Buffer Management for Aggregated Streaming Data with Packet Dependencies

Gabriel Scalosub, Peter Marbach, and Jörg Liebeherr

Abstract—In many applications, the traffic traversing the network has inter-packet dependencies due to application-level encoding schemes. For some applications, e.g., multimedia streaming, dropping a single packet may render useless the delivery of a whole sequence. In such environments, the algorithm used to decide which packet to drop in case of buffer overflows must be carefully designed, to avoid goodput degradation.

We present a model that captures such inter-packet dependencies, and design algorithms for performing packet discard. Traffic consists of an aggregation of multiple streams, each of which consists of a sequence of inter-dependent packets. We provide two guidelines for designing buffer management algorithms for this problem, and demonstrate their effectiveness. We devise an algorithm according to these guidelines and evaluate its performance analytically, using competitive analysis. We also perform a simulation study that shows that the performance of our algorithm is within a small fraction of the performance of the best offline algorithm.

Index Terms—Buffer Management, Competitive Analysis, FIFO, Switch and Router Architecture, Quality of Service

1 Introduction

In the vast majority of networked applications, application-layer data frames are split into several smaller sized packets, before transmission across the network. The receiving side can make use of the data only if it receives *all* (or at least sufficiently many) packets of a frame. The above implies that such a data stream has an inter-packet dependency structure. Most current best-effort networks, such as the Internet, are oblivious to these dependencies, and usually make decisions on a per-packet base. Higher-level mechanisms in the protocol stack usually handle retransmissions of lost packets. The exact dependency structure of the data stream depends on the encoding used and it may consist of a 1-level dependency structure, where frames are independent of each other, and the only dependencies are among packets corresponding to the same frame. Such an encoding may also have a higher-level dependency structure, such as the one occurring in MPEG video encoding schemes, where successfully decoding a frame might depend on successfully decoding previous or later frames. The problem of ensuring that all packets of a frame arrive at the destination is crucial when one considers real-time traffic, such as *streaming multimedia* traffic, where retransmission of missing packets is not feasible due to delay constraints posed by the application. Such services over packet switched networks have gained a huge increase in popularity in recent years. For example, subscriptions to live broadcasts over IPTV are expected to double and reach some 120M subscribers worldwide by 2013, according to a recent survey [1]. This trend motivates efforts for providing better algorithmic solutions to ameliorate network performance in such scenarios.

A common approach to deal with the hazardous effect of packet loss is to employ encoding schemes, which have long been known to provide substantial improvement in performance. However, this approach has its limitations in some networking environments. Specifically, some environments (e.g., wireless networks) make the usage of such approaches prohibitively costly, due to the increased traffic load. In scenarios, where traffic traverses bottleneck links, the additional traffic load due to encoding may trigger packet losses, thus, diminishing the benefits of proactive coding. Canonical examples of such bottlenecks include head-ends in cable content distribution networks, where downstream traffic arrives from the backbone before it is distributed to end users via an access network, wireless gateways, which multiplex traffic to a multi-hop wireless network, and network transcoders, that perform conversions of encoded data streams in real-time. We therefore believe that the availability of encoding schemes alone does not replace the need to design and analyze algorithms that aim to optimize the usage of available resources.

In this work we focus on a FIFO buffer architecture, which has several appealing features: (a) it is simple, (b) it maintains the arrival order of incoming traffic, hence avoiding the need for mechanisms that deal with packet reordering, and (c) it provides simple and reliable delay bounds. These properties make FIFO especially attractive for delay- and reordering-constrained streaming environments.

- *G. Scalosub is with the Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva 84105 Israel. E-mail:* sgabriel@bgu.ac.il. *P. Marbach is with the Department of Computer Science, University of Toronto, Toronto, ON, Canada. E-mail:* marbach@cs.toronto.edu. *J. Liebeherr is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. E-mail:* jorg@comm.utoronto.ca.
- *A preliminary version of this work appeared as a short paper in the Proceedings of IEEE INFOCOM 2010 Mini-Conference.*

The main causes for packet loss in networks are buffer overflows due to congestion. In cases where the underlying traffic has inter-packet dependencies, indiscriminately dropping packets upon overflow may result in very poor performance. For data streams with packet dependencies, we must differentiate between the *packet-level throughput*, i.e., the amount of data delivered in terms of packets, and the effective *goodput*, i.e., the amount of data that can be decoded effectively at the receiving end. These two measures can be drastically different, e.g., the throughput may be high, while its goodput is very low. As an extreme example of such a scenario consider the case where one packet is dropped from every frame, which results in zero goodput, although overall packet-level throughput might be high [2]. The method to decide which packets to drop in case of overflow is critically important to the performance of the system, bearing in mind that such a decision might effect other packets which have already been forwarded, or packets that have not yet arrived. The goal of this paper is to devise methods that maximize the goodput of successfully delivered traffic, as captured by the number of useful *complete* frames delivered.

In this work we consider the problem of buffer management of multiple data streams in scenarios where traffic has inter-packet dependencies. We provide guidelines for designing algorithms that are guaranteed to provide high performance in terms of goodput. Our approach and analysis provide bounds on the performance of the proposed algorithms for *any* traffic arrival pattern, without requiring any stochastic or deterministic assumptions on the processes generating the traffic. In effect, we commonly consider the traffic to be *adversarial*. Different from works which focused on estimating system performance using either statistical or deterministic traffic models (e.g., [3], [4]), we study packet discard policies in a much more fine-grain sense – in fact we study how "micro" decisions affect system performance. In this sense, our approach is orthogonal to works that aim at exploiting statistical multiplexing, or those that try to analyze the tradeoff between available network resources (e.g., in terms of the buffer size available) and system performance.

We restrict our attention to traffic where dependencies are restricted to the intra-frame domain. A better understanding of the algorithmic problems in this setting is essential before tackling more complex dependency structures, such as inter-frame dependencies. Our model is general enough to be applicable to various traffic encoding schemes, and can potentially be combined with buffer management schemes that deal with higher-level dependencies. Our approach is especially suitable for best-effort environments, where oversubscription of resources is common, and the system goal is to optimize the use of resources in an overloaded setting.

1.1 Our Contribution

We seek to gain a better understanding of designing buffer management algorithms for traffic with inter-packet dependencies. Our main contributions are as follows:

1) We provide two design guidelines for algorithms, namely, (a) *No-regret* – Once a frame has a packet *admitted* to the buffer, one should make every attempt possible to deliver the complete frame; and (b) *Ensure-progress* – Strive to deliver a complete frame as soon as possible.
2) We devise a buffer management algorithm, WEIGHT-PRIORITY, that follows these guidelines. We analyze the performance of our algorithm, and show that for *any* arrival traffic the ratio between its performance and that of an optimal algorithm is always bounded. We further prove lower bounds on the performance of any buffer management algorithm.
3) We conduct a simulation study that further evaluates the performance of algorithms which follow our design criteria. We show that algorithms that follow our guidelines have a stable high performance, close to that of the best offline algorithm. At the same time, algorithms which do not adhere to our criteria exhibit a fast degradation in performance as traffic characteristics become harder.

