Buffer Management for Aggregated Streaming Data with Packet Dependencies

Gabriel Scalosub
Dept. of Communication Systems Engineering
Ben Gurion University of the Negev
Beer-Sheva 84105, Israel
sgabriel@cse.bgu.ac.il

Peter Marbach
Department of Computer Science
University of Toronto
Toronto, ON, Canada
marbach@cs.toronto.edu

Jörg Liebeherr
Department of Electrical Engineering
University of Toronto
Toronto, ON, Canada
jorg@comm.utoronto.ca

***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 discards. 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 the effectiveness of these criteria. We devise an algorithm according to these guidelines and evaluate its performance analytically, using competitive analysis. We also present a simulation study that shows that the performance of our algorithm is within a small fraction of the performance of the best offline algorithm.

I. Introduction

In the vast majority of networked applications, application-layer data frames are split into several smaller sized packets, which are sent across the network. The receiving side can make use of the data only if it receives *all* (or at least sufficiently many of) the packets of a frame. Most current best-effort networks, such as the Internet, are oblivious to such an inter-packet dependency structure, and usually make discard decisions on a per-packet base. Higher-level mechanisms in the protocol stack usually handle retransmissions of lost packets, in order to provide adequate performance for the application. The exact dependency structure of the data stream depends on the encoding used, and it may consist of a 1-level dependency structure (i.e., frames are independent of each other, and the only dependencies are among packets corresponding to the same frame), and/or higher dependency structure (e.g., the dependency structure occurring in MPEG video encoding schemes, where successfully decoding a frame might depend on successfully decoding other previous/later frames). The problem of ensuring that all packets of a frame arrive at the destination is especially 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. The widespread popularity of such services over the Internet (e.g., IPTV) serves as a main motivation for providing better algorithmic solutions to ameliorate network performance in such scenarios.

A common approach to deal with packet losses is to employ proactive encoding schemes, which have long been known to provide substantial improvement in performance. However, this approach has its limitations in several networking environments. Specifically, some environments (e.g., wireless networks) make the usage of such approaches prohibitively costly, due to the increased traffic load. Also, in some scenarios where traffic may traverse bottleneck links, the effect of coding diminishes substantially, since the bottleneck fully determines the loss characteristics incurred by the traffic. These scenarios occur, e.g., at head-ends of content distribution networks, wireless gateways, and input queues of network transcoders. We therefore believe that the availability of good encoding schemes 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. All of these features make the FIFO architecture appealing, especially for delay- and reordering-constrained streaming environments.

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. One should 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. As an extreme example consider the case where one packet is dropped from every frame, which results in zero goodput, although overall packet-level throughput is high (this effect has been verified in experimental studies, e.g., [1]). 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. Our goal is to devise methods that maximize the goodput of successfully delivered traffic, 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 guarantees as to the performance of our proposed algorithms for *any* traffic arrival pattern, and without any stochastic or deterministic assumptions on the processes generating the traffic. Different from works which focused on estimating the system's performance using either statistical or deterministic models for traffic (e.g., [2], [3]), we study packet discard policies in a 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 available buffer size) and system performance. We restrict our attention to traffic with a 1-level dependency structure. A better understanding of the algorithmic problems in this setting is essential before tackling more complex dependency structures. 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.

A. Our Contribution

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

1) We provide two design guidelines for algorithms in such an environment: (a) *No-regret* – Make every attempt to deliver a frame that has a packet *admitted* to the buffer; and (b) *Ensure-progress* – deliver a complete frame as soon as possible.
2) We devise a buffer management algorithm, WEIGHTPRIORITY, that follows these guidelines.
3) We analyze the performance of our algorithm, and show that for *any* traffic the ratio between its performance and that of an optimal algorithm is always bounded.
4) We prove lower bounds on the performance of any buffer management algorithm.
5) We conduct a simulation study that further shows the benefits of our design criteria, as opposed to algorithms that fail to adhere to these criteria.

B. Previous Work

Various aspects of providing traffic with Quality-of-Service (QoS) guarantees have been studied extensively in recent years. 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 [4], [5]. Some attempts were made to answer the question of which *frame* to drop, based on the specific encoding scheme, so as to minimize the effect on the received data stream (e.g., [6]). However, discard decisions at the routers are made at the *packet* level, and their scope is limited to packets stored in the buffer and newly arriving packets.

Ramanathan et al. [7] propose a scheme that takes packet dependencies into account, namely, that in any case where too many packets are dropped, one drops the entire frame. They then evaluate their scheme assuming Markovian video sources.

