Bounded Delay Scheduling with Packet Dependencies

Michael Markovitch and Gabriel Scalosub
Department of Communication Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
Email: markomic@post.bgu.ac.il, sgabriel@bgu.ac.il

***Abstract*—A common situation occurring when dealing with multimedia traffic is having large data frames fragmented into smaller IP packets, and having these packets sent independently through the network. For real-time multimedia traffic, dropping even few packets of a frame may render the entire frame useless. Such traffic is usually modeled as having *inter-packet dependencies*. We study the problem of scheduling traffic with such dependencies, where each packet has a deadline by which it should arrive at its destination. Such deadlines are common for real-time multimedia applications, and are derived from stringent delay constraints posed by the application. The figure of merit in such environments is maximizing the system's *goodput*, namely, the number of frames successfully delivered.**

We study deterministic online algorithms for the problem of maximizing goodput of delay-bounded traffic with inter-packet dependencies, and use competitive analysis to evaluate performance. We present a competitive algorithm for the problem, as well as matching lower bounds that are tight up to a constant factor. We further present the results of a simulation study which further validates our algorithmic approach and shows that insights arising from our analysis are indeed manifested in practice.

I. Introduction

A recent report studying the growth of real-time entertainment traffic in the Internet predicts that by 2018 approximately 66% of Internet traffic in North America will consist of real-time entertainment traffic, and most predominantly, video streaming [1]. Such traffic, especially as video definition increases, is characterized by having large application-level data frames being fragmented into smaller IP packets which are sent independently throughout the network. For stored-video one can rely on mechanisms built into various layers of the protocol stack (e.g., TCP) that ensure reliable data transfer. However, for real-time multimedia applications such as live IPTV and video conferencing, these mechanisms are not applicable due to the strict delay restrictions posed by the application (such traffic is therefore usually transmitted over UDP). These restrictions essentially imply that retransmission of lost packets is in most cases pointless, since retransmitted packets would arrive too late to be successfully decoded and used at the receiving end. Furthermore, the inability to decode an original dataframe once too many of its constituent packets have been dropped, essentially means that the resources used by the network to deliver those packets that did arrive successfully, have been wasted in vain. Since network elements make their decisions on a packet-level basis, and are unaware of such dependencies occurring between packets corresponding to the same frame, such utilization inefficiencies can be quite common, as also demonstrated in experimental studies [2].

Some of the most common methods employed to deal with the hazardous effect of packet loss in such scenarios focus on trading bandwidth for packet loss; The sender encodes the data frames while adding significant redundancy to the outgoing packet stream, an approach commonly known as forward error correction (FEC). This allows the user to mitigate the effect of packet loss, at the cost of increasing the rate at which traffic is transmitted. This makes it possible (in some cases) to decode the data frame even if some of its constituent packets are dropped. However, increasing the bandwidth can be costly, especially in various scenarios such as wireless access networks, network transcoders, and CDN headends. Therefore it is important to strike a balance between redundancy and the resulting overhead.

In this work we study mechanisms and algorithms that are to be implemented within the network, targeted at optimizing the usage of network resources (namely, buffer space and link bandwidth). We consider traffic that is *delay-sensitive*, exhibiting *inter-packet dependencies*, and our goal is to maximize the number of frames which can be successfully decoded.

Previous models presenting solutions for packet dependencies focused on managing a bounded-buffer FIFO queue, and mainly addressed the questions of handling buffer overflows (see more details in Section subsection I-B). We consider a significantly different model where each arriving packet has a *deadline* (which may or may not be induced by a deadline imposed on the data frame to which it corresponds). We assume no bound on the available buffer space, but are required to maximize the system's *goodput*, namely, the number of frames for which all of their packets are delivered by their deadline.[1] This model better captures the nature of real-time video streaming, where a data frame must be successfully decoded in *real-time*, based on some permissible deadline by which packets should arrive, that still renders the stream usable. We note that our model and results assume no FEC/redundancy in the underlying data stream and we show that even this simplified scenario poses significant difficulties. We believe that better understanding these settings (and the dilemmas

[1]It should be noted that the objective of maximizing goodput (on the frame-level) is in most cases significantly different than the common concept of maximizing *throughput* (on the packet-level).

