institute IMDEA networks

technical report

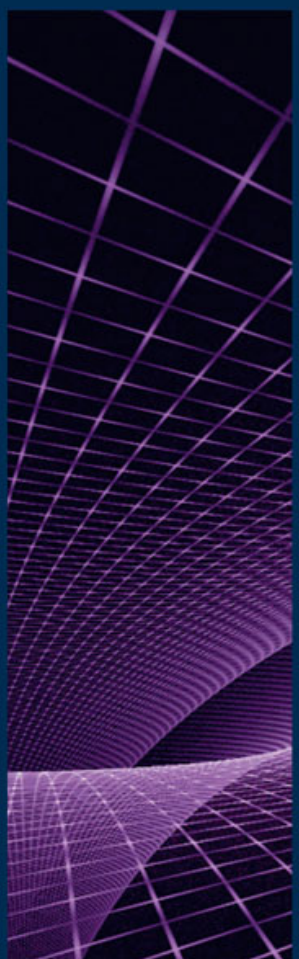
TR-IMDEA-Networks-2015-1

Anticipatory Quality-Resource Allocation for Multi-User Mobile Video Streaming

Nicola Bui

Stefan Valentin

Joerg Widmer



March 2015

Anticipatory Quality-Resource Allocation for Multi-User Mobile Video Streaming

Nicola Bui[12], Stefan Valentin[3] and Joerg Widmer[1]
[1]IMDEA Networks Institute, Leganes (Madrid), Spain
[2]UC3M, Leganes (Madrid), Spain
[3]Mathematical and Algorithmic Sciences Lab, FRC, Huawei Technologies, France

***Abstract*—Mobile video delivery forms the largest part of the traffic in cellular networks. Thus optimizing the resource allocation to satisfy a user's quality of experience is becoming paramount in modern communications. This paper belongs to the line of research known as anticipatory networking that makes use of prediction of wireless capacity to improve communication performance. In particular, we focus on the problem of optimal resource allocation for steady video delivery under maximum average quality constraints for multiple users. We formulate the problem as a piecewise linear program and provide a heuristic algorithm, which solution is close to optimal. Based on our formulation we are now able to trade off minimum video quality, average quality and offered network capacity.**

I. Introduction

Many reports confirm the recent trends in Internet traffic evolution: overall mobile traffic increased by 81% compared to 2013 according to the latest CISCO statistics [1], while Sandvine [2] states that real-time entertainment is having the biggest share in this growth (63%) and services such as NetFlix and YouTube occupy the first positions in the traffic generation list. At the same time, mobile networks are becoming increasingly constrained by limited spectral resources [3]. We believe that, in addition to solutions aimed to increase the wireless data rate, the available resources can be used much more efficiently.

In the literature, many works demonstrated that common trends and patterns can be used to anticipate traffic demands by predicting the future capacity in a network: in particular [4] highlights how network dynamics [5] can be understood, predicted and linked to human mobility patterns [6] while in [7] the authors devise to estimate which content is going to be requested in the future.

Utilizing such sophisticated models to build predictors and using the resulting prediction for efficient resource allocation is the objective of *anticipatory networking* .This work contributes to the area by investigating resource allocation efficiency when video delivery is the main traffic source in the network. Without being exhaustive, the following provides a brief overview of some recent works in that area.

The resource allocation problem for a discrete set of video qualities has been investigated in [8] where it has been modeled as a Mixed Integer Quadratically Constrained Program (MIQCP). Similarly, [9] studied resource allocation for uninterrupted video streaming devising a single-user optimal resource allocation algorithm. [10], [11] devised a solution using a Mixed Integer Linear Program to maximize throughput and minimize energy consumption respectively. Finally, [12] among a few others relaxed the assumption of perfect knowledge by accounting for prediction reliability and errors. Following the same direction, we recently presented a general model for mobile user capacity prediction [13].

In this paper, we study the network resource allocation problem aimed to minimize the average video re-buffering time (so called *lateness*) and, provided this objective is achieved, maximizing the average video bitrate for multiple users. Our paper differs from those mentioned above in the following aspects. We address video bitrate as a continuous quantity as in current streaming techniques, such as HTTP Live Streaming (HLS) [14], different segments can be encoded with different bitrates. Consequently, the average bitrate computed over multiple segments can take any value between the minimum and the maximum encoding bitrate.