1.2 Previous Work

Various aspects of providing traffic with Quality-of-Service (QoS) guarantees have been studied extensively. The works most related to our problem of packet forwarding with inter-packet dependencies are set in the context of video traffic. Most of these works consider specific encoding schemes (e.g., MPEG), and higher-level inter-frame dependencies. These works generally focus on aspects of network provisioning in terms of buffer size, bandwidth, etc., and stream admission, but mostly side-step the algorithmic question addressed in this study, namely: "which packet should be dropped when overflow occurs?". Attempts to answer this question were made at the *frame* level, by trying to decide which frame to drop for specific encoding schemes so as to have as little effect as possible on the received data stream [5]–[7]. However, discard decisions at routers are made at the *packet* level, and in cases where multiple streams contend for network resources, making the right decision is essential.

Ramanathan et al. [8] propose a buffer management policy that takes packet dependencies into account, namely, that in any case where too many packets are dropped, one drops the entire frame. As a rule for dropping packets, they suggest a latest-frame-first rule. This rule states that the frame to be dropped is that for which the least number of packets have been delivered. Ramanathan et al. then evaluate their scheme assuming Markovian video sources. In our study, we make no assumptions on the statistical nature of the streams. Our work considers arbitrary arrival patterns for each stream, and analyzes the system performance from a worst-case perspective. This enables uncovering the difficulties faced by any buffer management algorithm, and also provide the first analytic guarantees as to the performance of such algorithms where no assumptions are made on the traffic.

A substantial body of work considered priority-based frame discard policies, for inter-frame dependencies occurring in encoding schemes, either at the edge or in the core of the network [9], [10]. The common aspect of these approaches is discriminating between different frames according to their importance, also referred to as multi-frame impact. For example, for MPEG encoded video streams, dropping a B-frame has a mere "local" effect in terms of performance degradation, whereas dropping an I-frame might render subsequent P- and B-frames useless. As mentioned we do not consider these higher-level dependencies (or inter-frame dependencies). Instead our main concern is to obtain a better understanding of low-level inter-packet dependencies, and devise methods for dealing with them. Our approach can be integrated with any previously devised high-level priority-based policies.

Our model is close to that considered by Kesselman et al. [11], which presented several results for a general case where no stream-structure is underlying the arrival traffic (i.e., every frame is independent of all other frames). They provide a lower bound which states that no algorithm can have a bounded competitive-ratio in the general case. They further consider restricted arrival patterns where packets corresponding to different streams satisfy some order requirements. These assumptions are not general enough to model any arrival patterns encountered in real-life networks. However, they do provide worst-case guarantees for such restricted arrival patterns. Our model is less restrictive than the model considered in [11], and captures the structure of real-life data streams. Another recent related work [12] touches on the problems of set packing and task scheduling. This can be viewed as a special case of the model considered by [11], assuming a unit size buffer and limited adversaries. Several other works that deal with competitive algorithms which aim to provide different measures of QoS to the traffic have been studied (e.g., [13], [14]), but none of them addresses the issues of inter-packet dependencies. We refer to [15] for a recent survey.

Another networking environment which considered dependencies between transmitted data chunks is the ATM guaranteed frame rate service (see [16]). Research on this service mainly focused on the tradeoff between network provisioning and throughput, and similar to the work done in the context of video streams, mostly considered frames as fundamental entities which should be accepted/dropped. This body of research did not tackle the question of what cell-level decision should be made. Some work in this domain studied the interaction between cell-level dropping policies and TCP traffic [17], as well as the loss of MPEG video frames over ATM networks [18]. This line of research differs significantly from ours since we try to maximize the goodput given the *available* resources.

Much work has been done on proactive encoding schemes, commonly known as forward error correction or FEC, for packetized traffic, focusing on implementation and information-theoretic aspects of such schemes (e.g., [19], [20]). Our work is orthogonal to these efforts since encoding schemes aim at providing guarantees on the amount of packet loss a frame can incur, while still allowing the receiving end to decode it properly. Such schemes are usually oblivious to the specific buffer management algorithms employed in the network that are the cause of packet losses. Our work emphasizes the design of buffer management algorithms with *provable* properties, and on gaining insights into discard-decisions of packets. Although our model assumes no redundancy in the data (i.e., losing a single packet of a frame renders the entire frame useless), it can be a starting point for designing algorithms accounting for proactive coding. One should note that employing FEC is not without cost. It increases the traffic volume, without increasing the application-level data rate. Such an approach may be prohibitive in bandwidth-restricted environments, such as wireless networks, where increasing traffic volume might have a disastrous effect on the overall system performance, e.g., due to increased interference and collisions (see [21]–[23]). In such environments streaming is preferred with little (or no) FEC.

2 Model

2.1 Traffic model

We consider an aggregation of M streams of unit-sized packets, denoted by $S_1, \ldots, S_M$. Each stream S_m is viewed as a sequence of *frames*, f_i^m, each consisting of a sequence of exactly k *packets*, $p_1^{m,i}, \ldots, p_k^{m,i}$. A packet $p_j^{m,i}$ is referred to as the *j-packet* of frame f_i^m, and its arrival time is denoted by $a(p_j^{m,i})$. When referring to packets, we will sometimes omit the frame index i, and use the notation $\left\{p_j^m\right\}_j$ when referring to the sequence of packets corresponding to stream S_m, where p_j^m denotes the j'th packet of stream S_m, and the $(j \mod k)$-packet of frame $f^m_{\lfloor \frac{j}{k} \rfloor}$ (i.e., the $\lfloor \frac{j}{k} \rfloor$'th frame of stream S_m). The packets of a stream arrive in order, i.e., $a(p_j^m) \le a(p_{j+1}^m)$ for all j. The above notation implies the following structure on the arrival of packets in a stream S_m consisting of r_m frames:

$$\underbrace{p_0^m, \ldots, p_{k-1}^m}_{\text{frame } f_0^m}, \underbrace{p_k^m, \ldots, p_{2k-1}^m}_{\text{frame } f_1^m}, \ldots, \underbrace{p_{(r_m-1)k}^m, \ldots, p_{r_m k-1}^m}_{\text{frame } f_{r_m-1}^m}.$$

The arrival of packets from different streams implies a finite arrival sequence σ of the aggregated streams, which is the interleaving of the arrival sequences of the individual streams. We make no assumptions on the processes generating the arrival sequences. Figure 1 provides an example of an interleaved arrival sequence ($M = k = 2$). The example shows the grouping of packets into frames, where the grouping of frames corresponding to stream 1 appears above the arrival sequence, and the grouping of frames corresponding to stream 2 appears below the arrival sequence.