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

that they entail) lay at the basis of further investigation of more complex settings which include both redundancy, higher-level depedencies (e.g., inter-frame), and network wide effects. These aspects and extensions are to be pursued in future work.

We consider traffic as being *burst-bounded*, i.e., there is an upper bound on the number of packets arriving in a time-slot. This assumption does not restrict the applicability of our algorithms, since it is common for traffic (and especially traffic with stringent Quality-of-Service requirements) to be regulated by some token-bucket envelope [3].

We focus on the impact of algorithmic choices, and use competitive analysis to show how these choices can bring us closer to an optimal solution. This approach makes our results globally applicable, and independent of the specific process generating the traffic. We further present the results of a simulation study which better highlights the impact of the various parameters, and gives further insight into the algorithmic design.

Due to space constraints many of the proofs are omitted, and can be found in [4].

A. System Model

We consider a time-slotted system where traffic consists of a sequence of unit-size *packets*, $p_1, p_2, \ldots$, such that packets are logically partitioned into *frames*. Each frame f corresponds to k of the packets, $p_1^f, \ldots, p_k^f \in \{p_1, p_2, \ldots\}$, where we refer to packet p_ℓ^f as the *ℓ-packet* of frame f. For every packet p we denote its arrival time by $a(p)$, and we assume that the arrival of packets corresponding to frame f satisfies $a(p_\ell^f) \leq a(p_{\ell+1}^f)$ for all $\ell = 1, \ldots, k-1$. In general we assume packets from the same source arrive at the order they were created. Each packet p is also characterized by a *deadline*, denoted $e(p)$, by which it should be scheduled for delivery, or else the packet *expires*. We assume $e(p) \geq a(p)$ for every packet p, and define the *slack* of packet p to be $r(p) = e(p) - a(p)$. For every time t and packet p for which $t \in [a(p), e(p)]$, if p has not yet been delivered by t, we say p is *pending at t*. we further define its *residual slack at t* to be $r_t(p) = e(p) - t$.[2]

We refer to an arrival sequence as being *d-uniform* if for every packet p in the sequence we have $r(p) = d$. We assume that $k \leq d$, which implies that any arriving frame can potentially be successfully delivered. We further let b denote the *maximum burst size*, i.e., for every time t, the number of packets arriving at t is at most b.

The packets arrive at a queue residing at the tail of a link with unit capacity. The queue is assumed to be empty before the first packet arrival. In each time-slot t we have three substeps: (i) *the arrival substep*, new packets arrive at time t and are stored in the queue, (ii) *the scheduling/delivery substep*, at most one packet from the queue is scheduled for delivery, and (iii) *the cleanup substep*, where every packet p currently in the queue which can not be scheduled by its deadline is discarded from the queue, either because $r_t(p) = 0$, or because it belongs to a frame which has multiple pending packets at time t and it is not feasible to schedule at least one of them by its deadline. Such packets are also said to expire at time t.

For every frame f and every time t, if f is not yet successful, but all of its packets that have arrived by t are either pending or have been delivered, then f is said to be *alive at t*. Otherwise it is said to have *expired*. A frame is said to be *successful* if each of its packets is delivered (by their deadlines).

The *Bounded-Delay Goodput problem* (BDG) is defined as the problem of maximizing the number of successful frames. When traffic is d-uniform, we refer to the problem as the *d-uniform* BDG *problem* (d-UBDG).

The main focus of our work is to study and develop algorithms for the BDG problem. An algorithm is said to be *online* if at any point in time t the algorithm knows only of arrivals that have occurred up to t, and has no information about future arrivals. We employ competitive analysis [6] to bound the performance of the algorithms. We say an online algorithm ALG is c-competitive (for $c \geq 1$) if for every finite arrival sequence it produces a solution who's goodput is at least a $1/c$ fraction from the optimal goodput possible. c is then said to be an upper bound on the *competitive ratio* of ALG. For completeness, we also address the *offline* problem where the entire arrival sequence is given in advance. In such offline settings the goal is to study the *approximation ratio* guaranteed by an algorithm, namely, what fraction of the optimal goodput is the algorithm guaranteed to deliver.