Considering the video bitrate as a continuous quantity allows us to formulate the problem as a piecewise linear program (LP). In this LP, we i) minimize the aggregate lateness among all users when they are provided with the minimum allowed video bitrate and ii) the remaining resources are used to maximize the aggregate video encoding bitrate.

To solve this problem with a fast heuristic, we develop the Split, Sort & Swap (SS&S) algorithm. it achieves solutions very close to the optimum faster than standard solvers [15] for many relevant cases. In our evaluation of a 3GPP compliant macrocell scenario, SS&S never falls 0.5% below the optimum.

This algorithm starts from a greedy solution obtained in polynomial time, which is subsequently refined by means of an iterative process. In each iteration, the algorithm provides a feasible solution that represents an improvement to the previous step. This property allows to trade-off algorithm runtime with quality gain.

We believe that this practical trade-off and the ability to maximize quality while a minimum lateness is guaranteed, make our algorithm a promising candidate for a dedicated media streaming mode in fifth generation cellular networks. The algorithms low complexity supports online adaptation for a large number of users with a long prediction horizon.

In the following Section II we define the system model and discuss the optimization problem. In Section III we describe the SS&S algorithm, and we analyze its performance in Section IV. Section V provides our conclusions.

This work was supported by the PhD@Bell Labs Internship program. The majority of the work was performed while the first two authors were with the Alcatel-Lucent Bell Labs in Stuttgart, Germany.

II. Problem definition

In this paper we address resource allocation for the wireless downlink of a cellular network when future knowledge about the achievable data rate is available. To provide a simpler notation we will consider a system with a single base station to which all the K users connect. We call the set of users $\mathcal{U}$ and our prediction horizon is T time units and we refer to the set of time slots as $\mathcal{T}$. In the following, we consider unitary time unit $t = 1$, in order for data rates and download size to be used interchangeably. In the rest of the paper we use the following assumptions: 1) the future knowledge is perfect (this does not hold in practice, but the problem solution can be updated periodically. The present results can be considered as an upper bound for real scenarios); 2) the average video bitrate is continuous between 0 and q_M (e.g.: by combining segments of different quality) and 3) the quality of experience is proportional to the video bitrate.

The quantities of interest are:

• Per user achievable download rate $R = \{r_{i,j} \in [0, r_M],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where the entry $r_{i,j}$ represents the average rate that user i would achieve in slot j if he was using the cell alone. r_M is the maximum data rate of the specific mobile technology. This represents the future knowledge about network conditions, where slot 1 is the present slot.

• Minimum requirements $D = \{d_{i,j} \in [0, q_M],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where $d_{i,j}$ is the minimum amount of bytes user i should receive before the end of slot j in order to stream the video at the minimum allowed quality. If at any time the user receives more data than required, the excess can be stored in a buffer for later use.

• Assigned resources $A = \{a_{i,j} \in [0, 1],\ i \in \mathcal{U}, j \in \mathcal{T}\}$: each entry $a_{i,j}$ represents the average fraction of resources assigned to user i in slot j. In each slot, no user can be assigned more than the total available rate, $0 \leq a_{i,j} \leq 1$, nor can the sum of all the assignments exceed the total available resources in that slot, $0 \leq \sum_{i \in \mathcal{U}} a_{i,j} \leq 1$.

• Maximum extra video bitrate $U = \{u_{i,j} \in [0, q_M],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where $u_{i,j}$ is the additional bitrate user i could download before the end of slot j to increment the video quality.

• Assigned resources for extra quality $Q = \{q_{i,j} \in [0, 1],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where $0 \leq q_{i,j} \leq 1 - \sum_{k \in \mathcal{U}} a_{u,j}$ is the fraction of the available resources $r_{i,j}$ to be allocated for extra quality.

• Buffer state $B' = \{b'_{i,j} \in [0, b_M],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where

$$b'_{i,j+1} = [b'_{i,j} + a_{i,j} r_{i,j} - d_{i,j}]_0^{b_M} \tag{1}$$

is the buffer level of user i at the end of slot $j + 1$, b_M is the buffer size in bytes and $[\cdot]_a^b = \min\{\max\{\cdot, a\}, b\}$ is a bounding operator.

• Extra quality buffer state $B'' = \{b''_{i,j} \in [0, b''_{\max}],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where

$$b''_{i,j+1} = [b''_{i,j} + q_{i,j} r_{i,j} - u_{i,j}]_0^{b_M - b'_{i,j+1}} \tag{2}$$

is the buffered data to be used for extra quality. The total buffer is $B = B' + B''$.