2.2 Buffer model

The packets arrive at a FIFO buffer which can store up to $B \ge k$ packets and which can transmit one packet per cycle. Initially, the buffer is empty. Each cycle consists of two steps. The first step is the *delivery step*: if the buffer is

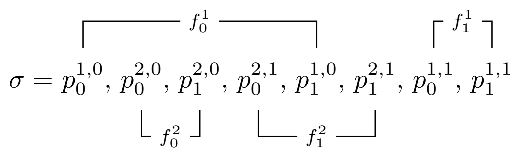
f_0^1
f_1^1
$\sigma = p_0^{1,0}, p_0^{2,0}, p_1^{2,0}, p_0^{2,1}, p_1^{1,0}, p_1^{2,1}, p_0^{1,1}, p_1^{1,1}$
f_0^2
f_1^2

Fig. 1. Example of an interleaved arrival sequence.

non-empty, the head-of-the-line packet is transmitted on the link. In the second step, called the *arrival step*, an arbitrary set of packets arrives at the buffer. At the discretion of the buffer management algorithm, some packets may be dropped, while other packets are stored in the buffer. Note that a buffer management algorithm may drop packets even if there is space available at the buffer.

2.3 Performance measure

We say a frame is *successfully delivered* by a buffer management algorithm ALG if ALG delivers all packets belonging to the frame. We define the MAX-FRAME-GOODPUT problem, denoted MFG, as the problem to devise a buffer management algorithm which maximizes the number of successfully delivered frames. This work studies online algorithms for solving the MFG problem. An *online algorithm* is an algorithm that, at any point in time, knows of arrivals that have occurred up to that time but has no information about future arrivals. This should be contrasted with offline algorithms, which are clairvoyant and know the entire input in advance.

We use competitive analysis [24], [25] to evaluate the performance of online algorithms. An algorithm ALG is said to be *c-competitive* if for any traffic arrival sequence σ, the maximum 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 $c \geq 1$. The value c is referred to as the *competitive ratio* of algorithm ALG. As is customary in competitive analysis, we view the online algorithm as competing against an offline *adversary* that generates the input stream, and provides an optimal schedule for that input. Given an algorithm ALG, we will sometimes abuse notation and refer to ALG also as the set of packets or frames delivered by ALG.

3 LOWER BOUNDS ON THE COMPETITIVE RATIO

In this section we show that one of the predominant elements affecting the competitive ratio of any buffer management algorithm is the number of streams M traversing the node. To show this, we describe two types of adversaries, depending on the ratio M/B between the number of streams and the buffer size. One adversary (described in Lemma 1) is targeted at large values of this ratio, whereas the other adversary (described in Lemma 2) is targeted at small values of this ratio. Each type of adversary is designed to cause any algorithm to drop a substantial number of packets, which implies dropping a substantial number of frames. Some of the dropped packets serve as "markers" to identify frames that will be accepted by the optimal solution. These packets are sometimes referred to as the *packets accepted by the adversary*. Subsequently, we will combine these two adversaries in proving a single lower that has the strongest guarantee of the two adversaries, while removing the dependency on the ratio M/B.

The following lemma provides a lower bound that holds for a large number of streams. The proof of this lemma draws from techniques developed in [11].

Lemma 1. *Any deterministic algorithm for* MFG *with* $M \geq 4B$ *streams has competitive ratio* $\Omega(\frac{M}{B})$*, even for* $k = 2$.

Proof: Let ALG be any deterministic algorithm. Consider the arrival of the first frame of each stream. Consider an adversary that first injects $M/2B$ bursts of $2B$ 1-packets, once every B time steps, starting from time $t = 0$. For every such burst, ALG may accept at most B 1-packets. It follows that there remain B 1-packets dropped by ALG, to which we refer as the adversary's packets. At time $t = M/2$ we have a burst of all $M/2$ 2-packets corresponding to the 1-packets accepted by ALG. Since ALG can accept at most B of them, it follows that ALG can deliver at most B complete frames. The 2-packets corresponding to the adversary's packets arrive one in every time step, starting at time $t + 1$, such that the adversary can deliver them all by time $t + M/2 + 1$. It follows that the ratio between the number of packets delivered by ALG and the number of packets delivered by the adversary is at least $M/2B$. Since we can repeat this arrival pattern arbitrarily many times, also for further stream frames, the result follows. □

The following lemma deals with the case where the number of streams is small.

Lemma 2. *Any deterministic algorithm for* MFG *with* $M \leq 4B$ *streams has competitive ratio* $\Omega(\frac{kM}{B})$.

Proof: We will prove the lemma for an adversary that uses merely $M \leq B$ streams. First, consider the arrival of the first frame of each stream. We identify stream S_i with the first frame f_i in that stream. Let $\ell = \frac{B}{M}$, and assume for simplicity that ℓ and $\log(\frac{k}{\ell}+2)$ are integers. We further let

$$t_i = \ell \sum_{s=1}^{i-1} 2^s + 1 = (2^i - 2)\ell.$$

Consider the following arrival sequence, which consists of $\log(\frac{k}{\ell}+2)$ blocks, $R_1, \ldots, R_{\log(\frac{k}{\ell}+2)}$, which are defined as follows: At the beginning of block R_1, we have the arrival of all M j-packets corresponding to all frames, for $j = 1, \ldots, 2\ell$. This is a total of $2\ell M = 2B$ packets. Since any algorithm can only hold B packets in its buffer, any algorithm must drop at least B of these packets. This implies that any algorithm must forfeit at least $M/2$ frames (and can also ensure no more than that, by dropping all packets corresponding to a forfeited frame). We refer to the frames corresponding to $M/2$ such frames dropped by the algorithm as the *adversary's frames* and to the remaining $M/2$ frames as the *algorithm's frames*.

For every $i = 2, \dots, \log(\frac{k}{\ell} + 2)$, block R_i consists of the arrival of all j-packets, for $j = t_i, \dots, t_{i+1} - 1$, corresponding to all frames. At the beginning of R_i we have the simultaneous arrival of all the packets corresponding to the algorithm's frames. After this burst, the j-packets of the adversary's frames arrive, one in each cycle, respecting the stream order.

The frames that have not yet been dropped by the algorithm at the end of a block R_i are called *live frames*. We claim, by induction on the block number, that the number of live frames at the end of R_i is at most $\frac{M}{2^i}$.