B. Previous Work

The effect of packet-level decisions on the successful delivery of large data-frames has been studied extensively in the past decades. Most of these works considered FIFO queues with bounded buffers and focused on discard decisions made upon overflows [7], as well as more specific aspects relating to video streams [8], [9]. This research thrust was accompanied by theoretical work trying to understand the performance of buffer management algorithms and scheduling paradigms, where the underlying architecture of the systems employed FIFO queues with bounded buffers. The main focus of these works was the design of competitive algorithms in an attempt to optimize some figure of merit, usually derived from Quality-of-Service objectives (see [10] for a survey). However, most of the works within this domain assumed the underlying packets are independent of each other, and disregarded any possible structure governing the generation of traffic.

Recently, a new model dealing with packet dependencies was suggested in [11]. The main focus of this work was buffer management of a single FIFO queue equipped with a buffer of size d, and the algorithmic question was how to handle buffer overflows. In what follows we refer to this problem as the *d-bounded FIFO problem* (d-BFIFO). Following this work, a series of works studied algorithms for various variants of the problem [12]–[15]. Our model differs significantly from these studies since in our model we assume no bounds on the available buffer size (as is more common in queueing theory

[2]Note that this is a tad different from the model used in [5] since we allow a packet to be scheduled also at time $e(p) = a(p) + r(p)$.

models), nor do we assume the scheduler conforms with a FIFO discipline.

Another vast body of related work focuses on issues of scheduling, and scheduling in packet networks in particular, in scenarios where packets have deadlines. Earliest-Deadline-First scheduling was studied in various contexts, including OS process scheduling [16], and more generally in the OR community [17]. Our framework is most closely related to [5] which considers a packet stream where each packet has a deadline as well as a weight, and the goal is to maximizing the weight of packets delivered by their deadline. Additional works provided improved competitive online algorithms for this problem (e.g. [18], [19]). However, none of these works considered the settings of packet-dependencies, which is the main focus of our work.

II. The Offline Settings

In order to study the d-UBDG problem in the offline settings, it is instructive to consider the d-BFIFO problem studied in [11] of managing a FIFO queue with buffer capacity d, where the goal is to maximize the number of successfully delivered frames (that none of their packets were dropped due to buffer overflows).

In what follows we first prove that these two problems are equivalent in the offline settings (proof omitted).

Lemma 1. *For any arrival sequence σ, a set of frames F constitutes a feasible solution to the d-UBDG problem if and only if it is a feasible solution to the d-BFIFO problem.*

Note that in particular, Lemma 1 implies that a set of frames F is optimal for d-UBDG if and only if it is optimal for d-BFIFO. By using the results of [11] for the d-BFIFO problem we obtain the following corollaries:

Corollary 2. *It is NP-hard to approximate the* BDG *problem to within a factor of $o(\frac{k}{\ln k})$ for $k \geq 3$, even for 0-uniform instances.*

Corollary 3. *There is a deterministic $(k+1)$-approximation algorithm for the d-UBDG problem.*

III. The Online Settings

The offline settings studied in section II, and the relation between the d-UBDG problem and the d-BFIFO problem, give rise to the question of whether one should expect a similar relation to be manifested in the online settings. In this section we answer this question in the negative.

A first fundamental difference is due to the fact that in the d-UBDG problem the scheduler is not forced to follow a FIFO discipline. This means that the inherent delay of packets stored in the back of the queue which occurs in a FIFO buffer (unless packets are discarded upfront) can be circumvented by the scheduler in the d-UBDG problem, allowing it to take priorities into account. Another significant difference between the two problems is that while in the d-BFIFO problem discard decisions in case of buffer overflow must be made *immediately* upon overflow, in the d-UBDG problem such decisions can be somewhat delayed. Intuitively, the online algorithm in the d-UBDG problem has more time to study the arrivals in the near future, before making a scheduling decision, and thus enable it to make somewhat better decisions, albeit myopic. We note that this view is also used in [18], [19] in the concepts of provisional schedules and suppressed packets (we give more details of these features in subsection IV-B).