• Lateness $L = \{l_{i,j} \in [0, 1],\ i \in \mathcal{U}, j \in \mathcal{T}\}$ is the fraction of slot j for which no data was available to stream the minimum video bitrate:

$$l_{i,j} = \begin{cases} [d_{i,j} - b_{i,j-1} - a_{i,j} r_{i,j}]_0 / d_{i,j} & d_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

• Extra quality $E = \{e_{i,j} \in [0, q_M],\ i \in \mathcal{U}, j \in \mathcal{T}\}$, where $e_{i,j}$ is the maximum number of bytes that user i receives for extra quality j,

$$e_{i,j} = [q_{i,j} r_{i,j} + b_{i,j-1}]^{u_{i,j}}. \tag{4}$$

We define the system average lateness λ, and total average quality θ as:

$$\lambda = \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{T}} l_{i,j} / (KT). \tag{5}$$

$$\theta = \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{T}} (e_{i,j} + a_{i,j} r_{i,j}) / T. \tag{6}$$

Thus we can define our optimization problem as computing two schedules A and Q the minimize lateness λ, and maximize quality θ, given C, D, U, and b_M as defined above. In this paper we assign a higher priority to lateness minimization so that under no circumstance the system is trading lateness for quality. Consequently we formulate the optimization problem as:

$$\begin{aligned} \underset{A,Q}{\text{minimize}} \quad & W\lambda - \theta \\ \text{subject to:} \quad & a_{i,j} \geq 0;\ \sum_{k \in \mathcal{U}} a_{k,j} \leq 1 \\ & q_{i,j} \geq 0;\ \sum_{k \in \mathcal{U}} q_{k,j} \leq 1 - \sum_{k \in \mathcal{U}} a_{k,j} \\ & \forall i \in \mathcal{U}; j \in \mathcal{T} \\ & \text{Eqns. } (1), (2), (3), (4), (5) \text{ and } (6), \end{aligned} \tag{7}$$

where Q and A are control variables, B', B'', L, E, λ and θ are additional variables and R, D, U and b_M are input parameters. The objective function is a linear combination of Eqns. (3) and (4) and the parameter W weights the two components. In particular, the solver has to use resources to either decrease the lateness or to increase the quality. Ideally, with $W \to \infty$ the solution of the problem would never choose quality over lateness, but in practice it is sufficient to have $W \gg \max\{\theta\}$.

In addition, since our objective function is a linear combination of piecewise linear equations, the overall optimization problem is piecewise linear as well. Unfortunately, while the formulation is quite compact, Eq. (1) that defines the evolution of the buffer state increases the overall complexity of the problem due to the bounding operator that limits the buffer between 0 and b_M in every slot.

III. Resource Allocation Algorithm

Our algorithm, Split, Sort & Swap (SS&S), is based on two main phases: first a greedy algorithm is used to obtain a feasible resource allocation (greedy phase), then the solution is iteratively improved (swap phase). The main idea is to i)

compute the minimum lateness allocation and ii) maximize quality as second priority. According to our formulation this is equivalent to addressing the two quantities together since any lateness increment is W times worse than a similar decrease in quality. Also, this prevents mixing quality (video bitrate) and lateness (time). Thus, in the following, we provide a description of the algorithm operations to minimize lateness and we only discuss how to adapt it for quality maximization.

The key ideas of the algorithm (from which we derived the name) are:

- **Split**: Consider the smallest number of slots.
- **Sort**: Use capacity in descending order.
- **Swap**: Change allocations only if it improves the objective.

The overall SS&S is given in Algorithm 1, using the notation from Section II and the following additional variables s_f and s_l that identify the first and the last slot the algorithm is considering at any given step. Variable s_l is increased whenever no more improvement to the objective function can be obtained to satisfy the requirements up to slot s_l. Variable s_f is increased if no improvement to the objective function can be obtained by changing the allocation earlier than s_f. Also, we define x and x_M as the current and the maximum allowed numbers of iterations of the greedy phase; λ_0 is the average lateness computed at the previous optimization iteration and δ_M is used to stop the greedy phase if the current improvement is smaller than that.