From the discussion above, 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 $M/2^{i-1}$ live frames. By the definition of block R_i, all j-packets, for $j = t_i, \dots, t_{i+1} - 1$, corresponding to these live frames, arrive together at the beginning of block R_i. This yields a total of $t_{i+1} - t_i = 2^i M\ell = 2^i B$ packets arriving simultaneously. Since a fraction of $1/2^{i-1}$ of the frames are live frames, this implies that $2^i B/2^{i-1} = 2B$ of the packets arriving in block R_i correspond 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 least half the live frames must be forfeited by the algorithm, leaving at most $M/2^i$ live frames that can survive at the end of block R_i, out of the $M/2^{i-1}$ live frames at the end of the previous block, as required.

It follows that at the end of block $R_{\log(\frac{k}{\ell}+2)}$, the algorithm can maintain at most $M/2^{\log(\frac{k}{\ell}+2)} = \frac{M}{\frac{k}{\ell}+2} \le \frac{M}{\frac{kM}{B}+2}$ live frames. The adversary, on the other hand, has none of its packets arrive in a bursty manner, and thus can deliver all of its $M/2$ 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 $\Omega(\frac{kM}{B})$.

Since we can repeat this arrival pattern arbitrarily many times, the result follows. □

We now turn to combine the two adversaries described in Lemmas 1 and 2, and provide a combined lower bound which applies to any relation between M and B.

Corollary 3. *Any deterministic algorithm for* MFG *with* M *streams has competitive ratio* $\Omega(\frac{kM}{B})$.

Proof: Given Lemmas 1 and 2, the only case left to consider is where $M \ge 4B$, and $k > c$, for some absolute constant c. Consider the arrival of the first frame of each stream. We begin by using the adversary described in Lemma 1. At time $T + M/2 + 1$, the adversary has delivered 2 packets of each of its frames, whereas ALG has delivered 2 packets of at most B of its frames. At this point, we can bootstrap the adversary described in Lemma 2, focusing merely on the B frames which are still relevant for ALG. We can therefore force ALG to drop all but an $O(\frac{1}{k})$ fraction of these B frames (the adversary need not concern itself at all with these frames). Once this adversary has completed its injection pattern for ALG's frames, we have the remaining packets corresponding to the adversary's $M/2$ frames arrive one at each time step, such that the adversary can deliver them all.

It follows that the ratio between the number of packets delivered by ALG and that delivered by the adversary is $\Omega(\frac{kM}{B})$. By repeating this arrival pattern arbitrarily many times, the result follows. □

4 Buffer Management Algorithms and Performance Bounds

In this section we describe buffer management algorithms for the MFG problem, where traffic consists of an interleaving of multiple streams.

4.1 Design Criteria

By a close examination of the adversaries described in the proofs of Lemmas 1 and 2, one can see that the adversary manages to force any algorithm to drop frames without resorting to forcing the algorithm to do any type of preemption. The traffic produced by the adversary actually puts all the emphasis on the algorithm's ability to discern which packets to accept to the buffer, and which to drop upon arrival. Note that these decisions are made when the algorithm's buffer is empty, so no preemption is required. The intuition that the main difficulty lies in admitting the "right" packets into the buffer is also implicit in the work of [11] where the best performing algorithm never preempts admitted packets.

By further examining the adversaries of Lemmas 1 and 2, one can notice that the traffic generated by the adversary forces any algorithm to focus on specific streams/frames, and dropping frames that would eventually turn out to be easier to manage. One should note that since arrivals are online, the algorithm has no way of discerning these frames at the point it is forced to make a decision.

Combining these observations leads to the following design criteria for competitive algorithms for our problem:

1) *No-regret policy*: Once a frame has a packet *admitted* to the buffer, make every attempt possible to deliver the complete frame.
2) *Ensure progress*: Ensure the delivery of a *complete frame* as early as possible.

To implement this criteria, we will use a dynamic ranking scheme for the traffic. The second criteria is the more intuitive one, which takes form in the usage of preemption rules. The balancing between the two criteria is done by a definition of the delicate interplay between the ranking-scheme and the preemption rules.

4.2 The WeightPriority Algorithm

We now turn to present our main algorithm, WeightPriority, which follows the design criteria outlined in the previous section. In (the beginning of the arrival step of) any cycle t, and for every frame f_i^m we define its *rank at* t by

$$r_t(f_i^m) = (w_t(f_i^m), m)$$

where $w_t(f_i^m)$ denotes the number of packets of f_i^m already delivered. For every t, the above ranking implies a strict order on all frames, where for every two frames $f_i^m, f_{i'}^{m'}$, f_i^m has rank at least as high as $f_{i'}^{m'}$ if and only if $r_t(f_i^m) \geq r_t(f_{i'}^{m'})$ lexicographically. For completeness, we also define a tie-breaking rule for frames of the same stream, where given any two such frames corresponding to the same stream we consider the frame with the lower frame index as having the higher rank (i.e., we are biased towards earlier frames of the same stream).

We say a frame is *alive* if none of its packets have been dropped yet, and a frame f_i^m is said to be *active* at time t if f_i^m is alive at time t, and $w_t(f_i^m) > 0$. Note that by the definition of the streams, at any time t, at most one frame can be active in every stream. Since the rank of a frame depends on its weight (which may only change during a delivery step) and its invariant stream (and frame) index, the rank of a frame does not change during the arrival step. However, a frame can stop being alive during the arrival step due to having some of its packets dropped.

We let $A_t(f_i^m)$ denote the set of packets of f_i^m arriving at time t. In what follows, whenever we refer to $A_t(f_i^m)$, we implicitly assume that $A_t(f_i^m) \neq \emptyset$. Let $R_t(f_i^m)$ denote the set of packets residing in the buffer at the arrival step of time t, when we consider $A_t(f_i^m)$. Given any frame f_i^m, we abuse notation and denote $p \in f_i^m$ if p is one of the packets of frame f_i^m. We extend the rank function r_t to be defined over packets as well, such that for every $p_j^{m,i}$, $r_t(p_j^{m,i}) = r_t(f_i^m)$.