A. Lower Bounds

In this section we provide several lower bounds for various ranges of our systems parameters. The main theorem is the following (proof omitted):

Theorem 4. *Any algorithm for the d-UBDG and d-BFIFO problems with burst size $b > 1$ has a competitive ratio $\Omega(b^{k-1})$.*

Our lower bounds can be adapted to token-bucket regulated traffic, with maximum burst size b and average rate r. Such restrictions on the traffic are quite common in SLAs. Of special interest is the case where the average rate is $r = 1$, which essentially means the link is not oversubscribed. Even for such highly regulated traffic, we have the following lower bound (proof omitted):

Theorem 5. *For token-bucket regulated traffic with parameters $(b,\ r = 1)$, any algorithm for the d-UBDG problem where $b \geq 2d$ has competitive ratio $\Omega(\left(\frac{b}{d}\right)^{k-1})$.*

This lower bound shows that the application of an access control mechanism does not in itself suffice for providing a goodput guarantee, thus emphasizing the need to study the underlying scheduling problem and address the algorithmic dilemmas it entails.

B. Upper Bounds

There are two natural criteria which can be used to decide what packet to schedule at a given time slot: (i) give priority to "important" packets, i.e., a packet which brings us the closest to the complete delivery of a frame, or (ii) give priority to "urgent" packets, i.e., a packet with the smallest residual slack.

For the case of d-uniform traffic, a FIFO scheduler implicitly opts for the latter criteria. By allowing a non-FIFO scheduling regime one obtains many more degrees of freedom which can potentially be exploited in an attempt to maximize the system's goodput. This lays at the core of the algorithms proposed in the sequel.

For every time t and frame f that has pending packets at t, let $I_t(f)$ denote the index of the first pending packet of f. Recall that by our assumption on the order of packets within a frame, this is the minimal index of a pending packet corresponding to f. We consider at every time t all pending frames as ordered in decreasing order $(I_t(f)$. In what follows we slightly abuse notation and refer to a frame as alive as long as none of its packets has expired nor was dropped, and it is possible to schedule all of its packets currently in the buffer before their deadlines. Our proposed algorithm, GREEDY is described in Algorithm 1.

Algorithm 1 GREEDY: at the scheduling substep of time t

1: drop all pending packets of frames that are not alive
2: $Q_t \leftarrow$ all alive frames with pending packets at t
3: $f \leftarrow \arg\max \{I_t(f') \mid f' \in Q_t\}$
4: deliver the first pending packet of f

The following theorem, for which we give merely a proof sketch here, is the main result in this section (as mentioned earlier, the complete proof can be found in [4]).

Theorem 6. *Algorithm* GREEDY *is* $O(b^{k-1})$*-competitive.*

Proof sketch: The analysis is based on partitioning the set of frames successfully delivered by some optimal solution into sets, such that every such set A_f can be mapped via a bijection to a frame f successfully delivered by GREEDY and showing that every such A_f is of size $O(b^{k-1})$. More specifically, we construct a dynamic mapping of arriving frames onto frames that have packets delivered by GREEDY. In particular, these frames "piggyback" the frames of the optimal solution. In case a frame is dropped by GREEDY its piggybacked frames are remapped onto another frame that is still alive at that time. This dynamic mapping eventually results in a set of (kb)-ary trees of height k, each rooted at some frame delivered by GREEDY, and each frame in the arrival sequence corresponds to a node in one of the trees. A closer examination of the tree and the mapping procedure allows us to significantly prune the tree and keep only frames that were successfully delivered by the optimal solution. A combinatorial argument then shows that the number of such frames in any such tree is at most $O(b^k)$ (as opposed to the overall size of the tree which is $(bk)^k$). This bound is achieved purely by considering the priority given to "important" packets. An improved bound of $O(b^{k-1})$ can be obtained by a closer look at the effect of deadlines on the performance of our algorithm as well as that of an optimal solution. In particular, this is facilitated by noting that our algorithm keeps frames alive for as long as possible. ∎

We note that our lower bounds indicate that GREEDY is asymptotically optimal. Furthermore it should be noted that GREEDY does not necessarily deliver packets in FIFO order, and therefore could not be implemented with a FIFO queue.

IV. FURTHER ALGORITHMIC CONSIDERATIONS

A. Tie-breaking

The results from the analysis above shows that an algorithm should prefer frames which are closer to completion (since this characteristic ensures competitiveness), and that live frames should be kept in the buffer for as long as possible (which is used to show the asymptotically tight performance guarantee). However, the GREEDY algorithm did does not make an explicit use of the deadline information in making scheduling decisions; the only effect of deadlines is on packets' expiry. This gives rise to the question of how to best use the deadline characteristics. One intuitive way to make use of this information is for performing tie-breaking between frames of with equal $I_t(f)$ value (instead of arbitrary tie-breaking).

