay *et al.* in [9], for the problem of call admission, where a call's value is its route length.

We say packet p was *rejected by packet* q if q is the packet with maximal length in S_p, and p is rejected by the algorithm. In case more than one such packet exists, we choose one of them arbitrarily. We will sometimes abuse notation, referring to a packet as the set of its edges and to a set of edges as the set of intervals defined by them. Assume the packets arrived in the order $p_1, \ldots, p_k$. We first introduce some notation. For every $1 \leq i \leq k$, and every wave c, let $A^c(i)$ be the set of packets scheduled on c after the arrival of the i'th packet. For every packet $p \in A^c(i)$, let us denote the following:

[2] $\phi = \frac{1+\sqrt{5}}{2}$

- S_p^c - the set of packets preempted by MNU in order to schedule p (might be empty).
- T_p^c - the transitive closure of S_p^c. This set is defined immediately after p arrives and remains unchanged thereafter.
- $R_p^c(i)$ - the set of packets up to the i'th packet, rejected because of packets in $T_p^c \cup \{p\}$.
- $I_p^c(i)$ - the collection of all edges in the paths of packets in $T_p^c \cup R_p^c(i) \cup \{p\}$.

Lemma 5: For every wave c, every i, and every $p \in A^c(i)$,

$$I_p^c(i) \subseteq [s_p - \phi|p|, t_p + \phi|p|].$$

Proof: Clearly, the scheduling and preemption of packets on any wave c is of no consequence to packets scheduled on waves other than c. We may therefore deal with each wave independently. Let c be any wave. We prove the claim by induction on i. The claim trivially holds for $i = 0$. Assume the claim holds for $i-1$. Let p be the i'th packet that arrived. If c is not eligible for p then the claim clearly holds, so assume c is eligible for p.

Assume first that p is scheduled on c, and does not intersect any currently scheduled packet on c. In this case, for every packet $q \in A^c(i)$ other than p, $I_q^c(i) = I_q^c(i-1)$ and the induction hypothesis ensures the required result. For p, $I_p^c(i) = \{e | e \text{ is in } p\text{'s path}\}$, and the claim trivially holds.

Assume next that p is not scheduled on c. Let q be the packet responsible for rejecting p. Hence $q = \arg\max_{w \in S_p^c} |w|$. We need to show that $p \subseteq [s_q - \phi|q|, t_q + \phi|q|]$. Assume the contrary. Therefore p's path contains a point to the left of $s_q - \phi|q|$, or it contains a point to the right of $t_q + \phi|q|$. Since p was rejected because of q, clearly p and q intersect. Hence, $|p| > \phi|q|$, contradicting the fact that p was rejected because of q, and should therefore satisfy $|p| \leq \phi \cdot \max_{w \in S_p^c} |w| = \phi|q|$.

The last case to consider is the case where p is scheduled on c, and preempts the packets in S_p^c. We only need concern ourselves with p, as for every packet $q \in A^c(i)$ other than p, $I_q^c(i) = I_q^c(i-1)$. We will show that for every packet $q \in S_p^c$ preempted by p, $I_q^c(i-1) \subseteq [s_p - \phi|p|, t_p + \phi|p|]$, which will complete our proof. Since $q \in S_p^c$, p and q intersect. Furthermore, since q was preempted by p we have that $|q| \leq |p|$. One of the following must therefore be true: either $s_p \leq s_q < t_p$, or $s_p < t_q \leq t_p$. In both cases we have $[s_q - \phi|q|, t_q + \phi|q|] \subseteq [s_p - (|q| + \phi|q|), t_p + (|q| + \phi|q|)]$. Since $|p| \geq \phi|q|$, we have $|q| + \phi|q| \leq |p|/\phi + |p| = \phi|p|$, where the last equality follows from the defintion of ϕ. This concludes the proof of the lemma. ∎

We will use the above lemma, to analyze the performance of algorithm MNU.

Theorem 3: MNU is a $(2\phi+1)$-competitive[3] algorithm for the problem of maximum network utilization, where ϕ denotes the golden ratio.