Assume we handle the arrivals in batches corresponding to distinct frames, i.e., at any cycle t, we get all packets in $A_t(f_i^m)$ together. Furthermore, assume we handle batches $A_t(f_i^m)$ in decreasing order of $r_t(f_i^m)$. For every set of packets $A_t(f_i^m)$, we let

$$L_t(f_i^m) = \left\{ p_j^{m',i'} \in R_t(f_i^m) \mid r_t(p_j^{m',i'}) < r_t(f_i^m) \right\}.$$

$L_t(f_i^m)$ denotes the set of packets of priority lower than f_i^m residing in the buffer when considering $A_t(f_i^m)$. We further define $H_t(f_i^m) = R_t(f_i^m) \setminus L_t(f_i^m)$. We assume $L_t(f_i^m)$ is ordered in ascending order by the $r_t(\cdot)$ order. We say that $D \subseteq L_t(f_i^m)$ is a *prefix* of $L_t(f_i^m)$ if for every frame $f_{i'}^{m'}$ satisfying $f_{i'}^{m'} \cap L_t(f_i^m) \neq \emptyset$, either $f_{i'}^{m'} \cap D = \emptyset$ or $f_{i'}^{m'} \cap (L_t(f_i^m) \setminus D) = \emptyset$, and if for any $p \in D \cap f_{i'}^{m'}$ and $p' \in (L_t(f_i^m) \setminus D) \cap f_{i''}^{m''}$, $r_t(f_{i'}^{m'}) < r_t(f_{i''}^{m''})$. With these requirements, if a packet p is in a prefix then so are all other packets of that frame that are currently in the buffer. Also, packets in a prefix have a rank strictly lower than all other packets in the buffer.

Given $A_t(f_i^m)$, we say a set of packets $Y \subseteq R_t(f_i^m)$ *yields to* $A_t(f_i^m)$ if $|R_t(f_i^m)| - |Y| + |A_t(f_i^m)| \leq B$. In this case, by dropping Y, we are able to accept $A_t(f_i^m)$. Furthermore, for any $A_t(f_i^m)$ we let $D_t(f_i^m) \subseteq L_t(f_i^m)$ be the minimal size prefix of $L_t(f_i^m)$ that yields to $A_t(f_i^m)$.

We can now define our algorithm, WEIGHTPRIORITY (or WP, for short), which upon overflow prefers to keep packets of higher-ranking frames. The formal definition of algorithm WP appears in Algorithm 1.

Algorithm 1 WEIGHTPRIORITY: upon the arrival of $A_t(f_i^m)$

```
1: if $f_i^m$ is alive and $L_t(f_i^m)$ yields to $A_t(f_i^m)$ then
2:     let $D_t(f_i^m) \in \mathcal{P}(L_t(f_i^m))$ be the minimal size set
       that yields to $A_t(f_i^m)$
3:     preempt $D_t(f_i^m)$
4:     accept $A_t(f_i^m)$
5: else
6:     reject $A_t(f_i^m)$
7:     drop $R_t(f_i^m) \cap f_i^m$
8: end if
9:
```

In the remainder of this section we prove the following theorem:

Theorem 4. *The competitive ratio of* WEIGHTPRIORITY *is* $O((kMB + M)^{k+1})$.

We first give a high-level description of our analysis. We would like to map frames delivered by an optimal solution to frames delivered by WP. To this end, we consider a partition of time into disjoint intervals, and identify every such interval with the highest ranking packet delivered during this interval. This implies a ranking over the intervals. We then map every interval to a strictly-higher ranking interval, culminating in an interval in which a frame is successfully delivered. By showing that there exists a number ℓ that bounds both (a) the number of frames successfully delivered by an optimal solution during any interval, and (b) the number of intervals mapped to any single interval, one obtains a k-height ℓ-ary tree-like structure underlying the mappings of intervals, which implies the required result.

We begin our analysis by examining some of the properties of WP, captured by the following lemmas.

Lemma 5. *If a packet $p \in f_i^m$ is dropped by* WP *at time t, then at the end of the arrival step of time t the buffer holds at least one packet with rank strictly greater than $r_t(f_i^m)$.*

Proof: Consider first the case where p is dropped because of rejecting $A_t(f_i^m)$ upon arrival. We could have either $p \in A_t(f_i^m)$, or p is already in the buffer when we consider $A_t(f_i^m)$. Since packets in $f_i^m \cap R_t(f_i^m)$ are also in $H_t(f_i^m)$, and since we cannot accommodate $A_t(f_i^m)$, we also drop all packets of f_i^m currently in $H_t(f_i^m)$. We therefore have $|L_t(f_i^m)| < |A_t(f_i^m)|$, and $|R_t(f_i^m) \cap f_i^m| \leq k - |A_t(f_i^m)|$, implying that the overall number of packets in the buffer that have priority at most $r_t(f_i^m)$ is strictly less than k. Hence, there are at least $B - k + 1 \geq 1$ packets with priority strictly greater than $r_t(f_i^m)$ (by our assumption that $B \geq k$). Note that for any $A_t(f_{i'}^{m'})$ considered after $A_t(f_i^m)$, its rank is strictly less than $r_t(f_i^m)$, due to the assumption on the order in which packets are processed. Hence, none of the packets corresponding to other packets in $H_t(f_i^m)$ would be preempted during cycle t.

Consider now the case where p is dropped due to preemption when considering $A_t(f_{i'}^{m'})$. By the definition of WP in line 3 this can only happen if $p \in D_t(f_{i'}^{m'}) \subseteq L_t(f_{i'}^{m'})$ for some frame $f_{i'}^{m'}$ such that $r_t(f_{i'}^{m'}) > r_t(p)$. Since any subsequent set of packets $A_t(f)$ considered after $A_t(f_{i'}^{m'})$ has rank strictly less than $r_t(f_{i'}^{m'})$, which follows from the assumption that we consider arrivals in decreasing order of their rank, and by WP's preemption condition in line 1, none of the packets of $A_t(f_{i'}^{m'})$ admitted to the buffer at time t would be preempted at time t. It follows that at least one packet with rank strictly greater than $r_t(f_i^m)$ still resides in the buffer at the end of the arrival step of time t. $\square$

In particular, the above lemma shows that at least one frame is still alive after each cycle where overflow occurs. The following lemma gives a bound on the number of active frames in any time interval.

Lemma 6. *For any time interval I, the number of active frames during I is at most $M + |I|$.*

Proof: Assume without loss of generality that $I = (0, \ldots, \ell]$. The number of active frames in cycle 1 is at most M, since at any point in time, each stream can have at most one active frame (by the order of frames within each stream, and the fact that the buffer follows a FIFO discipline). Since in any cycle at most one packet is sent, this could increase the number of active frames by at most 1 in each cycle. The result follows. $\square$

Let $\{t_\ell\}_{\ell \geq 0}$ be the sequence of cycles where overflows occur, such that the $(\ell+1)$'th cycle where overflow occurs is t_ℓ. For every overflow cycle t_ℓ, we define the interval $I_\ell = (t_\ell, t_\ell + B]$.

We henceforth identify every frame f_i^m that is not delivered by WP with the *first* packet $p_j^{m,i} \in f_i^m$ that is not delivered by WP. In such a case, we say packet $p_j^{m,i}$ is *responsible* for dropping frame f_i^m. Note that responsible packets are always dropped due to overflow. We can therefore associate each frame f_i^m that is not delivered by WP with a single overflow cycle where its responsible packet is dropped.