A natural choice for a tie-breaker is the residual slack $r_t(p^f)$ of the smallest-index packet $p^f \in Q_t \cap f$ for each frame f that has the maximal $I_t(f)$ value. The purpose of such a tie breaker is of course to improve performance by keeping as many frames alive as possible, which emphasizes the characteristics of GREEDY that allow it to match the lower bound.

One should note that the tie-breaking rule does not affect the asymptotic competitiveness of GREEDY, which is tight up to a constant factor. However, this it is expected to heavily influence the performance of the algorithms in practice. This is further discussed and validated in section V,

B. Scheduling

For the GREEDY algorithm presented in subsection III-B, the first packet of the preferred frame was sent, where the difference between the algorithms boiled down to the way other pending packets were treated. In particular, the residual slack of the packets is essentially ignored by these greedy approaches (although it can be taken into account in tie-breaking, as discussed above).

One common approach to incorporate residual slack into the scheduler is considering *provisional schedules*, which essentially try pick the packet to be delivered using a local offline algorithm, which takes into account all currently available information. Such an approach can be viewed as aiming to maximize the benefit to be accrued from the present packets, assuming no future arrivals. Such an approach lays at the core of the solutions proposed by [5], [18], [19] which each used an algorithm for computing an optimal offline local solution. In our case, as shown in Corollary 2, computing such an optimal provisional schedule is hard, but, as shown in Corollary 3, there exists a $(k+1)$-approximation algorithm for the problem.

We adapt this algorithm into a procedure for computing a provisional schedule, which would allow a smaller I-indexed frame to have one of its packets scheduled, only if non of the frames with a higher I-index would become infeasible in the following time slot. Our proposed heuristic, OPPORTUNISTIC, is described in Algorithm 2. OPPORTUNISTIC builds a provisional schedule F_t as follows:

1) Sort pending frames[3] in decreasing lexicographical order of $(I_t(f), d - r_t(f))$. I.e., preference is given to frames with higher I-index values. In case of ties, preference is given to the frame that has a pending packet with the minimal residual slack.
2) Initialize the provisional schedule $F_t = \emptyset$.
3) For each frame f in this order, test whether for all $s = 0, \ldots, d$, the pending packets of f can be added to F_t such that the overall number of packets in the provisional schedule with remaining slack at most s, does not exceed s. If f can be added, update $F_t = F_t \cup \{f\}$.

V. SIMULATIONS

We now turn to present our simulation results studying the performance of the algorithms, as well as the impact of the parameters k and d on the performance.

[3]A frame is pending if it has pending packets.

Algorithm 2 OPPORTUNISTIC: at the scheduling substep of time t

1: Build the provisional schedule F_t
2: transmit the packet with minimum residual slack in F_t

A. Traffic Generation and Setup

We recall that the problem of managing traffic with packet dependencies captured by our model is most prevalent in real-time video streams. We therefore perform a simulation study that aims to capture many of the characteristics of such streams.

We generate traffic which is an interleaving of *streams*, where each stream is targeted at a different receiver, and all streams require service from a single queue at the tail of a link. Specifically, we focus on traffic with the following characteristics. (i) We assume each stream has a random start time where packets are generated. This corresponds to scenarios of multiple independent streams. (ii) We assume the average bandwidth demand of all streams is identical. (iii) Frames of a single stream are non-overlapping, and are produced by the source in evenly spaced intervals. E.g., if we consider video streams consisting of 30FPS, each interval is $33ms$. (iv) The source transmits the packets of each frame in a burst, and we assume each frame consists of the same number of packets. Such a scenario occurs, e.g., in MPEG encoding making use of I-frames alone. We assume all packet have the same size, namely, the network's MTU. (v) We assume a random delay variation between the arrival of consecutive packets corresponding to the same stream (possibly introduced upstream). Specifically, we assume a uniform delay variation of up to 5 time slots. (vi) We assume each packet contains the frame number, and the index number of the packet within the frame. Such information can be encoded, e.g., in the RTP header.

For the setup we chose to simulate, the packet sizes are set such that every time slot one packet can be scheduled, and the aggregate bandwidth of all the streams is equal to the service bandwidth. We note that even in such cases, no online algorithm is guaranteed to obtain the optimal goodput.