Algorithm 1 Split, Sort & Swap (SS&S)

Input: R, D, b_M.
Output: $[A, B] = \text{SS\&S}(R, D, b_M)$
  $s_f = 1, s_l = 1$ // *initial optimization window*
  $a_{i,j} = 0$, $b_{i,j} = 0$ $\forall i \in \mathcal{U}, j \in \mathcal{T}$
  while $s_l \leq T$ **do**
    $[A, B, s_f, s_l] = \text{Greedy}(R, D, s_f, s_l, b_M, A, B)$
  end while
  $s_f = 1$; $x = 0$; $\lambda_0 = 0$
  while $x < x_M$ AND $|\lambda - \lambda_0| < \delta_M$ **do**
    $x = x + 1$; $\lambda_0 = \lambda$
    $[A, B, s_f] = \text{Swap}(R, D, b_M, A, B, s_f)$
  end while
  return A, B

The following paragraphs review the algorithm's mechanics through a simple example, while its formal definition and the complete pseudocode is given shortly after.

Let's consider the following achievable rates R (compare the topmost plot in Fig. 1), minimum video requirements D and buffer size $b_M = 1$:

$$R = \begin{bmatrix} 2 & 0 & 3 & 0 \\ 1 & 1 & 4 & 1 \end{bmatrix} \quad (8)$$

$$D = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (9)$$

We start with the greedy procedure, the allocation A and lateness L of which are illustrated in the second and third plots of Fig. 1 respectively.

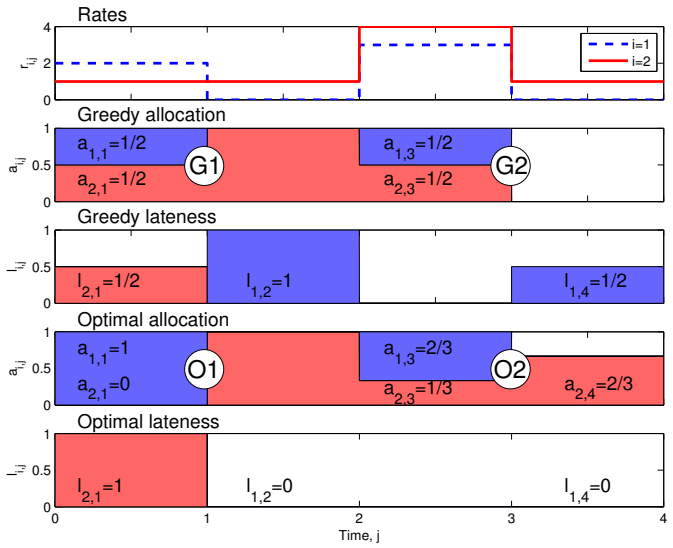


Fig. 1. Graphic example of simple SS&S operations: users' achievable rates, R are shown at the top, the allocation A and the lateness L after the greedy phase are plotted in the second and third plots from the top respectively. While the fourth and the fifth plot show the optimal A and L after the swap phase. The improvements from G1 to O1 and from G2 to O2 involve a Type 1 and a Type 2 swap respectively.

The first greedy allocation entails slot 1 only. Here the two users may obtain up to $r_{1,1} = 2$ and $r_{2,1} = 1$ respectively. The greedy procedure assigns $a_{1,1} = d_{1,1}/r_{1,1} = 1/2$ to satisfy slot 1 requirements. Then the remaining resources $a_{2,1} = 1 - a_{1,1} = 1/2$ are assigned to user 2. However, this is less than needed and causes a buffer under-run $l_{2,1} = d_{2,1} - a_{2,1}r_{2,1} = 1/2$. This under-run is unavoidable, due to the achievable rates in slot 1 and it obtains the minimum total lateness in slot 1 as increasing the resource allocation to user 2 would cause a larger lateness to user 1.

The greedy allocation in slot 2 is trivial for two reasons: slot 1 is fully allocated, thus no buffering is possible, and $r_{1,2} = 0$. Thus, we obtain $a_{1,2} = 0$, $a_{2,2} = 1$, $l_{1,2} = 1$ and $l_{2,2} = 0$ respectively. We annotated these two first slots as G1 in the figure and the total lateness so far is $\lambda = l_{2,1} + l_{1,2} = 3/2$.