Our model is close to that considered by Kesselman et al. [8]. They consider the case where there is no stream-structure underlying the arrival traffic (i.e., every frame is independent of all other frames), as well as other restricted arrival patterns that are not general enough to model arrival patterns encountered in real-life networks. They provide lower bounds and competitive algorithms for these settings. Our model is more general than the model they consider, and captures the structure of real-life data streams. Additional works that consider competitive algorithms that provide QoS have been studied (e.g., [9]), but none of them addresses inter-packet dependencies.

There has been much work done on proactive encoding schemes (commonly known as forward error correction, or FEC) for packetized traffic of larger data frames. Most of this work focuses on implementation and information-theoretic aspects of such schemes (e.g., [10], [11]). Although our model assumes no redundancy (i.e., upon losing a single packet of a frame, the entire frame is rendered useless), we believe a better understanding of this basic scenario is the starting point for designing algorithms that additionally account for proactive coding.

II. MODEL

The input to the system consists 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; this notation is interpreted as follows: p_j^m is 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., $a(p_j^m) \leq a(p_{j+1}^m)$ 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 essentially implies a finite arrival sequence σ of the aggregated streams, which is the interleaving of the arrival sequences of the individual streams. See Figure 1 for an example. It should be

$$\sigma = p_1^{1,0},\ p_1^{2,0},\ p_2^{2,0},\ p_1^{2,1},\ p_2^{1,0},\ p_2^{2,1},\ p_1^{1,1},\ p_2^{1,1}$$

Fig. 1. Example of an interleaved arrival sequence ($M = k = 2$).

noted that we make no assumptions (either deterministic, or stochastic) on the processes generating the arrival sequences.

The packets arrive at a FIFO buffer which has a capacity of $B \geq k$ packets, and has an output link that 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 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 system. At the discretion of the buffer management algorithm, some packets may be dropped, while other packets are stored in the buffer. The FIFO order of buffered packets is always maintained. A feasible system must satisfy the *capacity constraint*, according to which the maximum number of packets in the buffer must not exceed the given buffer size B. Note that a buffer management algorithm may drop packets even if there is space available at the buffer.

We say a frame is *successfully delivered* by an algorithm ALG if ALG delivers all the packets belonging to the frame. The MAX-FRAME-GOODPUT problem, denoted MFG, is to devise a buffer management algorithm which maximizes the number of successfully delivered frames, i.e., the *goodput*. This work studies *online algorithms* for solving the MFG problem, that, at any point in time, know of arrivals that have occurred up to that time but have 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 [12] to evaluate the performance of online algorithms. An algorithm ALG is said to be *c-competitive* if for any traffic arrival sequences, the goodput of *any* feasible schedule (and in particular, the optimal one) is at most c times the goodput of ALG, for some $c \geq 1$.

III. ANALYTICAL STUDY

In this section we study the performance of algorithms for the MFG problem. We first show that the performance of *any* algorithm can be made to degrade linearly in the number of streams. This is shown by the following theorem (proof omitted due to space constraints).

Theorem 1. *Any deterministic algorithm for* MFG *with M streams has competitive ratio $\Omega(\frac{kM}{B})$.*

We now turn to discuss the design of buffer management algorithms for the MFG problem, where traffic consists of an interleaving of multiple streams. We begin by identifying several design criteria, which follow from a close examination of the traffic patterns forcing the performance degradation of any algorithm, as given by Theorem 1. These traffic patterns manage to force any algorithm to drop frames without requiring the algorithm to do any type of preemption, and thus test the algorithm's ability to discern which packets to accept to the buffer, and which to drop upon arrival. The intuition that the main difficulty is admitting the "right" packets into the buffer is also implicit in the work of [8] where the only competitive algorithm they provide (for arbitrary frame length k) never preempts admitted packets.

An additional observation can be made by noticing that the traffic patterns used for proving Theorem 1 cause any algorithm to focus on specific streams/frames, and drop frames that would eventually turn out to be easier to manage (although the algorithm cannot discern these frames when it is forced to make a decision).

Combining these two 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 (not necessarily sent), make every attempt possible to deliver the complete frame.
2) *Ensure progress*: Ensure the delivery of a *complete frame* as early as possible.

The first of these criteria is the less intuitive of the two. It essentially means that for any algorithm, when faced with deciding between seemingly equivalent options, it should try and make a decision that is as consistent as possible with earlier such decisions (this can be viewed as a 'non-zigzagging' principle). In order to implement this criteria, we will use a dynamic ranking scheme for traffic. The second criteria 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.

