Competitive Buffer Management with Packet Dependencies

Alex Kesselman
Google Inc.
Mountain View, CA, USA
alx@google.com

Boaz Patt-Shamir
School of Electrical Engineering
Tel Aviv University
Tel Aviv 69978, Israel
boaz@eng.tau.ac.il

Gabriel Scalosub
Department of Computer Science
University of Toronto
Toronto, ON, Canada
scalosub@cs.toronto.edu

Abstract

We introduce the problem of managing a FIFO buffer of bounded space, where arriving packets have dependencies among them. Our model is motivated by the scenario where large data frames must be split into multiple packets, because maximum packet size is limited by data-link restrictions. A frame is considered useful only if sufficiently many of its constituent packets are delivered. The buffer management algorithm decides, in case of overflow, which packets to discard and which to keep in the buffer. The goal of the buffer management algorithm is to maximize throughput of useful frames. This problem has a variety of applications, e.g., Internet video streaming, where video frames are segmented and encapsulated in IP packets sent over the Internet. We study the complexity of the above problem in both the offline and online settings. We give upper and lower bounds on the performance of algorithms using competitive analysis.

1 Introduction

Classical Queuing Theory [16], as well as modern Adversarial Queuing Theory [10], study the conditions under which the system is stable, i.e., the queue sizes remain bounded. Ideally, a stable system can be designed so that buffer overflows virtually never occur, and hence the issue of buffer overflows was largely overlooked by these disciplines. However, in the Internet, buffer overflows occur quite often (being intentionally generated by TCP). In response, a new model to study overflows was defined recently [14, 17], and quite a few results were obtained (see, for example, [1, 3, 5–7, 11, 15, 20]). In this paper we extend this line of research with a new model and new results.

Specifically, we consider the following scenario. There is a FIFO buffer which can hold B packets. The packets arrive at the buffer according to an adversarial process, but only one packet can be sent out of the buffer in each time step. Therefore, overflows may occur, and the buffer management algorithm tries to minimize their damage. This is the model proposed in [14,17]. The new variant we consider here models dependencies among the arriving packets. In particular, we consider the common case where the arriving data stream originally consists of large *frames*, while the data link can carry only small packets, and therefore each frame must be fragmented into a few packets. This scenario introduces dependencies, because, for example, in many cases a frame is useless unless all its constituent packets are delivered. It follows that in this common case, deciding which packet to drop may influence the decision about packets that may arrive only in the distant future, giving rise to new algorithmic questions. For example, it turns out that some natural algorithms perform very poorly in the context of packets with dependencies, whereas other algorithms work relatively well.

In this paper we make a few first steps in the direction of studying packet dependencies, which include introducing the model and proving a few preliminary results. In particular, we provide near-optimal off-line and on-line algorithms for the case where all packets need to be delivered for a frame to be useful. Before we can state our results more meaningfully, we need to define our model.

1.1 The Model

The input to the system consists of a sequence of unit-size *packets*. We assume that the packets are derived from a sequence of *frames*, where a frame f_i comprises a set of packets denoted $p_i^1, \ldots, p_i^n$. A packet p_i^j is referred to as the j-packet of frame f_i. The system progresses in discrete time steps, where in each step an arbitrary set of packets arrive. The arrival step number of a packet p is denoted by $\mathrm{arr}(p)$. It is assumed that for each frame index i we have $\mathrm{arr}(p_i^j) \leq \mathrm{arr}(p_i^{j+1})$ for $1 \leq j < n$, i.e, the packets of each frame arrive in order.

The packets arrive at a FIFO buffer denoted Q. The buffer can contain at most B packets, and has an output link that can transmit one packet per step. Specifically, an execution proceeds as follows. Initially, the buffer is empty. Each step consists of three substeps. The first substep is the *delivery substep*: if the buffer is non-empty, the head-of-the-line packet is transmitted on the link (otherwise nothing happens in this substep). In the second substep, called the *arrival substep*, an arbitrary set of packets arrives at the system. Finally, at the discretion of the buffer management algorithm, some packets may be dropped. This substep is referred to as the *drop substep*. An algorithm which drops only newly arriving packets is called an *admission control* algorithm; an algorithm which may also discard packets that arrived earlier and are currently in the buffer is called *preemptive*. In any case, the FIFO order of the surviving packets is maintained. A feasible system satisfies the following *capacity constraint*: the maximum number of packets in the buffer between consecutive time steps (i.e., after the drop substep) must not exceed the given buffer size B. A buffer management algorithm may drop packets even if there is space available at the buffer. A *schedule* produced by a buffer management algorithm is an assignment of packets to time steps, such that a packet assigned to time t, is transmitted (or equivalently, delivered) in time t. Given an arrival sequence, we sometimes identify an execution of an algorithm with the schedule of its transmitted packets. Given any buffer management algorithm ALG, and every packet p, we let $s_{\mathrm{ALG}}(p)$ denote the time step in which the algorithm delivers packet p.

In the **k-of-n frame throughput maximization problem**, denoted k-of-n FTM, the aim of the buffer management algorithm is to maximize the number of *successfully delivered* frames, where a frame is considered successfully delivered if at least k out of its n constituent packets are delivered. A special case of this problem is the case where $k = n$, which we refer to as the **k-frame throughput maximization problem**, and denote by k-FTM. In this problem a frame is considered to successfully delivered if *all* of its constituent packets are delivered.

We use competitive analysis [9, 21] to evaluate the performance of online algorithms. An algorithm ALG is said to be *c-competitive* if for all traffic arrival sequences σ, the maximal number of frames successfully delivered by *any* feasible schedule is at most c times the number of frames delivered by ALG from σ, for some $c \geq 1$. As customary in competitive analysis, we may view the on-line algorithm as competing against an off-line *adversary* that generates the input stream, and provides an optimal schedule for that input. Given an algorithm ALG, we will sometime abuse notation and refer to ALG also as the set of packets or frames delivered by ALG.

1.2 Our Results

Most of our results are for the k-FTM problem. In Section 2, as a gentle warm-up, we look at the offline version of k-FTM. On the one hand we show that approximating k-FTM to within $o(k/\ln k)$ is NP-hard for $k \geq 3$, even for $B = 1$. On the other hand we give a simple algorithm which is guaranteed to produce a $(k+1)$-approximate solution to the problem, for any buffer size. In Section 3 we study online algorithms for k-FTM. We first prove that no algorithm can have bounded competitive ratio for general k-FTM even when $k = 2$. We therefore propose a certain natural condition on the ordering of packets in the arrival sequence, and proceed to show that the competitive factor for such "order respecting" arrival sequences is between $O(k^2)$ and $\Omega(k)$. Our online algorithm used to prove the upper bound is non preemptive: no packet admitted to the buffer is ever dropped. We also investigate a natural greedy algorithm, and show that while it is $O(1)$-competitive for the 2-FTM problem, its competitive factor for k-FTM problem with $k \geq 3$ is unbounded. In Section 4 we consider some special cases of the k-of-n FTM problem. We conclude with open problems in Section 5.