The allocation is now complete until slot 2, since no other greedy allocations can occur before and including slot 2, thus the next optimization phase will have $s_f = s_l = 3$. We annotated the following two slots ($j = \{3, 4\}$) as G2 in Fig. 1. Again, the greedy operation starts by considering slot 3 alone by sorting users according to their achievable rates. Thus, slot 3 requirements are satisfied by allocating $a_{1,3} = 1/4$ and $a_{2,3} = 1/3$ and leaving $5/12$ of the resources free.

Now the algorithm accounts for slot 4 requirements by considering free resources both in slot 3 and slot 4 in order of decreasing capacity. The following greedy allocations are made: since $r_{1,3} = 4$ is the highest, $a_{1,3} = a_{1,3} + d_{1,4}/r_{1,3} = 1/4 + 1/4 = 1/2$ of which $1/4$ is used to fill the buffer $b_{1,3} = b_{1,2} + a_{1,3}r_{1,3} - d_{1,3} = 1$; the highest capacity for user 2 is in slot 3 too, but here user 2 can only be assigned $a_{2,3} = 1/3 + 1/6 = 1/2$ to buffer $b_{2,3} = 1/2$ and, since user 2

capacity in the last slot is $r_{2,4} = 0$, the greedy decision cannot avoid some lateness $l_{2,4} = d_{2,4} - b_{2,3} = 1/2$.

The final greedy allocation is obtained with a total lateness $\lambda = 2$ consisting of an unavoidable under-run in slot 1 for user 2, two under-runs for user 1 in slots 2 and 4 and all resources in the last slot left unassigned.

In the swap phase, the algorithm considers those slots where it was not possible to avoid some lateness by modifying the greedy allocation obtained so far. In our example, the first case to be addressed is user 1 in slot 2, where the algorithm obtained a lateness of $l_{1,2} = 1$. If we do not consider user 2, user 1 can fill the buffer in slot 1 to satisfy the requirements of slot 2. However, this cause more lateness for user 2 in slot 1. In particular, if we swap a quantity δ_a of resources from user 2 to user 1, we obtain that the total lateness varies of a proportional quantity $\delta_\lambda = \delta_a(r_{2,1} - r_{1,1})$ that is the sum of the increase of lateness $l_{2,1} = l_{2,1} + \delta_a r_{2,1}$ and the decrease of $l_{1,1} = l_{1,1} - \delta_a r_{1,1}$ due to the resource swap.

If $r_{1,1} > r_{2,1}$, then $\delta_\lambda < 0$ and the total lateness is decreasing. However, δ_a is limited by the minimum among $a_{2,1} = 1/2$ the resources allocated to the users we are removing resources from, $(b_M - b_{1,1})/r_{1,1} = 1/2$ the maximum resources that can be buffered by the receiving user and $l_{1,2}/r_{1,1} = 1/2$ the lateness we are trying to reduce. Since $\delta_\lambda = -\delta_a$, the maximum improvement is obtained for the maximum $\delta_a = 1/2$ and the following optimal allocation $a_{1,1} = 1$, $a_{1,2} = 0$, that allows user 1 to buffer $b_{1,1} = 1$ and avoid the under-run in slot 2 and increase the under-run for user 2 to $l_{2,1} = 1$. However, the total lateness of these first two slots is $\lambda = 1$ only: δ_λ less than that obtained by the greedy procedure. We annotate these two slots $j = \{1, 2\}$ as O1 in the figure.

The forth and the fifth plot in Fig. 1 show the optimal allocation and associated lateness obtained after the swap phase. We call **Type 1** the swap occurring in the first two slots from G1 to O1 and is characterized by replacing an under-run with a smaller one earlier in the sequence.

Before addressing the buffer under-run in slot 4, we note that in the two first slots all resources are given to the user with the highest achievable rate, thus further resource swapping can only make the total lateness worse. Also, applying the type 1 method to this is not worthy since $r_{1,3} < r_{2,3}$ and will lead to an increased lateness $l_{2,3} > l_{1,4}$. However, slot 4 has unused resources and it may be possible to use them to improve the earlier allocations.

In fact, since slot 4 requirements for user 2 are satisfied with buffered data, it is possible avoid buffering for user 2 and, instead, use the free resources in the last slot to satisfy $d_{2,4}$, while the remaining resources in slot 3 can be buffered for user 1 to satisfy $d_{1,4}$. More formally, we can swap a quantity of resource δ_a from user 2 to 1, which will cause a proportional lateness variation $\delta_\lambda = \delta_a(r_{2,3} - r_{1,3})$ and we can recover $\delta_a r_{2,3}$ by allocating $a_{2,4} = \delta_a r_{2,3}/r_{2,4}$.