We simulate 2 minutes worth of traffic for 50 streams, where for all the streams the frame rate is 30FPS (for a total of 3600 frames per stream). Since we fix the service rate as 1, in the simulation the interval between consecutive frames arrival in a stream is $\Delta F = k \cdot 50$ time slots, where k is the number of packets per frame. As k grows the "real" duration of a single time slot decreases, as the service rate effectively increases.

B. Simulated Algorithms

We performed the simulation study for four scheduling algorithms, where in all algorithms in case of ties in the priorities, these are broken according to the a random (but fixed) priority on the streams: (i) The offline $O(k+1)$-approximation algorithm of [11]. This algorithm serves as a benchmark for evaluating the performance of the online algorithms. (ii) Algorithm GREEDY, described and analysed in subsection III-B. This algorithm represents our baseline for studying the the performance of online algorithms for the problem. (iii) Algorithm $\text{GREEDY}_{\text{SLACK}}$, which implements GREEDY with ties broken according to minimum residual slack, as discussed in subsection IV-A. (iv) Algorithm OPPORTUNISTIC, based on provisional schedules, presented in subsection IV-B.

C. Results

The simulation results confirm our hypothesis that implementation of the proposed algorithm design guideline does indeed impact the performance of online algorithms. We depict the performance of each online algorithm by its *goodput ratio*, measured by the ratio between the goodput of the online algorithm and that of the offline algorithm.

Figure 1 presents the performance of the online algorithms as a function of the slack each packet has, for $k = 12$. It can be seen that as the slack increases the tie-breaking rule in $\text{GREEDY}_{\text{SLACK}}$ shows significant improved performance in comparison with the vanilla greedy algorithm. The figure also shows that the OPPORTUNISTIC exhibits a better performance than $\text{GREEDY}_{\text{SLACK}}$ (although this improvement is paid for by significant additional complexity). We note that results for greater values of k exhibit the same trends. Also of note is that the $\text{GREEDY}_{\text{SLACK}}$ and OPPORTUNISTIC manage to trace the performance of the offline algorithm (and actually complete all the frames of all the streams) for traffic with sufficiently large slack.

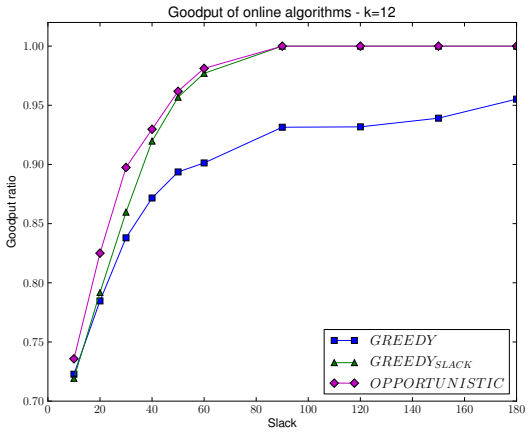
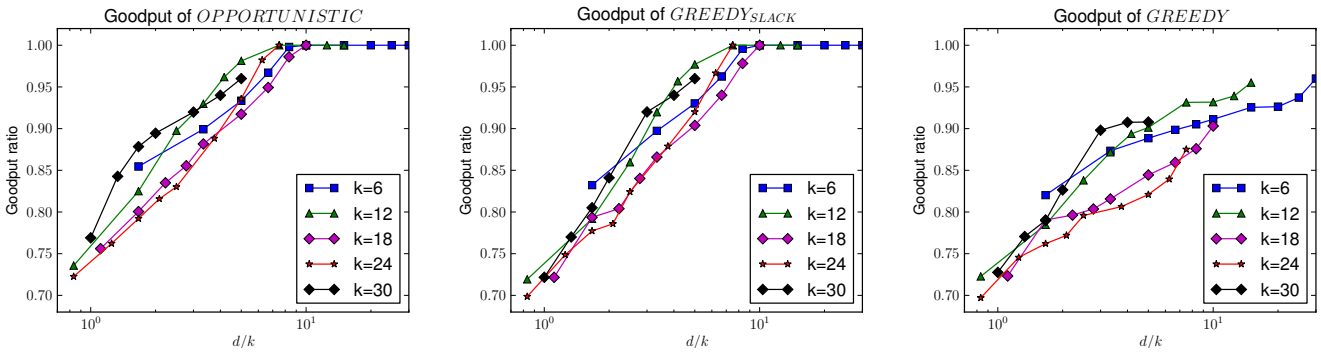