Proof: An immediate consequence of Lemma 5 is the fact that for every wave c, every i, and every $p \in A^c(i)$, $|I_p^c(i)| \leq (1 + 2\phi)|p|$. Given a set X of packets, let $U(X) = \sum_{p \in X} |p|$. Consider the set of packets O scheduled in some optimal solution. Denote by M the set of packets that MNU schedules. Every packet in the sequence contributes its edges to at least one set $I_p^c(n)$, for some wave c, and some $p \in A^c(n)$ (since every packet is either scheduled, or was rejected or preempted). Moreover, every packet scheduled in an optimal schedule contributes its edges to at least one such set. We therefore have

$$\begin{aligned} U(O) &\leq \sum_{\text{wave} c} \sum_{p \in A^c(n)} |I_p^c(n)| \\ &\leq \sum_{\text{wave } c} \sum_{p \in A^c(n)} (1 + 2\phi)|p| \\ &= (1 + 2\phi) U(M) \end{aligned}$$

which completes the proof of the theorem. ∎

[3] $2\phi + 1 \sim 4.236$

Baruah *et al.* ([10]) present a lower bound of 4 for a problem of online task scheduling on a single machine, which applies to our model as well. It follows that any deterministic algorithm for our problem cannot have a competitive factor better than 4.

B. Arbitrary Weights

Assume without loss of generality that the minimum density of any packet, which we denoted by $\rho_{\min}$, is 1. We can scale all weights otherwise. Due to the lower bound for the problem of online task scheduling on a single machine appearing in [10], on the performance of any online deterministic algorithm, the following result is the best one could hope for, up to a constant factor.

Theorem 4: There exists an online algorithm for arbitrary weights with competitive ratio $O(\beta)$, where β is the ratio between the maximum and minimum densities of the packets in the input.

Proof: Apply MNU, thus seeking to maximize the network utilization. Let A be the set of packets scheduled by MNU. For every set of packets Y, define $U(Y) = \sum_{p \in Y} |p|$ and $w(Y) = \sum_{p \in Y} w_p$. Let O be the set of packets scheduled in some optimal solution. Let $c = (2\phi + 1)$ be the constant competitive ratio guaranteed by MNU. We thus have

$$\begin{aligned} w(A) &\geq U(A) \geq && (2) \\ &\geq \frac{1}{c} \cdot U(O) \geq && (3) \\ &\geq \frac{1}{c\beta} \cdot w(O) && (4) \end{aligned}$$

where (2) follows from our assumption that $\rho_{\min} = 1$, (3) follows from MNU's performance guarantee, and (4) is due to the fact that the weight of an optimal solution is bounded by the best network utilization solution, where all the scheduled packets have maximal density. ∎

Note that the algorithm depicted above need not know the value of β in advance.

IV. The Ring Topology

Our results readily extend to a ring network topology. To see this, notice that our algorithms for a linear network compute a packing of the packets on the waves. We therefore need only present an appropriate notion of waves for a ring topology, which we call *ring-waves*. Given these waves, our algorithms can be adapted in a straightforward manner to the ring topology.

A ring is characterized by an underlying digraph $G = (V, E)$, where $V = \{0, \ldots, n-1\}$ and $E = \{(i, i+1 \mod n) | 1 \leq i \leq n-1\}$. In the linear topology, we have an unbounded number of waves, each of finite length defined by the size of the network. In a ring topology, however, we have a finite number ring-waves, defined by the size of the network, where each ring-wave is of infinite length. Every ring-wave is specified by sequence of pairs (t, j), where t represents a time step, and j represents a node in the network. Ring-wave i corresponds to the sequence $\{(t, j) | t - j \mod n = i\}$. See Figure 3 for an illustration of the ring waves for a ring of size 6.

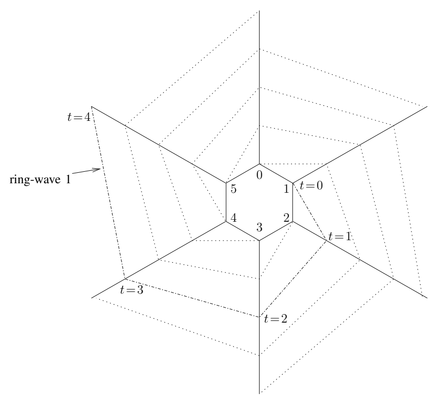

Fig. 3. Geometric interpretation of ring-waves for a network of size 6. The hexagon represents the ring, and the solid lines represent time. Each wave is represented by a dotted line.