1.3 Previous Work

As mentioned earlier, there has been extensive work dealing with management of buffer overflows. Many of these works study systems that should provide some Quality-of-Service (QoS) guarantees to the underlying traffic subject to the buffer capacity and FIFO order constraints. In the delay-oriented approach, each packet has a deadline by which it must be delivered. In the FIFO model, it is typically assumed that each packet has a value, and the goal of the algorithm is to maximize the total value of delivered packets [14, 17]. In the case of a single buffer, the best known competitive ratio for algorithms under general values is $\sqrt{3} \approx 1.732$, and the best lower bound is $1 + 1/\sqrt{2} \approx 1.707$ [12]. If packets have only one of two values, 1 and $\alpha > 1$, then the competitive ratio is roughly 1.3 [12], and this is optimal [4].

Our definition of the k-of-n FTM problem is motivated by traffic with forward error-correction (FEC), which can tolerate some losses. FEC is also useful to get shorter delays by using more bandwidth. There are many papers discussing different methods of implementing such an approach in the coding and information theory communities (e.g., [2, 8, 18]), where the predominant application is encoding schemes for video streaming (see, e.g., [19]). Our model provides an abstraction of the buffer overflow management problem in such systems, and our algorithms can be seen as treating such settings from a more systems-oriented viewpoint.

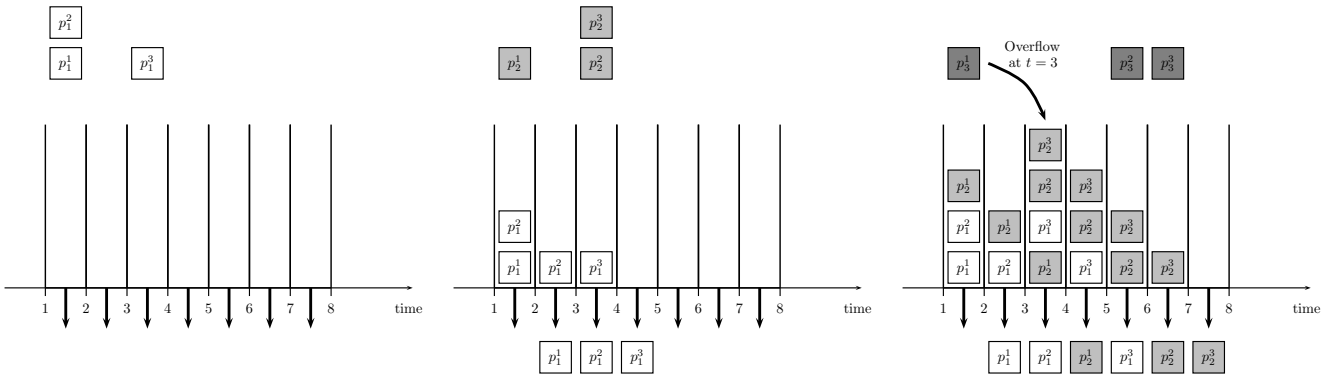

Figure 1. An example of Algorithm G-OFF with $B = 4$ and $k = 3$, depicting buffer occupancy in every time step. The algorithm first adds frame f_1 whose packets (white) arrive as described on the left, resulting in the schedule depicted in the middle. Then f_2 (light gray) is added. Finally, f_3 (dark gray) cannot be added due to an overflow of p_3^1 that would occur at time 3.

2 Offline k-FTM

In this section we consider the offline version of k-FTM. Corollary 2 proves that it is NP-hard to approximate this problem to within $o(k/\log k)$, and Theorem 3 shows that it can be approximated to within a factor of $k + 1$.

2.1 A Lower Bound

We now show that a special case of k-FTM is hard to approximate, by giving an approximation-preserving reduction from k-dimensional matching. In k-dimensional matching, denoted k-DM henceforth, there are k pairwise disjoint sets $V_1, \ldots, V_k$, and the input consists of a set $S \subseteq V_1 \times \cdots \times V_k$. The goal is to find a maximum subset $M \subseteq S$ such that every element $x \in \bigcup_{i=1}^{k} V_i$ appears in at most one element of M. For $k = 2$ the problem is simply bipartite matching, which can be solved in polynomial time, but for $k \geq 3$, k-DM cannot be approximated to within $o\left(\frac{k}{\ln k}\right)$ unless P=NP [13].

We now give the reduction from k-DM to k-FTM with $B = 1$. Note that in our model, $B = 1$ means that in each time step the algorithm can keep only one packet to transmit in the following step.

Lemma 1. *The special case of k-FTM where $B = 1$ is equivalent to k-DM.*

Proof. Let $S \subseteq V_1 \times \cdots \times V_k$ be an instance of k-DM. We construct an instance of k-FTM as follows. Map each element $x \in \bigcup_{i=1}^{k} V_i$ to a distinct time step t_x such that if $x \in V_i$ and $y \in V_j$ for $j > i$ then $t_x < t_y$. This way, each member $e \in S$ corresponds to k times steps. We define these time steps to be the arrival times of packets of a single frame corresponding to e. Having done this for all elements in S, we obtain an arrival sequence $\sigma(S)$. Consider a feasible schedule for $\sigma(S)$: Since $B = 1$, any two packets delivered by a feasible schedule arrive at different time steps. Therefore, the set of complete frames delivered by a feasible schedule for $\sigma(S)$ uniquely defines a k-dimensional matching for S and vice versa. Hence the maximal number of delivered frames in $\sigma(S)$ is exactly the maximal number of elements in the matching for S, and we are done. □

Lemma 1, in conjunction with the hardness result of [13], implies the following.

Corollary 2. *Unless P=NP, no poly-time algorithm for k-FTM can guarantee $o(k/\ln k)$ approximation factor.*

2.2 An Upper Bound

The offline version of k-FTM is easy to approximate by the following greedy algorithm, denoted G-OFF. Algorithm G-OFF constructs the schedule iteratively by scanning the frames in arbitrary order, and adding each frame to the current schedule if none of its packets violate the buffer capacity constraint. See Figure 1 for an example.

Theorem 3. *Algorithm G-OFF is a $(k + 1)$-approximation for the k-FTM problem.*

Proof. Fix an input arrival sequence σ, and let OPT be an optimal schedule for σ. Consider a frame $f_i \in \text{OPT} \setminus \text{G-OFF}$: if $f_i \notin$ G-OFF then at least one of its packets would have caused an overflow, and could not be accepted into the buffer by G-OFF, whereas all of f_i's packets are accepted by OPT. When considering the buffer occupancy of G-OFF compared to that of OPT, it follows that in

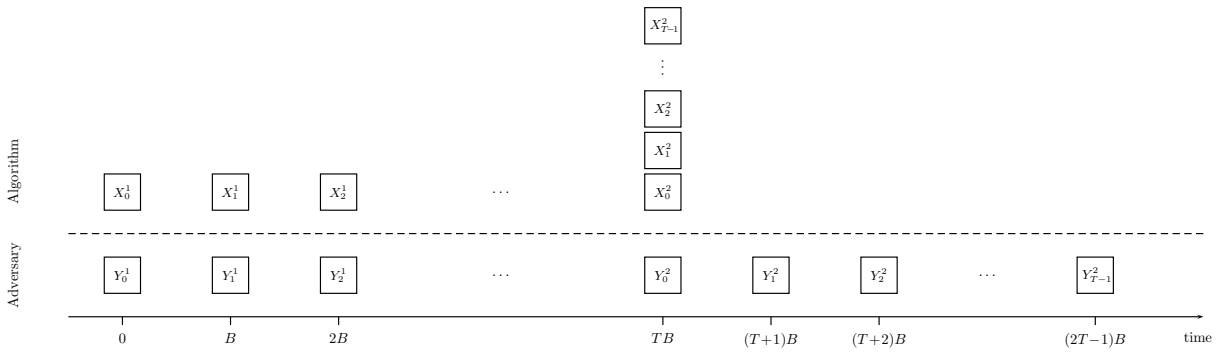

Figure 2. Outline of lower bound described in Theorem 4. Each square represents the arrival of B packets. Squares above the dashed line represent packets corresponding to the algorithm's frames, and squares below the dashed line represent packets corresponding to the adversary's frames.