The WEIGHTPRIORITY *Algorithm:* We now turn to describe our algorithm which follows the above design criteria. 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 and $f_{i'}^{m'}$, 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 any time t, we will extend the definition of rank also to *packets*, such that the rank of a packet at time t is the rank of the frame to which this packet corresponds.

We say a frame f_i^m is *active* at time t if (a) none of f_i^m's packets have been dropped yet, (b) $w_t(f_i^m) > 0$, and (c) f_i^m has not yet been delivered in full. 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 index, the rank of a frame does not change during the arrival step. However, a frame can stop being active during the arrival step when some of its packets are dropped.

Assume we handle the arrivals in batches corresponding to distinct frames, i.e., at any cycle t, the buffer receives all packets corresponding to f_i^m that arrive at cycle t simultaneously. Furthermore, assume the buffer handles batches corresponding to different frames in decreasing order of their rank. Note that these assumptions are *local*, and are restricted to each single cycle individually. They pose no restriction in the common case where we have multiple incoming links, and a single outgoing link where all links have the same capacity.

We now describe the operation of our algorithm WEIGHT-PRIORITY (or WP, for short). Consider any cycle t and frame f_i^m that has packets arriving at t. Denote by $A_t(f_i^m)$ this set of packets, and denote by $L_t(f_i^m)$ the set of packets residing in the buffer at t, whose rank is strictly smaller than the rank of f_i^m. We consider this set as ordered by rank. The algorithm works as follows: If $A_t(f_i^m)$ can be accommodated in the buffer, accept it. Otherwise, if preempting $L_t(f_i^m)$ would leave sufficient room to accept $A_t(f_i^m)$, preempt the minimum size prefix of $L_t(f_i^m)$ that would enable accommodating $A_t(f_i^m)$, and accept $A_t(f_i^m)$. Finally, drop any remaining packets in $L_t(f_i^m)$ of frames that have become inactive due to the preemption triggered by $A_t(f_i^m)$.

Upon overflow, algorithm WP prefers to keep packets of higher-ranked frames. The following theorem provides an absolute bound on the performance of WP in terms of its competitive ratio.

Theorem 2. *The competitive ratio of* WEIGHTPRIORITY *is* $O((kMB + M)^{k+1})$.

Due to space constraints we only provide the idea underlying the proof; 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.

IV. SIMULATION STUDY

In this section we provide selected results (due to space constraints) obtained by 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 effect of various traffic characteristics on the performance of our proposed algorithms. In our study, we consider the performance of each algorithm in comparison with the performance of the best known *offline* algorithm, which serves as a benchmark. This algorithm which essentially tries to pack complete frames onto the buffer greedily, is guaranteed to provide a $(k+1)$-approximation [8].

In addition to our algorithm WEIGHTPRIORITY we also consider the performance of several other 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 is a standard drop-tail algorithm). This algorithm is appealing due to the fact that it does not need to maintain any state information, and does not perform any buffer scanning upon overflow. The second additional algorithm we consider, 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 does perform a buffer scanning upon overflow. The third algorithm we consider, 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, which conforms with our design criteria, referred to as STREAMPRIORITY (SP). This algorithm is similar to our WP algorithm, except for its choice of rank criteria. SP focuses on the mere criteria of stream index, which defines a complete order over all frames (using the same tie-breaking rule used by WP for frames corresponding to the same stream). This algorithm maintains a state information per stream, however, it does not resort to maintaining counters or performing packet inspection in order to determine the dynamic weight of the stream.

a) Traffic Generation and Setup: We study the performance of all algorithms under high load for an aggregate of multiple bursty streams. We now provide some further information about the common elements of our simulation study. Each stream is generated using a Markov modulated Poisson process (MMPP) with two states, `ON` and `OFF`, with symmetric transition rates. During the `ON` stage unit-size packets are generated with a rate of λ, which results in an average rate across both `ON` and `OFF` states of $\lambda/2$ (the effective rate is half the `ON` rate since the transition rates are chosen to be symmetric). Each stream consists of a sequence of 200 frames. The exact number of packets in each stream depends on the size of each frame, which using our previously defined notation k for the frame length, is $200 \times k$.

b) Simulation Results: Figures 2 and 3 depict the results of our simulation study. In both figures we consider the *goodput ratio* of each algorithm, i.e., the ratio between the performance of the algorithm and the performance of the benchmark offline algorithm. The results show the average of a set of simulations for each choice of parameters. Throughout our simulations the confidence intervals were negligible. Figure 2 shows the effect of 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 average per-stream rate is 0.025 (implying an aggregate total average rate of 1.25). This models a high-load system, since the service rate is 1. Figure 3 shows the effect of increasing the number of streams, while keeping the per-stream rate fixed on the performance of each algorithm. The average per-stream