For every overflow cycle t_ℓ, we let f_ℓ denote the highest ranking frame that has a packet in the buffer at the end of cycle t_ℓ, and let $p_\ell \in f_\ell$ denote the packet closest to the head of the buffer at the end of that cycle.

We now define a mapping over the set of frames $\{f_\ell\}_{\ell \geq 0}$. Consider any f_ℓ for which p_ℓ is not delivered by WP. This can only happen due to overflow which occurs at some time $t_{\ell'} \in (t_\ell, t_\ell + B]$. In such a case we map f_ℓ to $f_{\ell'}$. This mapping essentially defines sequences of frames $f_{\ell_1}, f_{\ell_2}, \ldots$. By Lemma 5 we are guaranteed that for any j, $r_{t_{\ell_j}}(f_{\ell_j}) > r_{t_{\ell_j}}(f_{\ell_{j-1}})$, i.e., as we progress in the sequence, the rank of the frames strictly increases. Since our arrival sequence is finite, any such sequence must be finite, and can therefore be referred to as $(f_{\ell_1}, \ldots, f_{z(\ell_1)})$ for some index $z(\ell_1)$. We henceforth refer to such a sequence as an $(f_{\ell_1}, f_{z(\ell_1)})$-*preemption sequence*. The following lemma shows some properties of such a sequence.

Lemma 7. *Any $(f_{\ell_1}, f_{z(\ell_1)})$-preemption sequence satisfies the following properties:*

1) *The sequence contains at most kM frames.*
2) *The overflow cycles associated with the sequence are contained in a time interval of length at most kMB.*
3) *The packet $p_{z(\ell_1)} \in f_{z(\ell_1)}$ is delivered by* WP *at some cycle $t \in (t_{z(\ell_1)}, t_{z(\ell_1)} + B]$. Hence, $w_{t_{z(\ell_1)}+B}(f_{z(\ell_1)}) \geq w_{t_{z(\ell_1)}}(f_{z(\ell_1)}) + 1$, that is, the weight increases by at least 1 by time $t_{z(\ell_1)} + B$.*

Proof: Since the rank of the frames along the sequence are strictly increasing, and the number of distinct ranks is kM, it follows that the sequence cannot be longer than kM.

Since by definition of the mapping, the preemption of p_{ℓ_j} occurs at time $t_{\ell_{j+1}} \in (t_{\ell_j}, t_{\ell_j} + B]$, and since the sequence is of length at most kM, it follows that all overflow cycles associated with the sequence are contained in the time interval $(t_{\ell_1}, t_{\ell_j} + kMB]$.

Since $f_{z(\ell_1)}$ is the last frame of the sequence, by definition of the mapping it follows that $p_{z(\ell_1)} \in f_{z(\ell_1)}$ is delivered by WP. This further occurs at some cycle $t \in (t_{z(\ell_1)}, t_{z(\ell_1)} + B)$, since $p_{z(\ell_1)}$ is in the buffer but not yet delivered at cycle $t_{z(\ell_1)}$. It therefore follows that the weight of $f_{z(\ell_1)}$ increases by 1 by cycle $t_{z(\ell_1)} + B$. $\square$

We now define a sequence of non-overlapping intervals $\mathcal{S} = \{I_1, I_2, \ldots\}$. Each I_j will be of length kMB, starting from some overflow cycle t_{ℓ_j}, such that each overflow cycle is contained in exactly one interval in $\mathcal{S}$.

1) $I_1 = (t_1, t_1 + kMB]$.
2) Given I_{j-1}, we let t_{ℓ_j} denote the earliest overflow cycle after I_{j-1}, and define $I_j = (t_{\ell_j}, t_{\ell_j} + kMB]$.

For every such interval I_j starting at t_{ℓ_j}, we can consider the $(f_{\ell_j}, f_{z(\ell_j)})$-preemption sequence. We associate I_j with frame $f_{z(\ell_j)}$, that by property (3) in Lemma 7 is guaranteed to have its weight increased by time $t_{z(\ell_j)}$. Every such interval can be associated with the maximum priority packet delivered by WP during that interval, denoted $\tilde{p}_{\ell_j} \in \tilde{f}_{\ell_j}$. We hereby refer to this frame as the *increase-frame* of interval I_j. Note that $\tilde{f}_{\ell_j}$ need not necessarily be one of the frames in the preemption-sequence. However, by the maximality of the rank of $\tilde{p}_{\ell_j}$, along with property (3) in Lemma 7, we are guaranteed that it has a *weight* strictly greater than $f_{z(\ell_j)}$ at time $t_{z(\ell_j)}$.

We can now complete the proof of Theorem 4.

Proof of Theorem 4: We will map frames accepted by an optimal solution, OPT, but dropped by WP, to frames delivered by WP. Our mapping will ensure that every frame delivered by WP is mapped to by at most $O((kMB + M)^{k+1})$ packets, which would complete the proof.

Let $f_i^m \in \mathrm{OPT} \setminus \mathrm{WP}$. Since f_i^m is dropped by WP, there exists an overflow cycle t_ℓ where the responsible packet of f_i^m is dropped. By the definition of our sequence of intervals $\mathcal{S}$, there exists an interval $I_j \in \mathcal{S}$ such that $t_\ell \in I_j$. We map f_i^m to the increase-frame of interval

I_j. Since $|I_j| \leq kMB$, it follows that there are at most $kMB + B$ packets accepted by OPT during such an interval. In particular, there can be at most $kMB+B$ responsible packets accepted by OPT during such an interval. This implies that the above method maps at most $kMB + B$ packets from $\mathrm{OPT} \setminus \mathrm{WP}$ to any single increase-frame.

We now describe a mapping over the set of increase-frames. Consider any increase frame $\tilde{f}_{\ell_j}$. If this frame is not delivered successfully by the algorithm, it is because its responsible packet p was dropped at some overflow cycle $t_{\ell'}$. Let $I_{j'} \in \mathcal{S}$ be the interval for which $t_{\ell'} \in I_{j'}$. We map $\tilde{f}_{\ell_j}$ to $\tilde{f}_{\ell_{j'}}$. By the definition of increase-frames, and similarly to the argument used for the preemption-sequences, where we also defined sequences of frames with strictly increasing weights, one can verify that the *weight* of $\tilde{f}_{\ell_{j'}}$ is strictly greater than the weight of $\tilde{f}_{\ell_j}$. It follows that the length of any such sequence is at most k, and the last frame of every such sequence is delivered successfully by WP.

We now turn to consider the overall number of increase-frames mapped to a single increase-frame that is successfully delivered by WP. By definition, any increase-frame is active, since at least one of its packets is delivered by WP.