at least one timestep, there exists some location in the buffer of G-OFF which contains some packet p corresponding to some frame previously accepted by G-OFF, whereas at the same time, and at the same location, OPT has stored one of the packets corresponding to f_i. The above follows from a pigeonhole argument, since otherwise G-OFF would have had sufficient buffer space resources to accept all packets of f_i. We construct a mapping where we map each such frame f_i to one such packet p. We note that regardless of the packets we choose for determining the mapping, the resulting mapping is a bijection, since by feasibility of both OPT and G-OFF, any location at any specific time may be occupied by a single packet, and both algorithms are non-preemptive.

It therefore follows that any packet delivered by G-OFF accounts for at most one frame which is mapped to it according to the above procedure. It therefore follows that the number of frames in $\mathrm{OPT} \setminus \text{G-OFF}$ accounted for by any frame successfully delivered by G-OFF is at most k, which completes the proof. □

3 Online k-FTM

In this section we study the online version of k-FTM. We start by showing that for unrestricted arrival sequences, no deterministic algorithm can have a bounded competitive ratio. We then proceed to propose a natural restriction on the order of packets in an arrival sequence, and give upper and lower bounds on the competitive ratio under that constraint.

3.1 Order is Essential

Theorem 4. *No online algorithm has bounded competitive ratio for general k-FTM, even for $k = 2$.*

Proof. Let T be some positive integer, and consider the following arrival sequence. For $i = 0, \ldots, T-1$ we have $2B$ 1-packets arrive at time $t_i = iB$. Let X^1_i denote the set of 1-packets accepted by the algorithm at time t_i. Then $\left|X^1_i\right| \leq B$, and the algorithm drops at least B packets at time t_i. For every such i, denote by Y^1_i some B 1-packets out of this set of dropped packets.

For $i = 0, \ldots, T-1$, let X^2_i and Y^2_i denote the set of 2-packets corresponding to X^1_i and Y^1_i, respectively. At time TB we have all 2-packets in $\bigcup_{i=0}^{T-1} X^2_i$ arrive. The algorithm can accept at most B of these 2-packets, and can therefore deliver an overall of at most B complete frames. For every $i = 0, \ldots, T-1$ we have all 2-packets of Y^2_i arrive at time $s_i = TB + iB$. It follows that the adversary can accept all 1-packets in $\bigcup_{i=0}^{T-1} Y^1_i$, as well as the set of all their corresponding 2-packets $\bigcup_{i=0}^{T-1} Y^2_i$. It can therefore successfully deliver an overall of TB complete frames. Since T can be arbitrarily large, we conclude that no online algorithm can have a bounded competitive ratio, even for $k = 2$. Figure 2 gives an outline of the arrival sequence, and the packets accepted by the algorithm and the adversary. □

Theorem 4 motivates us to consider restricted adversaries. Specifically, we consider *order-respecting* adversaries, in which the arrival order of j-packets is independent of j. Intuitively, an arrival sequence is order-respecting if the order of frames induced by considering solely j packets, is the same as the order of frames when considering solely j' packets, for every j and j'. The above is captured by the following formal definition.

Definition 5. *An input sequence is* order-respecting *if for all* i, i', j, j'*:* $\mathrm{arr}(p^j_i) \leq \mathrm{arr}(p^j_{i'}) \iff \mathrm{arr}(p^{j'}_i) \leq \mathrm{arr}(p^{j'}_{i'})$.

Note that the adversary used in the proof of Theorem 4 is not order-respecting: the frame order of 1-packets and

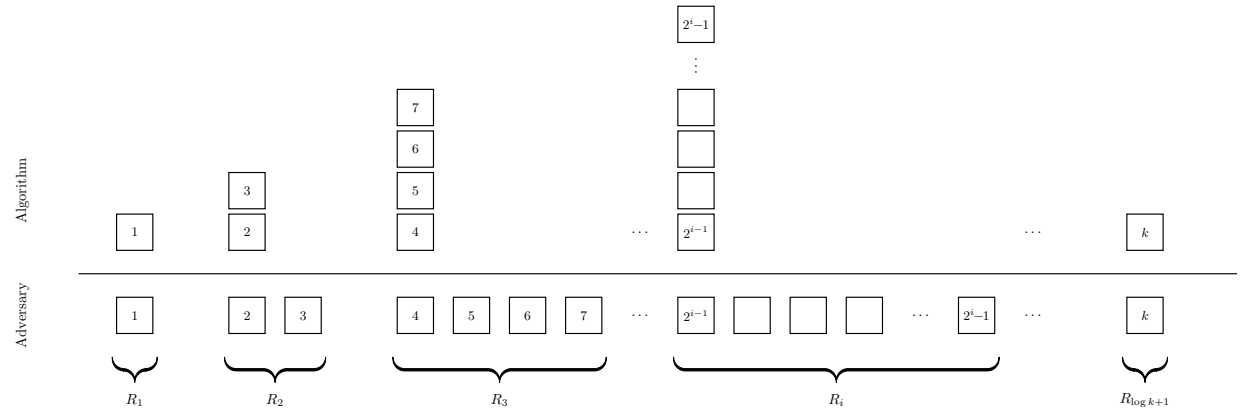

Figure 3. Outline of lower bound described in Theorem 6. Each square marked j represents the arrival of B j-packets. Squares above the time axis represent packets of the algorithm's frames, and squares below the time axis represent packets of adversary's frames.

the frame order of 2-packets differ. To see this consider for example the set of 1-packets Y_1^1 which arrives strictly before the set of 1-packets X_2^1. When considering the 2-packets of these frames, we see that the 2-packets in the set Y_1^2 arrive strictly after the 2-packets in the set X_2^2.

3.2 Lower Bounds for Order Respecting Adversaries

We now show that even order-respecting arrivals are non-trivial to manage.

Theorem 6. *Any deterministic algorithm for k-FTM has competitive ratio $\Omega(k)$, even for order-respecting arrival sequences.*

Proof. Assume that k is a power of 2, and consider the following arrival sequence, consisting of $\log k + 1$ blocks, $R_1, \ldots, R_{\log k+1}$. At the beginning of R_1, $2B$ 1-packets arrive simultaneously. Any feasible algorithm can accept at most B of these packets, and hence must drop at least B 1-packets, which imply forfeiting B frames. We refer to the frames corresponding to B such 1-packets dropped by the algorithm as the *adversary's frames* and to the remaining B frames as the *algorithm's frames*.

For every $i = 2, \ldots, \log k$, block R_i consists of the arrival of all j-packets, for $j = 2^{i-1}, \ldots, 2^i - 1$ as follows: At the beginning of the block we have a burst of all j-packets of the algorithm's frames, for $j = 2^{i-1}, \ldots, 2^i - 1$. After this burst, the j-packets of the adversary's frames arrive one by one, respecting the overall frame order, i.e., we first have a stream of 2^{i-1}-packets, followed by a stream of $2^{i-1} + 1$-packets, and so on, ending by a stream of $2^i - 1$-packets, where for each such block of j-packets, their order is according to the overall frame order. Finally, $R_{\log k+1}$ consists of the arrival of all the k-packets of all frames, arriving one by one, respecting the overall frame order. Note that the above arrival sequence is indeed order-respecting. Figure 3 gives an outline of the arrival sequence.