V. Experimental Results

We conducted several simulations to examine the performance of MT for the problem of throughput maximization on a line network topology. In order to illustrate the performance of MT, we considered 3 algorithms for the problem:

- Algorithm MT - The algorithm specified in Section II-B.
- Algorithm OFFLINE - The offline algorithm of Adler *et al.* ([1]). This algorithm loops through the waves and computes a maximum independent set on each wave, i.e., it tries to mount as many packets as possible on the given wave, considering only packets for which the wave is eligible. Packets that are mounted are discarded in subsequent iterations. As was shown in [1], the schedule produced by OFFLINE is guaranteed to successfully deliver at least half the maximum number of packets delivered by any optimal schedule. We use the output of this algorithm as an estimate to the performance of an optimal schedule.
- Algorithm URGENT - The algorithm given in Algorithm 3 specified below. The intuition underlying this algorithm is the following: At a given time t, a packet which has not yet been scheduled is considered *urgent* if its residual slack at time t is zero. In every node i the algorithm might have at most 2 packets contending for the outgoing link at any time t: One that is en-route and has arrived from node $i-1$, and a *pending* packet whose source is i, with minimum residual slack among all the packets with source i. The algorithm prefers the packet en-route, unless its pending packet is urgent. The algorithm will preempt the packet-en route q for such a pending packet only if q wasn't urgent when it left its source.

Algorithm 3 Algorithm URGENT

At any given node i with buffer B_i, at any time t

Insert all packets p such that $r_p = t$ and $s_p = i$ into B_i.
Let $S(i, t)$ be the set of packets with minimum slack among the packets currently in B_i.
Let $A(i, t)$ be the set of packets sent
    from node $i-1$ at time $t-1$ whose
    destination is $j > i$. ▷ $|A(i, t)| \leq 1$
if $S(i, t)$ isn't empty **then**
    Let $p = \arg\min_{q \in S(i,t)} |q|$.
    if $A(i, t)$ isn't empty **then**
        Let $q \in A(i, t)$.
        if $\ell(p) = 0$ and $d_q - t > t_q - i$ **then**
            Schedule p to leave i at time t.
        else
            Schedule q to leave i at time t.
        end if
    else ▷ $A(i, t)$ is empty
        schedule p to leave i at time t.
    end if
else ▷ $S(i, t)$ is empty
    if $A(i, t)$ isn't empty **then**
        Schedule $q \in A(i, t)$ to leave i at time t.
    end if
end if
Remove all scheduled and expired packets from B_i.

A. Sequence Generation

In our simulations, we use the release time to control the traffic intensity. In every time step t we release a quantity of λ packets to every node. For each such packet we choose uniformly at random its path length [4]. For every packet p, we choose its slack to be $\kappa \cdot |p|$, where κ is chosen uniformly at random from some set $\{1, \ldots, m\}$. Once these parameters are set, they imply the packet's deadline and target node. We refer to λ as the *overload parameter*, and to m as the *slack factor*. We denote a sequence generated in this manner by $\sigma(\lambda, m)$. As the overload parameter increases, the amount of overload encountered by the network increases, and as the slack factor

[4] Notice that in a line network topology, as the node index increases, the maximum possible path length is shorter.