By the fact that any interval I_j is of length kMB, combined with Lemma 6, the overall number of active frames encountered by WP during I_j is at most $kMB+M$. Since by definition of our mapping, every increase-frame $\tilde{f}_{\ell_{j'}}$ is mapped-to only by increase-frames that are active during $I_{j'}$, it follows that every increase-frame is mapped-to (directly) by at most $kMB+M$ increase-frames. Hence, the overall number of increase-frames mapped to a single increase-frame that is successfully delivered is bounded by the size of the $(kMB + M)$-ary tree of height k.

By combining this with our upper bound on the number of frames in $\mathrm{OPT} \setminus \mathrm{WP}$ mapped to any single increase-frame (i.e., to a "leaf" of the tree), we obtain that every frame successfully delivered by WP is mapped to by at most $(kMB + B)(kMB + M)^k = O((kMB + M)^{k+1})$, as required. $\square$

5 SIMULATION STUDY

We provide a simulation study where we compare the performance of several buffer management and discard policies for traffic with inter-packet dependencies. Specifically, we examine the impact of various traffic characteristics on the performance of our proposed algorithms. We compare the performance of the considered algorithms with the performance of the best known *offline* algorithm as a benchmark. The offline algorithm tries to pack complete frames onto the buffer greedily and is guaranteed to provide a $(k + 1)$-approximation [11].

In addition to our WEIGHTPRIORITY algorithm we consider the performance of several other online reference algorithms.

The first additional algorithm, referred to as FRAMEOBLIVIOUS, disregards the frame structure altogether. Upon overflow, it simply refrains from accepting further packets, regardless of the identity of the frames to which they correspond. This algorithm is appealing due to the fact that it does not need to maintain any state information, and does not perform a buffer scan upon overflow. The second additional algorithm, referred to as SEMIFRAMEOBLIVIOUS, is similar to FRAMEOBLIVIOUS, except for the fact that it scans the buffer upon overflow, and drops any packets residing in the buffer that correspond to the packet dropped due to the overflow. This algorithm also does not require any state information, however, it performs a buffer scan upon overflow. The third algorithm, referred to as STREAMOBLIVIOUS, drops an overflowing packet, as well as all other packets of its frame. This algorithm both scans the buffer upon overflow, and maintain state information per stream, in order to drop also future arriving packets of the dropped frame. We note that these algorithms do not follow either of our design criteria.

We further consider one additional algorithm, referred to as STREAMPRIORITY (SP), which conforms with our design criteria. This algorithm is similar to our WEIGHT-PRIORITY algorithm, except for its choice of rank criteria. SP uses the stream index to define a complete order over all frames (using the same tie-breaking rule used by WP for frames corresponding to the same stream). SP maintains state information per stream, however, it does not require maintaining counters or performing packet inspection in order to determine the dynamic weight of the stream.

5.1 Traffic Generation and Setup

We study the performance of all algorithms under high load for an aggregate of multiple bursty streams. Each stream is generated using a Markov modulated Poisson process (MMPP) with two states, `ON` and `OFF`, with symmetric transition rates. In the `ON` state, unit-size packets are generated with a rate of λ, which results in an average rate across both `ON` and `OFF` states of $\lambda/2$, that is, the effective rate is half the `ON` rate. Using this process, each stream generates a sequence of 200 frames, where each frame has k packets.

5.2 Simulation Results

Figures 2–4 depict the results of the simulations. We compare the goodput of the algorithms with the benchmark offline algorithm. The performance metric is the goodput ratio, defined as the ratio between the number of frames successfully delivered by the algorithm, and the number of frames successfully delivered by the benchmark offline algorithm. The results show the average of a set of 6 simulations for each choice of parameters, as well as the confidence intervals. We now discuss the conclusions derived from our simulation study.

Figure 2 shows the impact of the frame size on the performance of each algorithm. The traffic is an aggregate of $M = 50$ streams, each generated by an MMPP process as described above with an average per-stream rate of 0.025 (implying an aggregate total average rate of 1.25). This

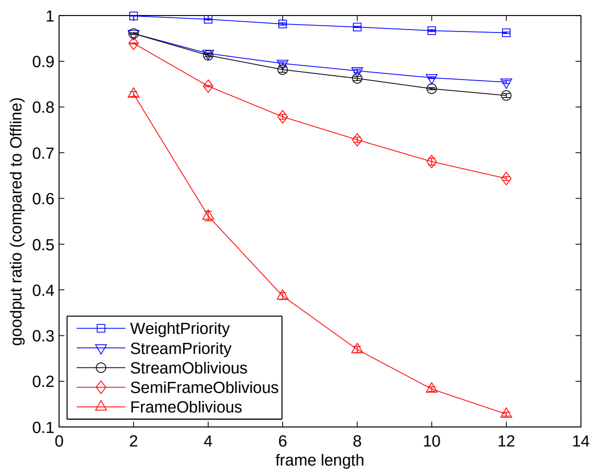

Fig. 2. Goodput ratio as a function of frame size ($M = 50, B = 12$).

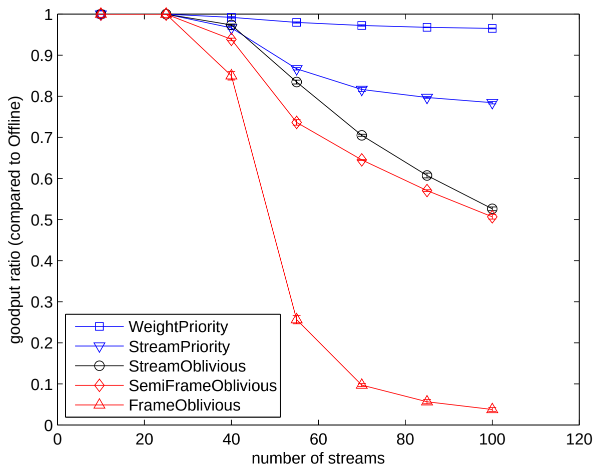

Fig. 3. Goodput ratio as a function of number of streams ($k = 6, B = 12$, average stream rate of 0.025).

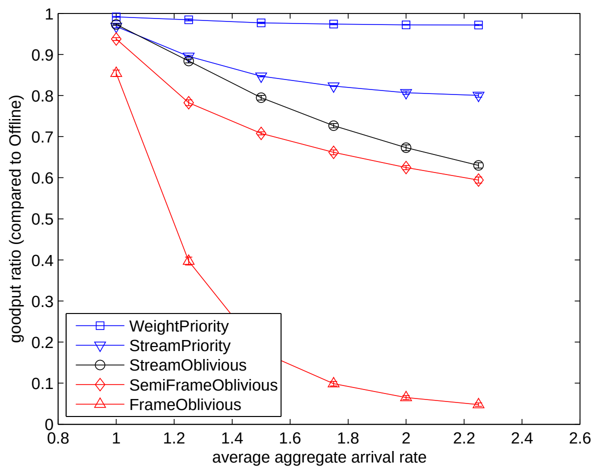

Fig. 4. Goodput ratio as a function of aggregate average arrival rate ($M = 50, k = 6, B = 12$).

models a high-load system, since the service rate is 1. The buffer size bound is set to $B = 12$, and the frame size is set to $k = 6$.