The frames that have not yet been dropped by the algorithm at the end of a block R_i are called *live frames* at that time. We claim, by induction on the block number, that the number of live frames at the end of R_i is at most $B/2^{i-1}$. The claim clearly holds for $i = 1$. Assume it holds for $i - 1$, and consider the arrival of packets in block R_i. By the induction hypothesis, at the beginning of block R_i the algorithm has at most $B/2^{i-2}$ live frames. By the definition of block R_i, all the j-packets, for $j = 2^{i-1}, \ldots, 2^i - 1$, corresponding to these live frames, arrive together at the beginning of block R_i. This yields a total of $2^{i-1} \cdot B/2^{i-2} = 2B$ packets corresponding to the live frames of the algorithm. The algorithm may only accept B out of these $2B$ packets. In order for a frame to remain a live frame at the end of the block, all of its packets arriving at the beginning of the block must be accepted by the algorithm. Hence, at most $B/2^{i-1}$ live frames can survive at the end of block R_i, out of the $B/2^{i-2}$ live frames at the end of the previous block.

It follows that at the end of block $R_{\log k}$, the algorithm can maintain at most $B/2^{\log k-1} = 2B/k$ live frames. The adversary, on the other hand, has none of its packet arrivals in a bursty manner, and thus can deliver all of its B frames. It follows that the ratio between the number of frames delivered by the adversary and the number of frames delivered by the algorithm is at least $k/2$.

The above argument holds for any value of k which is a power of 2. For arbitrary k one loses at most an extra factor of 2. □

The above argument can be adjusted to provide a stronger lower bound for the case of k-packet frames, where $k = 2, 3$, as the following lemma shows.

Lemma 7. *No deterministic algorithm for k-FTM can have a competitive ratio better than 2, even if we only allow 2-frames, and even for order-respecting arrival sequences.*

Proof. Consider the following arrival sequence, comprising solely of 2-frames: At time $t = 0$, $2B$ 1-packets arrive. Any feasible algorithm can accept at most B of these packets, and hence must forfeit at least B frames. We refer to the frames corresponding to B such 1-packets dropped by the algorithm as the *adversary's frames* and to the remaining B frames as the *algorithm's frames*.

At time B we have additional B 1-packets arriving (we refer to the frames to which these 1-packets correspond as the *new frames*), along with the B 2-packets corresponding to the algorithm's frames. At time $2B$ we have the B 2-packets corresponding to the adversary's frames arrive, and at time $3B$ we have the B 2-packets corresponding to the new frames arrive. Note that the algorithm may accept at most B frames out of the set of its frames and the new frames, whereas the adversary can accept both all its frames, as well as all the new frames. □

3.3 The Static-Partitioning Algorithm

In this section we show that a simple admission control algorithm achieves $O(k^2)$ competitive ratio for k-FTM. A nice feature of our algorithm is that it is non-preemptive, i.e., no packet that enters the buffer is ever evicted. This allows for efficient implementation, in contrast to preemptive algorithms.

Our algorithm, called Static-Partitioning Algorithm (SPA), aims at ensuring that for any $j = 1, \ldots, k$, sufficiently many j-packets are delivered, where these j-packets are relatively evenly spaced. This ensures that eventually, sufficiently many of their corresponding k-packets are also delivered. Specifically, SPA virtually partitions its buffer into k sub-buffers, each of size $\frac{B}{k}$, where the j'th sub-buffer is dedicated solely to j-packets (and referred to as the *j-buffer*). We emphasize that the buffer is a FIFO queue, and the partition is virtual, implemented by a simple counter for each sub-buffer.

To describe the algorithm, we define the key concept of *j-synchronization* for $j = 0, \ldots, k$. Let the input frames be $f_1, \ldots, f_\ell$. It is convenient to extend the input sequence with a fictitious frame $f_{\ell+1}$ satisfying $\mathrm{arr}(p^j_{\ell+1}) = \mathrm{arr}(p^j_\ell) + B$ for all $j = 1, \ldots, k$. Now, a j-synchronization is a frame index, and we let $i(m, j)$ denote the frame index of the mth j-synchronization (j is a packet number within the frame, and m is the running number of the j-synchronizations). We define j-synchronizations inductively in the following way:

- Every frame index is a *0-synchronization*: $i(m, 0) = m$ for all $1 \le m \le \ell + 1$.
- For $j > 0$, the *j-synchronization* $i(m, j)$ is defined as follows: $i(1, j) = 1$, and $i(m, j)$ for $m > 1$ is the smallest $(j-1)$-synchronization index i, such that $\mathrm{arr}(p^j_i) \ge \mathrm{arr}(p^j_{i(m-1,j)+\frac{B}{k}-1}) + B$, namely the first $(j-1)$-synchronization index i for which its j-packet arrives after the arrival of $B/k - 1$ additional j-packets after the previous j-synchronization, and B additional time steps. If no such frame exists, then the j-synchronization is defined to be $\ell + 1$.

A packet $p^j_{i(m,j)}$ is called a *j-synchronization packet*.

Let A_j denote the set of frame indices which are j-synchronizations. Then $A_j \subseteq A_{j-1}$ for every $j = 1, \ldots, k$. Also, $\{1, \ell + 1\} \subseteq A_j$ for every $j = 1, \ldots, k$. For each $j = 1, \ldots, k$, partition the frame indices according to the j-synchronizations: let $F(m, j)$ consist of all frame indices of j-packets that arrive after $p^j_{i(m,j)}$ (inclusive) until the arrival of $p^j_{i(m+1,j)}$ (exclusive). Note that the partition corresponding to some j is in general different from the partition corresponding to j', but if the traffic is order-respecting, the partition corresponding to j is a refinement of the partition corresponding to $j' \ge j$ (this is a direct consequence of the definition of $F(m, j)$, along with the fact that $A_j \subseteq A_{j-1}$ for every j).

We now describe Algorithm SPA in detail. Intuitively, for every $j = 1, \ldots, k$, the algorithm alternates between accepting and rejecting states. During an "accept" state a contiguous block of $\frac{B}{k}$ j-packets is accepted, starting with a j-synchronization packet. The state then flips to "reject" state, where it stays long enough to allow delivery of all recently accepted j-packets. The transition between the states is governed by the arrival of the next j-synchronization packet. The basic argument of our analysis is that sufficiently many k-packets are delivered by SPA between any two consecutive k-synchronizations.

Pseudo code for SPA is given in Algorithm 1. The only delicate issue in the Algorithm's specification is the identification of j-synchronizing packets (lines 12–14). Recall that, $\mathrm{arr}(p^{j-1}_i) \le \mathrm{arr}(p^j_i)$ for all i. Identifying 1-synchronizing packets is easy, since it depends only on the termination of a reject phase. For $j \ge 2$ it can be shown by induction, that since $\mathrm{arr}(p^{j-1}_i) \le \mathrm{arr}(p^j_i)$ for all i, determining whether $i \in A_{j-1}$ is done before the algorithm is presented with p^j_i.