Fig. 1: Comparison of the goodput of online algorithms

In Figure 2 we present the goodput ratio of OPPORTUNISTIC and $\text{GREEDY}_{\text{SLACK}}$ as a function of d/k, where different values of k correspond to different (independent) simulations. The first lesson learnt from this data is that the performance of the opportunistic algorithm is superior to that of the enhanced greedy algorithm, in particular for small d/k values where the difference becomes more pronounced (these results are also hinted by Figure 1, but are not as pronounced). Furthermore, the performance of both algorithms depends exponentially on the ratio between d and k (shown by the log scale), as even though both graphs present results of many simulations



(a) OPPORTUNISTIC (b) GREEDY$_{\text{SLACK}}$ (c) GREEDY

Fig. 2: Goodput of the online algorithms as a function of d/k on a logarithmic scale

with different inputs having different parameters, the plots show a linear trend up to the point where they match the goodput of the offline algorithm. We note that this exponential dependency on d/k highlighted by our simulation results is obfuscated by our worst-case analysis, which merely shows exponential dependency in $1/k$. This gives further insight into the performance of our proposed algorithm.

VI. Conclusions and Future Work

In this paper we address the problem of maximizing the goodput of delay sensitive traffic with inter-packet dependencies. We provide lower bounds on the competitiveness of deterministic online algorithms for the general case that the traffic is burst bounded, and present competitive scheduling algorithms for the problem. Our goal is to study the impact of algorithmic considerations, and through the analysis we show that there exists an algorithmic guideline that ensures competitiveness, namely, giving preference to frames that are closer to completion. We provide an algorithm that ensure the optimal performance possible, up to a small constant factor.

Our analysis further provides insights into improving the performance of online algorithms for the problem. These insights are further verified by a simulation study which shows that our improved algorithms which are inspired by our analytic results, are very close to the performance of the currently best known offline algorithm for the problem. More specifically, the performance of our algorithms approach the performance of our benchmark algorithm with an exponential correlation to the increase in delay-slack.

This work serves as an initial study of scheduling delay-bounded traffic with inter-packet dependencies and raises new questions about the performance of algorithms for this problem, including (i) how one should deal with non-uniform deadlines, (ii) what are the algorithmic principles that should be employed in the face of redundancy, and (iii) what are network wide effects of such an algorithmic approach.

References

[1] Sandvine, "Global Internet phenomena report – 1H 2013," http://www.sandvine.com/, July 2013.

[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] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Addison-Wesley, 2011.

[4] M. Markovitch and G. Scalosub, "Bounded delay scheduling with packet dependencies," December 2013. [Online]. Available: http://arxiv.org/pdf/1402.6973.pdf

[5] 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.

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

[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. 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.

[9] 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.

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

[11] A. Kesselman, B. Patt-Shamir, and G. Scalosub, "Competitive buffer management with packet dependencies," *Theoretical Computer Science*, vol. 489–490, pp. 75–87, 2013.

[12] Y. Emek, M. M. Halldórsson, Y. Mansour, B. Patt-Shamir, J. Radhakrishnan, and D. Rawitz, "Online set packing," *SIAM Journal on Computing*, vol. 41, no. 4, pp. 728–746, 2012.

[13] Y. Mansour, B. Patt-Shamir, and D. Rawitz, "Competitive router scheduling with structured data," in *Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA)*, 2011.

[14] ——, "Overflow management with multipart packets," *Computer Networks*, vol. 56, no. 15, pp. 3456–3467, 2012.

[15] G. Scalosub, P. Marbach, and J. Liebeherr, "Buffer management for aggregated streaming data with packet dependencies," *IEEE Transacations on Parallel and Distributed Systems*, vol. 24, no. 3, pp. 439–449, 2013.

[16] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. John Wiley & Sons, 2012.

[17] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.

[18] M. Englert and M. Westermann, "Considering suppressed packets improves buffer management in quality of service switches," *SIAM Journal on Computing*, vol. 41, no. 5, pp. 1166–1192, 2012.

[19] L. Jez, F. Li, J. Sethuraman, and C. Stein, "Online scheduling of packets with agreeable deadlines," *ACM Transactions on Algorithms*, vol. 9, no. 1, 2012.