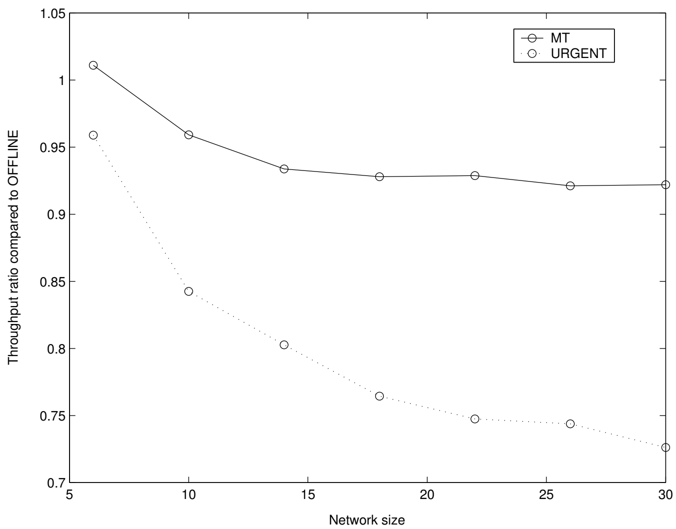

Fig. 4. Performance of MT and URGENT as a function of network size (overload parameter= 1, slack factor= 2)

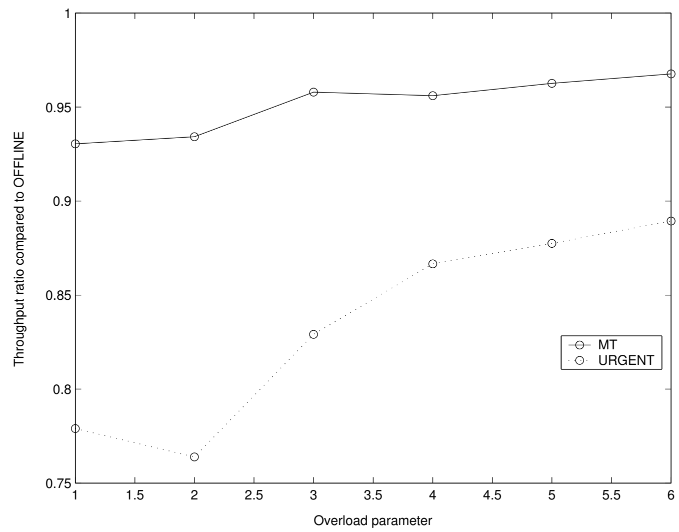

Fig. 5. Performance of MT and URGENT as a function of overload ($n = 16$, slack factor= 2)

increases, packets might have bigger slack. Given a network size, we generate a $\sigma(\lambda, m)$ sequence. This sequence is given as input to OFFLINE. When executing MT and URGENT, the sequence is given in an online fashion, such that the algorithm has no information about packets released after the current clock tick.

B. Results

We conducted several trials of every experiment. The plots in this section correspond to the average of the results obtained. When comparing the performance of either MT or URGENT, we compare the ratio between the number of packets successfully scheduled by the algorithm, and the number of packets successfully scheduled by OFFLINE. We present results of two types of experiments: experiments comparing the performance of MT and URGENT, and experiments investigating the performance of MT in several network settings, namely its performance under various overloads, and its performance under varying slack bounds.

Figure 4 shows a comparison of the performance of MT and URGENT as the network size increases. We consider networks of varying sizes, from 6 to 30. The input given to both algorithms are $\sigma(1, 2)$ sequences, i.e., one packet is released to every node at every clock tick, and for every packet p in the sequence, $\ell(p) \in \{|p|, 2|p|\}$. We can see that MT maintains its performance of 90-95% compared to OFFLINE as the network size increases while the performance of URGENT deteriorates monotonically to 70-75% compared to OFFLINE.

Figure 5 shows the difference between the performance of MT and that of URGENT, as the network becomes more and more overloaded. We consider a network of size 16 and generate input sequences $\sigma(\lambda, 2)$, where the overload parameter λ varies from 1 to 6, while allowing each packet p to have slack in $\{|p|, 2|p|\}$. The plots show that as might be expected, as the network becomes more and more overloaded, the performance of both algorithms improves. Note that in this case as well, the performance of MT compared to OFFLINE is still better than the performance of URGENT compared to OFFLINE by more than 20% for low overload. This gap decreases, as the overload increases, down to ~8% for highly overloaded networks, as both algorithms improve their performance.

The simulation results in Figure 6 show the behavior of MT and URGENT, as packets are allowed to have larger slacks. We consider a network of size 16, and generate input sequences $\sigma(1, m)$, for a slack factor $m = 1, \ldots, 10$, thus releasing one packet at every node at every clock tick. The results show that MT outperforms URGENT as packets are allowed to have greater slack. Furthermore, the performance of MT compared to OFFLINE remains in the range of 92-94%, with a slight monotonic increase, while the performance of URGENT compared to OFFLINE degrades monotonically as the slack factor increases, dropping from ~80% for a slack factor of 1, to ~75% for slack factor of 10.