Note that since the traffic is order-respecting, once a frame index i has been identified as a $(j$–$1)$-synchronization (line 12), all $(j-1)$-synchronizations i' such that $i' \le i$ are no longer relevant. We do not explicitly remove these indices from A_{j-1} in the algorithm's specification merely for

Algorithm 1 SPA:

$m_j \leftarrow 1$ for all $j = 1, \ldots, k$ ▷ counter for j-synchronizations
$i(m_j, j) \leftarrow 1$ for all $j = 1, \ldots, k$ ▷ start index of a j-synchronization
$A_j \leftarrow \{1\}$ for all $j = 1, \ldots, k$ ▷ set of indices of j-synchronizations
$A_0 \leftarrow \emptyset$ ▷ set of indices of possible 1-synchronizations
for every packet p_i^j, in order of arrival **do**
  $A_0 \leftarrow A_0 \cup \{i\}$ ▷ makes sure every i is a candidate for a 1-synchronization
  if $i \leq i(m_j, j) + \frac{B}{k} - 1$ **then** ▷ p_i^j arrives during the accept phase
    accept p_i^j
  else if $\mathrm{arr}(p_i^j) < \mathrm{arr}(p^j_{i(m_j,j)+\frac{B}{k}-1}) + B$ **then** ▷ p_i^j arrives during the reject phase
    reject p_i^j
  else ▷ wait for a j-synchronization
    if $i \in A_{j-1}$ **then** ▷ p_i^j is a j-synchronizing packet
      $m_j \leftarrow m_j + 1$; $i(m_j, j) \leftarrow i$; $A_j \leftarrow A_j \cup \{i\}$
      accept p_i^j
    else ▷ p_i^j is not yet a j-synchronizing packet
      reject p_i^j
    end if
  end if
end for

the sake of brevity. The above implies that very little state information has to actually be maintained by the algorithm (in contrast to the greedy algorithm, presented in Section 3.4). We now turn to bound the competitive ratio of SPA.

Theorem 8. *Algorithm SPA is $(2k^2+k)$-competitive for the k-FTM problem.*

We start the analysis with the following lemma.

Lemma 9. *For every $j = 1, \ldots, k$ and for every $m = 1, \ldots, |A_j| - 1$, $\mathrm{arr}(p^j_{i(m+1,j)}) - \mathrm{arr}(p^j_{i(m,j)}) \geq B$.*

Proof. For $1 \leq j \leq k$, and $1 \leq m \leq |A_j|-2$, the claim follows immediately from the fact that j-synchronizations, by definition, are at least B time units apart. For $m = |A_j|-1$, the claim follows from the fact that $p^j_{\ell+1}$ is defined to arrive at time $\mathrm{arr}(p^j_\ell) + B \geq \mathrm{arr}(p^j_{i(m,j)}) + B$. □

Given $j' \geq j$, let $P_{j'}(m, j)$ denote the set of j'-packets corresponding to frames whose indices are in $F(m, j)$. In what follows, for every $j = 1, \ldots, k$ and every $m = 1, \ldots, |A_j| - 1$, we let $r(m, j) = \min\left\{\frac{B}{k}, |P_j(m, j)|\right\}$. The following lemma ensures that SPA accepts and delivers sufficiently many j-packets between any two consecutive j-synchronizations.

Lemma 10. *For every $j = 1, \ldots, k$ and for every $m = 1, \ldots, |A_j| - 1$, the j-buffer of SPA is empty before the arrival substep in time $\mathrm{arr}(p^j_{i(m,j)})$, all $r(m, j)$ packets $p^j_{i(m,j)}, \ldots, p^j_{i(m,j)+r(m,j)-1}$ are accepted by SPA, and they are all delivered by time $\mathrm{arr}(p^j_{i(m+1,j)})$.*

Proof. Let j be such that $1 \leq j \leq k$. The proof is by induction on m. For the base case consider $m = 1$, and note that by definition $i(m, j) = 1$. Clearly the j-buffer is empty before the arrival substep in time $\mathrm{arr}(p^j_1)$, and therefore all the $r(1, j)$ packets $p^j_1, \ldots, p^j_{r(1,j)}$ are accepted. Since SPA delivers packets according to FIFO order, no packet resides in the buffer more than B time steps, and therefore by Lemma 9 all these $r(1, j)$ packets are delivered by time $\mathrm{arr}(p^j_{i(2,j)})$.

For the induction step, assume the claim holds for the $(m-1)$'th j-synchronization, and consider the m'th j-synchronization. By the induction hypothesis all the j-packets $p^j_{i(m-1,j)}, \ldots, p^j_{i(m-1,j)+r(m-1,j)-1}$ are delivered by time $\mathrm{arr}(p^j_{i(m,j)})$. Since by definition SPA does not accept any j-packets out of $p^j_{i(m-1,j)+r(m,j)}, \ldots, p^j_{i(m,j)-1}$, the j-buffer is empty before the arrival substep in time $\mathrm{arr}(p^j_{i(m,j)})$. It follows that SPA has sufficient buffer space in its j-buffer to accept all $r(m, j)$ j-packets $p^j_{i(m,j)}, \ldots, p^j_{i(m,j)+r(m,j)-1}$. By the fact that the buffer delivers packets according to FIFO order and Lemma 9, all these packets are delivered by time $\mathrm{arr}(p^j_{i(m+1,j)})$, thus completing the proof. □

Lemma 10 implies the following lower bound on the number of frames delivered by SPA between two consecutive k-synchronizations.

Corollary 11. *SPA delivers at least $r(m, k)$ frames of indices in $F(m, k)$, for any m.*

Proof. Note that by the definition of synchronization, for any $i(m,k) \in A_k$ and every $j = 1, \ldots, k-1$, $p^j_{i(m,k)}$ is also a j-synchronizing packet. By Lemma 10 it follows that all the packets corresponding to frame indices in $\{i(m,k), \ldots, i(m,k) + r(m,k) - 1\}$ are delivered by SPA, as required. □

Next, we turn to consider the performance of an optimal policy, OPT, for a given input sequence. We first bound the number of j-packets any policy may accept between any two consecutive j-synchronizations.

Lemma 12. *For every $j = 1, \ldots, k$, and every $m = 1, \ldots, |A_j| - 1$, OPT can accept at most $j(\frac{B}{k} + 2B)$ j-packets out of $P_j(m,j)$.*

Proof. The proof is by induction on j. For the base case, assume $j = 1$, and consider any m such that $1 \leq m \leq |A_1| - 1$. By the definition of 1-synchronization packets, $p^1_{i(m+1,1)}$ is the earliest arriving 1-packet after time $\mathrm{arr}(p^1_{i(m,1)+r(m,1)-1}) + B$. Out of packets in $P_1(m,1)$, OPT can accept at most the set of $r(m,1)$ consecutive 1-packets accepted by SPA out of $P_1(m,1)$, and at most $2B$ additional packets arriving during the interval $[\mathrm{arr}(p^1_{i(m,1)+r(m,1)-1}), \mathrm{arr}(p^1_{i(m,1)+r(m,1)-1}) + B)$, since this is an interval of length B. It follows that the overall number of 1-packets OPT could have accepted out of $P_1(m,1)$ is at most $r(m,1) + 2B \leq \frac{B}{k} + 2B$.