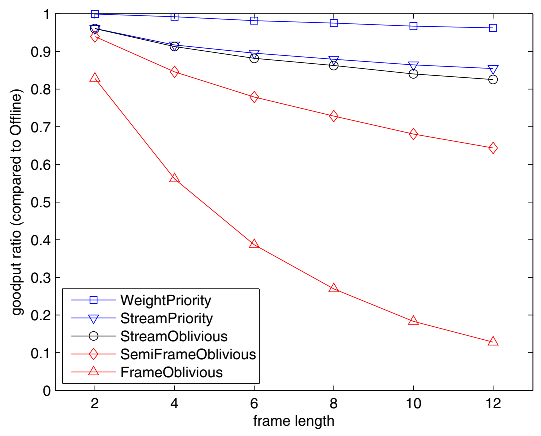

Fig. 2. Goodput ratio as a function of increasing frame size ($M = 50, B = 12$).

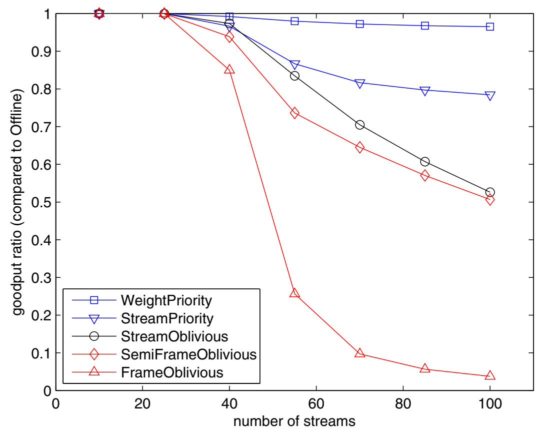

Fig. 3. Goodput ratio as a function of increasing number of streams ($k = 6, B = 12$, average stream rate of 0.025).

rate is 0.025, which implies that while the number of streams is below 40, the average aggregate arrival rate is below the service rate of 1.

Throughout our experiments, the algorithms satisfying our design criteria are shown to have a stabilized throughput as traffic conditions become harder (either in terms of load, or in terms of decision-complexity, implied by the increase in the number of packets per frame). The algorithms that do not conform with our guidelines show a fast degradation in performance as traffic conditions become harder. Specifically, WP exhibits a performance which is within 96% of the performance of the best known offline algorithm for the problem, while SP stabilizes at the 80% mark. This latter finding points to a possible tradeoff: while SP is "stream-neutral", in the sense that its ranking (and therefore drop decisions) are independent of stream identity, its performance in terms of goodput is inferior to that of WP, which prioritizes over streams. This can be seen trading off fairness for goodput.

V. Conclusions and Future Work

In this work we address the problem of managing buffer overflows for traffic consisting of multiple streams with inter-packet dependencies. We provide guidelines for the design of algorithms for the problem, and analyze the performance of one such algorithm, both from a worst-case competitive approach, as well as by a simulation study. We provide guarantees as to its performance under any traffic conditions by proving it has a bounded competitive ratio. We also show that the competitive ratio of any algorithm for our problem might degrade linearly in the number of streams. Our simulation study shows that algorithms that follow our proposed design criteria exhibit a stable, 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 open questions including: (a) 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 in order to provide a better performance; (b) How to settle the gap in the competitive ratio of algorithms for the MFG problem; and (c) What is the tradeoff between fairness and goodput.

Acknowledgements

G. Scalosub's work was done while he was with the University of Toronto. The research in this paper is supported in part by a Strategic Project Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] J. M. Boyce and R. D. Gaglianello, "Packet loss effects on MPEG video sent over the public internet," in *ACM ICM*, 1998, pp. 181–190.
[2] 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.
[3] 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.
[4] A. Awad, M. W. McKinnon, and R. Sivakumar, "Goodput estimation for an access node buffer carrying correlated video traffic," in *IEEE ISCC*, 2002, pp. 120–125.
[5] 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 *ECUMN*, 2002, pp. 107–115.
[6] 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.
[7] 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.
[8] A. Kesselman, B. Patt-Shamir, and G. Scalosub, "Competitive buffer management with packet dependencies," in *IEEE IPDPS*, 2009.
[9] 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.
[10] 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.
[11] T. Nguyen and A. Zakhor, "Distributed video streaming with forward error correction," in *PV*, 2002.
[12] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.