Figure 3 shows the sensitivity of increasing the number of streams, while keeping the per-stream rate fixed, on the performance of each algorithm. The average per-stream rate is 0.025. Hence, as long as the number of streams is below 40, the average aggregate arrival rate is below the service rate of 1. The performance of all algorithms starts to degrade as the system becomes more overloaded.

Figure 4 shows the effect of increasing the overall load of traffic, while keeping the number of streams fixed. This is done by increasing the rate of each stream uniformly. As can be seen in Figure 4, if the load is increased by increasing the rate of each stream, while keeping the number of streams fixed, we obtain the same effect as increasing the number of streams while maintaining the rate of each stream fixed. This is consistent with the fact that the goodput of our algorithms is a function of the local per-frame performance. Hence the correspondence of a frame to a stream has a marginal impact on the overall system goodput.

Our experiments indicate that both algorithms satisfying our design criteria result in a stabilized goodput as traffic conditions become harder (either in terms of load, or in terms of decision-complexity, underlined by the increase in the number of packets per frame). The three algorithms that do not conform with our guidelines exhibit a fast degradation in performance as traffic conditions become harder. Specifically, the performance of WP is within 96% of the performance of the best known offline algorithm for the problem. The performance of SP, although inferior to that of WP, also guarantees at least 80% of the goodput achieved by the reference offline algorithm.

6 Conclusions and Future Work

We addressed the problem of managing buffer overflows for traffic consisting of multiple streams, with inter-packet dependencies.

We provided guidelines for the design of such algorithms, and analyzed the performance of one such algorithm, both from a worst-case competitive approach, as well as by a simulation study. We provided guarantees on its performance under any traffic conditions by proving it has a bounded competitive ratio. We also showed that the competitive ratio of any algorithm for our problem might degrade linearly as a function of the number of streams in the traffic. Our simulation study shows that algorithms that follow our proposed design criteria yield a close to optimal performance under variable traffic characteristics and load, while algorithms that fail to adhere to our guidelines show a fast degradation in performance as traffic characteristics become more intense.

Our work raises several interesting questions. For example, what is the interplay between the buffer management algorithm and coding, and how can prior information about the coding scheme (and its redundancy) be taken into account by the algorithm for improving performance? Another question is how to design buffer management for such settings, with provable performance guarantees, and settle the gap in the competitive ratio of algorithms for the problem.

References

[1] iSuppli Market Intelligence, http://www.isuppli.com/, 2009.

[2] J. M. Boyce and R. D. Gaglianello, "Packet loss effects on MPEG video sent over the public internet," in *Proceedings of the 6th ACM International Conference on Multimedia*, 1998, pp. 181–190.

[3] E. W. Knightly and N. B. Shroff, "Admission control for statistical QoS: Theory and practice," *IEEE Network*, vol. 13, no. 2, pp. 20–29, 1999.

[4] D. E. Wrege, E. W. Knightly, H. Zhang, and J. Liebeherr, "Deterministic delay bounds for VBR video in packet-switching networks: fundamental limits and practical trade-offs," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 352–362, 1996.

[5] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R. P. Tsang, "Efficient selective frame discard algorithms for stored video delivery across resource constrained networks," *Real-Time Imaging*, vol. 7, no. 3, pp. 255–273, 2001.

[6] A. Kesselman and Y. Mansour, "QoS-competitive video buffering," *Computing and Informatics*, vol. 21, no. 6, pp. 1001–1018, 2002.

[7] E. Gürses, G. B. Akar, and N. Akar, "A simple and effective mechanism for stored video streaming with TCP transport and server-side adaptive frame discard," *Computer Networks*, vol. 48, no. 4, pp. 489–501, 2005.

[8] S. Ramanathan, P. V. Rangan, H. M. Vin, and S. S. Kumar, "Enforcing application-level QoS by frame-induced packet discarding in video communications," *Computer Communications*, vol. 18, no. 10, pp. 742–754, 1995.

[9] A. Awad, M. W. McKinnon, and R. Sivakumar, "Goodput estimation for an access node buffer carrying correlated video traffic," in *Proceedings of the 7th IEEE Symposium on Computers and Communications (ISCC)*, 2002, pp. 120–125.

[10] A. Ziviani, J. F. de Rezende, O. C. M. B. Duarte, and S. Fdida, "Improving the delivery quality of MPEG video streams by using differentiated services," in *Proceedings of the 2nd European Conference on Universal Multiservice Networks (ECUMN)*, 2002, pp. 107–115.

[11] A. Kesselman, B. Patt-Shamir, and G. Scalosub, "Competitive buffer management with packet dependencies," in *Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2009.

[12] Y. Emek, M. M. Halldórsson, Y. Mansour, B. Patt-Shamir, J. Radhakrishnan, and D. Rawitz, "Online set packing and competitive scheduling of multi-part tasks," in *Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC)*, 2010, pp. 440–449.

[13] 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)*, 2003, pp. 761–770.

[14] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, "Buffer overflow management in QoS switches," *SIAM Journal on Computing*, vol. 33, no. 3, pp. 563–583, 2004.

[15] M. H. Goldwasser, "A survey of buffer management policies for packet switches," *ACM SIGACT News*, vol. 41, no. 1, pp. 100–128, 2010.

[16] O. Bonaventure and J. Nelissen, "Guaranteed frame rate: a better service for TCP/IP in ATM networks," *IEEE Network*, vol. 15, no. 1, pp. 46–54, 2001.

[17] A. Romanow and S. Floyd, "Dynamics of TCP traffic over ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 4, pp. 633–641, 1995.

[18] A. Mehaoua, R. Boutaba, Y. Rasheed, and A. Leon-Garcia, "An integrated framework for efficient transport of real-time MPEG video over ATM best effort service," *Real-Time Imaging*, vol. 7, no. 3, pp. 287–300, 2001.

[19] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan, "Priority encoding transmission," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1737–1744, 1996.

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

[21] T. Stockhammer, H. Jenkač, and G. Kuhn, "Streaming video over variable bit-rate wireless channels," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 268–277, 2004.

[22] I. F. Akyildiz, T. Melodiaa, and K. R. Chowdhurya, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921–960, 2007.

[23] B. Girod, M. Kalman, Y. J. Liang, and R. Zhang, "Advances in channel-adaptive video streaming," *Wireless Communications and Mobile Computing*, vol. 2, no. 6, pp. 573–584, 2002.

[24] 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.

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