We further investigate the tolerance of MT to varying network settings, as the network size increases. Figure 7 shows the performance of MT where we let the slack factor take values 1, 3, and 5. We consider network sizes from 6 to 30, where we take the overload parameter to be 1. We can see that the performance of the algorithm differs by $\sim 3\%$ under the various slack factor bounds, and the performance is mainly dominated by the network size.

The effect of increasing network overload on the performance of MT is demonstrated in Figure 8, where we consider overload parameters 1, 3, and 5. We examine networks of sizes 6 to 30, with slack factor 2. Note that as in the previous experiment, as the network becomes more overloaded the performance improves. In addition we see the degradation in performance, as the network size increase, maintains its pro-

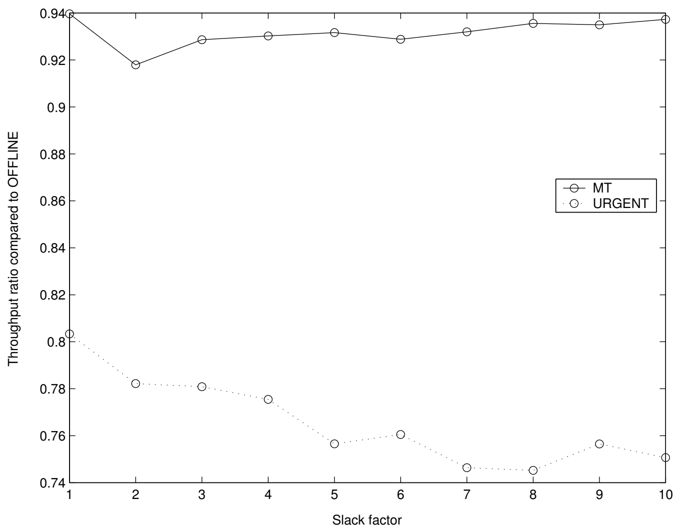

Fig. 6. Performance of MT and URGENT as a function of maximum slack factor bound ($n = 16$, overload parameter= 1)

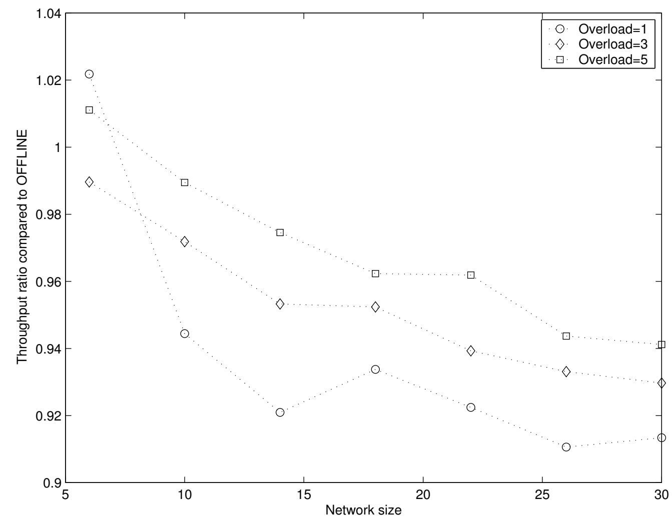

Fig. 8. Performance of MT under variable overloads, as a function of network size (slack factor= 2)