For the induction step, assume the claim holds for $j-1$ and consider j. Let m be such that $1 \leq m \leq |A_j| - 1$. Similarly to the base case, OPT can accept the $r(m,j)$ consecutive packets accepted by SPA. Let i' be the minimal frame index of a j-packet such that $\mathrm{arr}(p^j_{i'}) \geq \mathrm{arr}(p^j_{i(m,j)+r(m,j)-1}) + B$. OPT can accept at most $2B$ j-packets arriving during the interval $[\mathrm{arr}(p^j_{i(m,j)+r(m,j)-1}), \mathrm{arr}(p^j_{i(m,j)+r(m,j)-1}) + B)$ since, again, this is an interval of length B. Next, consider the number of j-packets OPT can accept by the next j-synchronization. Given the above frame index i', since $\{F(t, j-1)\}_{t=1}^{|A_{j-1}|-1}$ is a partition of all frame indices, it follows that there exists some $1 \leq m' \leq |A_{j-1}| - 1$ such that $i' \in F(m', j-1)$. Furthermore, by definition there must be a j-synchronization corresponding to packet $i(m'+1, j-1)$. To see this, note that $i(m'+1, j-1) \geq i'$ (since i' is in $F(m', j-1)$), and $i(m'+1, j-1)$ is the frame index of the earliest arriving packet after $i(m', j-1)$ corresponding to a $(j-1)$-synchronization. This implies that $i(m+1, j) = i(m'+1, j-1)$.

Since any j-packet whose frame index i'' is in $\{i', i'+1, i(m+1,j)-1\}$ satisfies $i'' \in F(m', j-1)$, it follows that the overall number of j-packets OPT could have accepted out of $P_j(m', j-1)$ is bounded by the number of $(j-1)$-packets it could have accepted out of $P_{j-1}(m', j-1)$.[1] By the induction hypothesis this implies that OPT could have accepted at most $(j-1)(\frac{B}{k} + 2B)$ j-packets out of $P_j(m', j-1)$, and therefore at most $(j-1)(\frac{B}{k} + 2B)$ j-packets out of the j-packets whose frame index is in $\{i', i'+1, i(m+1,j)-1\}$.[2]

Combining the above bounds we conclude that OPT could have accepted at most

$$\frac{B}{k} + 2B + (j-1)\left(\frac{B}{k} + 2B\right) = j\left(\frac{B}{k} + 2B\right)$$

j-packets out of $P_j(m,j)$, as required. □

Lemma 12 immediately implies the following upper bound on the overall number of frames successfully delivered by any policy between any two consecutive k-synchronizations:

Corollary 13. *OPT accepts at most $2kB + B$ frames of indices in $F(m,k)$, for any m.*

The proof of Theorem 8 now follows from Corollaries 11 and 13 due to the fact that $\{F(m,k)\}_{m=1}^{|A_k|-1}$ is a partition of all frame indices.

3.4 The Greedy Algorithm

In this section we consider a natural online algorithm called G-ON. The basic intuition underlying this greedy algorithm is, in case of an overflow, to prefer keeping packets from frames for which it has already delivered many packets, and dropping packets from frames with fewer packets already delivered. This algorithm essentially tries to "cash-in" on effort already invested in delivering earlier packets of a frame. We show that while G-ON is effective for 2-FTM, it breaks down in the case of k-FTM with $k \geq 3$. In what follows we make use of the following notation with regards to G-ON:

- A packet p^j_i is said to be *relevant* if either $j = 1$, or $j = 2$ and p^1_i has not yet been dropped by G-ON.
- If G-ON sends p^1_i at time t, packet p^2_i is said to be *committed* as of time t.
- For a time step t and algorithm ALG, we let $A(t)$ denote the set of packets arriving at time t, and we let $\mathrm{Buff}_{\mathrm{ALG}}(t)$ denote the set of packets residing in the buffer of ALG at the beginning of time t.

[1] Note that we can assume without loss of generality that if OPT drops some packet p^j_i, then it also drops all packets $p^{j'}_i$, for all $j' > j$.

[2] This follows from the fact that since traffic is order-respecting, the partition of frame indices induced by the $j-1$ synchronization packets is a refinement of the partition induced by the j synchronization packets.

- For a time step t and algorithm ALG, we let $L_{\mathrm{ALG}}(t) \subseteq A(t) \cup \mathrm{Buff}_{\mathrm{ALG}}(t)$ denote the set of all relevant packets in $A(t) \cup \mathrm{Buff}_{\mathrm{ALG}}(t)$.

We now specify the algorithm more formally for the case of 2-FTM. The greedy algorithm G-ON accepts all relevant packets, except for the case of an overflow, in which it keeps packets according to the following strict priority (ties are broken according to time of arrival, by preferring earlier arriving packets):

1. First, 2-packets whose 1-packets were delivered are taken
2. If there is additional room, complete frames (both their 1-packet and 2-packet) are taken.
3. Finally, remaining 1-packets fill the leftover space if any.

Note that Algorithm G-ON is preemptive: a packet that has entered the buffer may be dropped later.

The following lemma shows that once a 2-packet is accepted by G-ON, it is never dropped in subsequent time steps.

Lemma 14. *When considering 2-FTM, G-ON never drops a previously-accepted 2-packet.*

Proof. Let p_j^2 be any packet in $\mathrm{Buff}_{\text{G-ON}}(t)$. By the assumption that the traffic is order-respecting, any 2-packet p_ℓ^2 arriving at time t must satisfy $\ell > j$, and if any such packet is committed upon arrival at t, then p_j^2 must also be committed at t. It therefore follows that in scanning either committed packets, or complete frames, p_j^2 is always preferred to any such p_ℓ^2. In addition, clearly p_j^2 is always preferred to any 1-packet, which completes the proof. □

We now turn to prove our main result in this section as to the performance of Algorithm G-ON.

Theorem 15. *When considering 2-FTM, the online algorithm G-ON is* $(11 + \frac{8}{B-1})$*-competitive.*

Proof. Consider any frame $f_i \in \mathrm{OPT} \setminus \text{G-ON}$. This can happen because one of the following:

1. $p_i^1 \in$ G-ON, and $p_i^2 \notin$ G-ON:

   For every such frame f_i, by Lemma 14, p_i^2 is dropped upon arrival due to overflow, such that at time $\mathrm{arr}(p_i^2)$ G-ON's buffer holds at least $\left\lfloor \frac{B}{2} \right\rfloor \geq \frac{B-1}{2} = \frac{B}{2}\left(1 - \frac{1}{B}\right)$ 2-packets. To see this, note that a 2-packet is only dropped when at least $B-1$ slots in the buffer are dedicated to committed 2-packets, and complete frames. We henceforth ignore the floor notation, which will later be accounted for by the above factor of $\left(1 - \frac{1}{B}\right)$.

   Let $t_i = \mathrm{arr}(p_i^2)$ and consider the intervals $I_i = (t_i, t_i + B]$ corresponding to frames f_i which adhere to the case under consideration. Let R be any maximal set of such intervals such that $I(R) = \cup R$ is a continuous interval, and WLOG assume $R = \{I_1, \ldots, I_\ell\}$.

   We now show that during any such interval $I(R)$, G-ON delivers at least $\frac{I(R)}{2}$ 2-packets.

   For every $i = 2, \ldots, \ell$, denote by m_i the number of packets in $\mathrm{Buff}_{\text{G-ON}}(t_{i-1}) \cap \mathrm{Buff}_{\text{G-ON}}\, G(t_i)$, and let $m_1 = 0$.