Similar to a Type 1 swap, δ_a is limited by the minimum among $a_{2,3} = 1/2$, $(b_M - b_{1,3})/r_{1,3} = 1/3$ and $l_{1,4}/r_{1,3} = 1/3$ and, in addition, by the free resources in slot 4 $(1 - \sum_{i\in\mathcal{U}} a_{i,4})/r_{1,3} = 1/3$. Thus the optimal allocation becomes $a_{1,3} = 2/3$, $a_{2,3} = 1/3$, and $a_{2,4} = 2/3$. These last two slots are annotated as G2 in the figure and we call **Type 2** the resource swap between G2 and O2, which is defined by the recovery of an under-run by modifying earlier buffer state with the usage of later free resources.

More formally the greedy and the swap phases are given in Algorithm 2 and Algorithm 4 respectively. In Algorithm 2 we use the indicator function $\mathrm{I}(x) = 1$ if $x > 0$ and $\mathrm{I}(x) = 0$ otherwise. Also, while the greedy phase is deterministic and performs at most $O((KT)^2)$ iterations, the swap phase improves the objective functions iteratively.

For what concerns the greedy phase, Algorithm 2 checks whether further resources can be assigned between slot s_f and s_l by computing $\hat{A}$, which elements $\hat{a}_{i,j}$ represent the maximum usable rate for user i in slot j accounting for available resources in the slot $(1 - \sum_{k\in\mathcal{U}} a_{k,j})$, future requirements to be satisfied $(\sum_{k=j}^{s_l} d_{i,k})$ and available buffer space $(b_M - \max_{k\in[j,s_l]}\{b_{i,k}\} + d_{i,j})$ as follows:

$$\hat{a}_{i,j} = \min\Big\{(1 - \sum_{k\in\mathcal{U}} a_{k,j})r_{i,j},\ \sum_{k=j}^{s_l} d_{i,k},\ b_M - \max_{k\in[j,s_l]}\{b_{i,k}\} + d_{i,j}\Big\} \quad \forall\, i \in \mathcal{U}, s_f \le j \le s_l. \tag{10}$$

Algorithm 2 Greedy phase

Input: $R, D, s_f, s_l, b_M, A, B$.
Output: $[A, B, s_f, s_l] = \mathrm{Greedy}(R, D, s_f, s_l, b_M, A, B)$
compute $\hat{A}$ as per Eq. (10) *// feasible resource usage*
while $\sum_{i\in\mathcal{U}} \sum_{j=s_f}^{s_l} > 0$ **do**
  $k, l = \underset{i,j}{\mathrm{argmax}}\ r_{i,j}\mathrm{I}(\hat{a}_{i,j})$ *// choose best user and slot*
  $a_{k,l} = a_{k,l} + \hat{a}_{k,l}/r_{k,l}$ *// increase allocation*
  $b_{k,m} = b_{k,m} + \hat{a}_{k,l}\ \forall\, l \le m < s_l$ *// adjust buffer*
end while
$s_l = s_l + 1$
$s_f = \mathrm{NewStart}(A, B, R, b_M, s_f, s_l)$
return A, B, s_f, s_l

Then, among all users and slots to whom resources can be assigned ($\hat{a}_{i,j} > 0$) the user k with the highest rate in slot l is assigned further resources, so that $a_{k,l} = a_{k,l} + \hat{a}_{k,l}/r_{k,l}$. One greedy phase step continues until either no new resources can be assigned or all requirements up to slot s_l are satisfied.

Finally, s_l is increased and the NewStart procedure (see Algorithm 3) updates s_f if needed. In particular, a slot s_f is completed if it is not possible to swap resources between users either because that would cause a buffer overflow or because that would degrade the objective function.

The formal definition of the swap phase requires first to generalize the two type of resource swapping that, in turn, requires to identify the best possible swap which is improving the objective function the most by increasing the resource allocation the least.