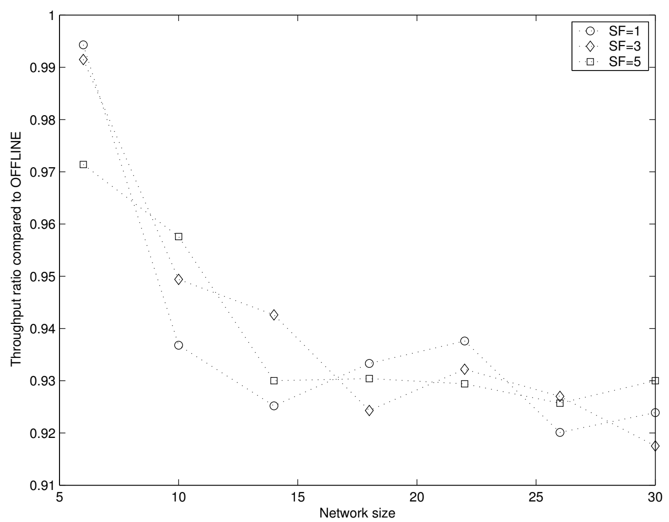

Fig. 7. Performance of MT under variable maximum slack bounds, as a function of network size (overload parameter= 1)

portions under the different overload parameters considered.

VI. Discussion

We have presented the first online algorithms for the problem of bufferless time-constrained scheduling of packets in a linear network. These results extend to the ring topology as well. For the problem of maximum throughput, i.e., when packets have uniform weights, our algorithm achieves a competitive ratio of $O(\min\{\log \alpha, R\})$, where α is the ratio between the longest and shortest path lengths a packet has, and R is the number of different lengths of packet paths appearing in the input sequence. We additionally show that no online deterministic algorithm can achieve a competitive ratio better than 2 for this setting. We present a constant competitive algorithm for the problem of maximizing network utilization, where the weight of each packet is its length. For the case of arbitrary packet weights we give an algorithm with competitive ratio $O(\beta)$, where β is the ratio between the maximum and minimum weight-to-length ratios. Our algorithms for these cases are optimal up to a constant factor.

Our experimental results show that our algorithm for the problem of throughput maximization performs much better than its worst case guarantee, for randomly generated input sequences. Its performance is very close to the performance of an offline algorithm, which is guaranteed to schedule at least half the number of packets scheduled in an optimal schedule. Our algorithm also outperforms an intuitive online greedy algorithm, which prefers to schedule urgent packets first.

It would be interesting to try and close the gap between the upper and lower bounds for the problem of throughput maximization, as well as to see how rescheduling can effect the performance of such algorithms.

VII. Acknowledgements

We thank Danny Raz for useful discussions.

References

[1] M. Adler, A. L. Rosenberg, R. K. Sitaraman, and W. Unger, “Scheduling time-constrained communication in linear networks,” *Theoretical Computer Science*, vol. 35, no. 6, pp. 599–623, 2002.

[2] J. Liebeherr, “Multimedia networks: Issues and challanges,” *IEEE Computer*, vol. 28, no. 4, pp. 68–69, 1995.

[3] J. Rexford, J. Hall, and K. G. Shin, “A router architecture for real-time communication in multicomputer networks,” *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1088–1101, 1998.

[4] M. Adler, S. Khanna, R. Rajaraman, and A. Rosén, “Time-constrained scheduling of weighted packets on trees and meshes,” *Algorithmica*, vol. 36, no. 2, pp. 123–152, 2003.

[5] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[6] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[7] G. Koren and D. Shasha, "D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems," *SIAM Journal on Computing*, vol. 24, no. 2, pp. 318–339, 1995.

[8] J. A. Garay, J. Naor, B. Yener, and P. Zhao, "On-line admission control and packet scheduling with interleaving," in *Proceedings of the IEEE INFOCOM'02*, New York, NY, June 2002, pp. 94–103.

[9] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, "Efficient on-line call control algorithms," *Journal of Algorithms*, vol. 23, no. 1, pp. 180–194, 1997.

[10] S. K. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. E. Rosier, D. Shasha, and F. Wang, "On the competitiveness of on-line real-time task scheduling," *Real Time Systems*, vol. 4, no. 2, pp. 125–144, 1992.

[11] R. J. Lipton and A. Tomkins, "Online interval scheduling," in *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 302–311.

[12] R. Adler and Y. Azar, "Beating the logarithmic lower bound: Randomized preemptive disjoint paths and call control algorithms," *Journal of Scheduling*, vol. 6, no. 2, pp. 113–129, 2003.

[13] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén, "On-line competitive algorithms for call admission in optical networks," *Algorithmica*, vol. 31, no. 1, pp. 29–43, 2001.