   By the definition of m_i, for every $i = 2, \ldots, \ell$, the number of new packets accepted by greedy during interval $[t_{i-1}, t_i)$ is exactly $B - m_i$. Since there is an overflow causing the dropping of a 2-packet at time t_i, it follows that at least $\frac{B-m_i}{2}$ of these packets must be 2-packets. Since by the end of $I(R)$ all these 2-packets have already been delivered by G-ON, and since at time t_1 the buffer contains at least $\frac{B}{2}$ 2-packets, it follows that the number of 2-packets delivered by G-ON during $I(R)$ is at least

   $$\frac{B}{2} + \sum_{i=2}^{\ell} \frac{B - m_i}{2} = \frac{1}{2} \sum_{i=1}^{\ell} B - m_i.$$

   Furthermore, since G-ON is never idle during $I(R)$ (by the definition of R), and $t_i - t_{i-1} \leq B$, we have

   $$|I(R)| = B + \sum_{i=2}^{\ell} B - m_i = \sum_{i=1}^{\ell} B - m_i,$$

   which implies that G-ON delivers at least $\frac{|I(R)|}{2}$ 2-packets during $I(R)$.

   When considering the number of 2-packets which can be accepted by OPT during $I(R)$, note that this number is bounded by $|I(R)| + B$, implying that the ratio between the number of 2-packets delivered by G-ON, and the number of 2-packets of frames corresponding to the case under consideration, is no more than

   $$\frac{\frac{|I(R)|}{2}}{|I(R)| + B} \leq \frac{1}{4},$$

   where the inequality follows from the fact that $|I(R)| \geq B$.

   It therefore follows that we can map the 2-packets accepted by OPT during any such interval to a distinct set of packets delivered by G-ON during such a block, such that every such 2-packet in G-ON is mapped to by at most four 2-packets in OPT. Specifically we do not map more than 4 distinct frames corresponding to the case under consideration, to any single frame delivered by G-ON.

2. $p_i^1 \notin$ G-ON:

   p_i^1 is not delivered by greedy because it was dropped due to overflow at some time $t \geq \mathrm{arr}(p_i^1)$. It must follow that at time t, G-ON's buffer is full, and there exists some packet $p \in$ G-ON such that $s_{\text{G-ON}}(p) = s_{\mathrm{OPT}}(p_i^1)$. In what follows we distinguish between several cases.

   (a) $p = p_j^2$ is a 2-packet:
   This particularly means that $f_j \in$ G-ON, in which case we map f_i to f_j.

   (b) $p = p_j^1$ is a 1-packet, and $p_j^2 \in$ G-ON:
   In this case we map f_i to f_j.

   (c) $p = p_j^1$ is a 1-packet, and $p_j^2 \notin$ G-ON:
   When restricting our attention to frames f_i corresponding to this case, we first note that the indexing mapping $j = j(i)$, implied by the definition of p, is a bijection.

   In this case it must follows that at time $\mathrm{arr}(p_j^2)$, G-ON's buffer contains at least $\lfloor \frac{B}{2} \rfloor$ 2-packets.

   By the order in which G-ON scans the packets, we are guaranteed to have $\mathrm{arr}(p_j^1) \leq \mathrm{arr}(p_i^1)$. To see this assume the contrary. If $t \geq \mathrm{arr}(p_j^1)$, then by the assumption that traffic is order-respecting, G-ON would have preferred dropping p_j^1 before resorting to dropping p_i^1. It therefore follows that $\mathrm{arr}(p_j^1) > t$. Since at time t G-ON's buffer is full, and G-ON employs a FIFO discipline, it must follow that $s_{\text{G-ON}}(p_j^1) > t + B \geq \mathrm{arr}(p_i^1) + B$. On the other hand, OPT also follows a FIFO discipline, implying that $s_{\mathrm{OPT}}(p_i^1) \leq \mathrm{arr}(p_i^1) + B$, contradicting our assumption that $s_{\text{G-ON}}(p_j^1) = s_{\mathrm{OPT}}(p_i^1)$.

   Note that $s_{\mathrm{OPT}}(p_i^1) > \mathrm{arr}(p_i^1)$ and $s_{\text{G-ON}}(p_j^1) \leq \mathrm{arr}(p_j^1) + B$. Since $s_{\text{G-ON}}(p_j^1) = s_{\mathrm{OPT}}(p_i^1)$ we are guaranteed to have $\mathrm{arr}(p_i^1) < \mathrm{arr}(p_j^1) + B$.

   Consider the interval $[\mathrm{arr}(p_j^1), \mathrm{arr}(p_i^1)]$. By the above argument we are guaranteed to have that during this interval, OPT can accept at most $\mathrm{arr}(p_i^1) - \mathrm{arr}(p_j^1) + B \leq 2B$ 1-packets which are not accepted by G-ON.

   Consider the interval $[\mathrm{arr}(p_j^2), \mathrm{arr}(p_i^2)]$. Although this interval may be longer than B, by the assumption that traffic is order-respecting, assumption we are guaranteed that during this interval, OPT can accept at most $2B$ 2-packets that are not accepted by G-ON.

   Let $t_i = \mathrm{arr}(p_{j(i)}^2)$ and consider the intervals $I_i = (t_i, t_i + B]$ corresponding to frames f_i which adhere to the case under consideration. We can now employ the same argument as the one appearing in case (1).

   It therefore follows that we can map every frame f_i corresponding to the case under consideration to a frame successfully delivered by G-ON, such that every such packet delivered by G-ON is mapped to by at most four such frames in OPT.

By reaccounting for the $(1 - \frac{1}{B}) = (1 + \frac{1}{B-1})^{-1}$ factor emanating from cases (1) and (2c), and summing over all possibilities, it follows that the number of frames in OPT $\setminus$ G-ON that are mapped to a single frame $f_j \in$ G-ON is at most

$$4\left(1 + \frac{1}{B-1}\right) + 1 + 1 + 4\left(1 + \frac{1}{B-1}\right) = 10 + \frac{8}{B-1},$$

where the summands correspond to cases (1), (2a), (2b), and (2c), respectively. □

The greedy algorithm naturally extends to the k-packet frame case: in case of overflow, prefer taking packets corresponding to frames for which the most packets have already been delivered. This intuitive algorithm tries to "cash in" the work that has been invested in delivering previous packets. Perhaps surprisingly, this greedy algorithm fails to be competitive even for 3-FTM, as the following lemma implies.

Lemma 16. *The natural extension to G-ON has no bounded competitive ratio for frames consisting of 3 packets, even for order-respecting input sequences.*

Proof. Consider the following packet arrivals, consisting of 2 batches of frames:

- $t = 0, \ldots, T-1$ (for an arbitrary large value of T):
  a sequence of 1-packets arriving, one in every time step. These are the 1-packets of the first batch of frames.
- $t = T$:
  B 2-packets corresponding to the first sequence, followed by the remaining $T - B$ 2-packets, arriving one in every time step. These are the 2-packets of the first batch of frames. This sequence terminates at time $2T - B$.
- $t = T, \ldots, 2T - B$:
  a second sequence of 1-packets arriving, one in every time step. These are the 1-packets of the second batch of frames
- $t = 2T - B + 1$:
  T 3-packets corresponding to the first sequence. These are the 3-packets of the first batch of frames

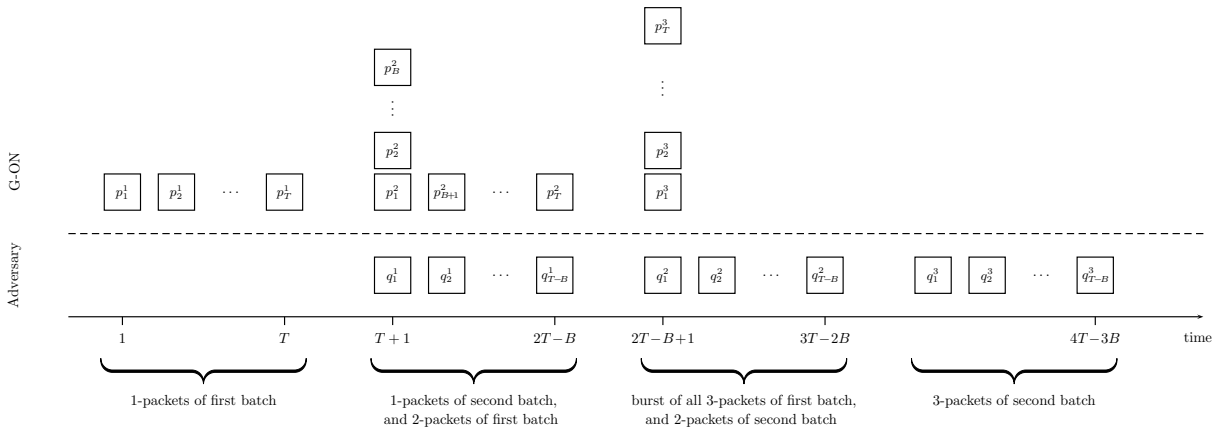

Figure 4. Outline of adversary described in Lemma 16 yielding an unbounded competitive ratio for the greedy algorithm. Each square marked by p_j^i (q_j^i) represents the arrival of the i-packet corresponding to the j'th frame of G-ON (of the adversary).

- $t = 2T - B + 1, \ldots$:
  the remaining 2-packets and then 3-packets corresponding the second batch of frames, arriving one in every time step.

Figure 4 gives an outline of the above arrival sequence. It is not hard to see that G-ON would drop all 1-packets corresponding to the second batch in favor of accepting all 2-packets corresponding to the first batch, and will eventually be able to accept only B of the 3-packets corresponding to the first batch, thus completing the delivery of only B frames. The adversary, on the other hand, would focus solely on the second batch, and can deliver all frames in that batch, for a total of $T - B$ frames. Since T may be arbitrarily large, this implies that G-ON cannot have a bounded competitive ratio for the case where frames consist of 3 packets each. □

4 k-of-n FTM

In this section we consider some special cases of the k-of-n FTM problem. We provide only statements here. Proofs are omitted due to space constraints.

It turns out that the k-FTM problem can be as hard as the k-of-n FTM in the following sense.

Theorem 17. *If there is an offline polynomial time algorithm solving $(k+1)$-FTM for $B = 1$ with approximation ratio α, then there is a polynomial time algorithm solving k-of-n FTM with approximation ratio α for the case where $B = 1$ and either k or $n - k$ is constant.*

For the special case of 1-of-n FTM, we have the following results.

Theorem 18. *The 1-of-n FTM problem can be solved optimally off-line in polynomial time.*

Theorem 19. *The online greedy algorithm for the 1-of-n FTM problem is 2-competitive.*

5 Conclusions and Open Questions

Motivated by the scenarios where large data frames are fragmented into multiple packets, we present in this paper a new model for managing buffer overflows, where arriving packets have dependencies among them. We present several lower bounds and provide algorithms for both the online and offline cases. In the online setting, we analyzed the performance of our algorithms by means of competitive analysis. Our results provide initial insight into the new buffer management problem with packet dependencies.

There are many interesting problems that we leave open, and many ways to extend the model. Here are a few obvious open problems.

- What is the true competitiveness of online k-FTM for order-respecting inputs? We conjecture it is $\Theta(k)$ (but our upper bound is $O(k^2)$).
- What is the complexity of off-line 2-FTM when $B > 1$? We know that it is polynomial when $B = 1$ and NP-hard when $k > 2$.
- What is the online competitiveness of order-respecting k-of-n FTM for general n and k?
- Our results focused on deterministic settings. Does randomization help in tackling the problem of buffer management with dependencies? If so, to what extent?

Regarding extensions of the model, here are a few possible directions.

- This paper defined a 2-level model of packets and frames. It seems natural to ask about general hierarchies. (This occurs frequently in practice: For example, MPEG video frames are broken in *slices* and aggregated in *Groups of Pictures*.)

- Study conditions under which competitive algorithms are possible: perhaps there is a better alternative to our notion of "order-respecting arrivals."

- This work focused on the problem of inter packet dependencies where packets can be dropped in a single bottleneck buffer: it is natural to extend it to a general network setting, where multiple buffers on the packets path can overflow.

Acknowledgment

Research supported in part by the Next Generation Video (NeGeV) consortium, Israel. This work was done while the third author was with Tel Aviv University.

References

[1] W. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosén. Competitive queue policies for differentiated services. *J. Algorithms*, 55(2):113–141, 2005.

[2] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Transactions on Information Theory*, 42(6):1737–1744, 1996.

[3] N. Andelman. Randomized queue management for diffserv. In *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10. ACM Press, 2005.

[4] N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for qos switches. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–770, 2003.

[5] S. Angelov, S. Khanna, and K. Kunal. The network as a storage device: Dynamic routing with bounded buffers. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–13, 2005.

[6] Y. Azar and Y. Richter. The zero-one principle for switching networks. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 64–71, 2004.

[7] N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for qos buffering. In *Proceedings of the 31st International Colloquium on Languages and Programming (ICALP)*, pages 196–207, 2004.

[8] J.-C. Bolot, S. Fosse-Parisis, and D. F. Towsley. Adaptive FEC-based error control for internet telephony. In *Proceedings of IEEE INFOCOM 1999, the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1453–1460, 1999.

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

[10] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.

[11] F. Y. Chin, M. Chrobak, S. P. Fung, W. Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. Discrete Algorithms*, 4:255–276, 2006.

[12] M. Englert and M. Westermann. Lower and upper bounds on fifo buffer management in qos switches. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 352–363, 2006.

[13] E. Hazan, S. Safra, and O. Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006.

[14] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM J. Comput.*, 33(3):563–583, 2004.

[15] J.-H. Kim. Optimal buffer management via resource augmentation. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 618–628, 2004.

[16] L. Kleinrock. *Queueing Systems*. Wiley-Interscience, 1975.

[17] Y. Mansour, B. Patt-Shamir, and O. Lapid. Optimal smoothing schedules for real-time streams. *Distributed Computing*, 17(1):77–89, 2004.

[18] A. E. Mohr, E. A. Riskin, and R. E. Ladner. Unequal loss protection: Graceful degradation over packet erasure channels through forward error correction. *IEEE J. Selected Areas in Communication*, 18(7):819–828, 2000.

[19] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction. In *Proceedings of the 12th International Packet Video Workshop (PV)*, 2002.

[20] A. Rosén and G. Scalosub. Rate vs. buffer size: Greedy information gathering on the line. In *Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 305–314, 2007.

[21] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.