According to type 1 swap, the best swap is the one obtaining the highest δ_λ. To this extent, we first define $\delta_{\lambda,(i,j)}$, and $\delta_{a,(i,j)}$ as the best lateness improvement and the maximum

Algorithm 3 New Start

Input: A, B, R, b_M, s_f, s_l.
Output: $s_f = \text{NewStart}(A, B, R, b_M, s_f, s_l)$
  for $j \in [s_f, s_l]$ **do**
    if $\min\{b_M - \max_{l\in[j,s_l]} b_{i,l},\ \sum_{k\in\mathcal{U}|r_{k,j}<r_{i,j}} a_{k,j}\} = 0$ $\forall\, j \in \mathcal{U}$ **then**
      $s_f = s_f + 1$
    else
      return s_f
    end if
  end for
  return s_f

exchangeable resources to move an under-run in (i, j) to $n_{i,j}$, where

$$n_{i,j} = (m, n) = \underset{l\in\mathcal{U},k<j}{\text{argmax}}\ \delta_{a,(i,j)}(r_{i,k} - r_{l,k}) + \delta_{\lambda,(l,k)} \quad (11)$$

$$\delta_{\lambda,(i,j)} = \delta_{a,(i,j)}(r_{i,n} - r_{m,n})\delta_{\lambda,(m,n)} \quad (12)$$

$$\delta_{a,(i,j)} = \min\{a_{m,n},\ (b_M - \overline{b}_{i,n\to j})/r_{i,n},\ \delta_{\lambda,(i,j)}/r_{i,n}\}, \quad (13)$$

and $\overline{b}_{i,n\to j} = \max_{k\in[n,j]}\{b_{i,k}\}$ is the maximum buffer state for user i from slot n to j. Computing Eqns. (11), (12) and (13) is recursive as chained swapping is also considered and can be computed from $j = s_f$ to s_l after initializing $n_{i,j} = (i, j)$, $\delta_{\lambda,(i,j)} = 0$ and $\delta_{a,(i,j)} = 1$.

The best type 1 swap (i^*, j^*) among all the possible (l, k) (there is an under-run ($l_{l,k} > 0$ to be solved and resources can be swapped $\delta_{a,(l,k)} > 0$) is

$$(i^*, j^*) = \underset{(l,k)\,\text{s.t.}\,(l_{l,k}>0\wedge\delta_{a,(l,k)}>0)}{\text{argmax}}\ \delta_{\lambda,(l,k)} \quad (14)$$

and can be resolved by following the chain starting from (i^*, j^*) which moves the under-run to $(m, n) = n_{i^*,j^*}$ by swapping $\delta_{a,(i^*,j^*)}$ resources from user m to user i^* in slot n. Then, $(i^*, j^*) = (m, n)$ and the next chained swap is resolved until $n_{i^*,j^*} = (i^*, j^*)$, which means no more swaps can be done to reduce an under-run in the last (i^*, j^*).

Conversely, type 2 swaps are defined as those that reduce the total lateness by by modifying earlier buffer states exploiting later free resources. The best type 2 swap is the one obtaining the highest product between exchengeable resources and lateness decrease $\delta_a\delta_\lambda$. To this extent, we redefine $\delta_{\lambda,(i,j)}$, $\delta_{a,(i,j)}$ and $n_{i,j}$, but in this case $(m, n) = n_{i,j}$ is where new resources are allocated to compensate for the swap from user m to i in slot j:

$$n_{i,j} = (m, n) = \underset{l\in\mathcal{U},k>j}{\text{argmax}}\ \delta_{a,(l,k)}\delta_{\lambda,(l,k)} \quad (15)$$

$$\delta_{\lambda,(i,j)} = \delta_{\lambda,(m,n)} \quad (16)$$

$$\delta_{a,(i,j)} = \min\{\delta_{a,(m,n)}r_{m,n}/r_{m,j},\ a_{m,j},\ \underline{b}_{m,n\to j})/r_{l,n}\} \quad (17)$$

where $\underline{b}_{m,n\to j} = \min_{k\in[n,j-1]}\{b_{m,k}\}$ is the minimum buffer level for user m from slot n to $j-1$ and ensures that it is possible to reduce the buffer allocation to save resources in slot j. Computing Eqns. (15), (16) and (17) is recursive as chained swapping is also considered and can be computed by going backwards from $j = s_l$ to s_f after initializing $n_{i,j} = (i, j)$, $\delta_{\lambda,(i,j)} = I(1 - \sum_{k\in\mathcal{U}} a_{k,j})r_{i,j}$ and $\delta_{a,(i,j)} = 1$.

The best type 2 swap (i^*, j^*) among all the possible (l, k) (under-run in $l_{l,k} > 0$ and $\delta_{a,(l,k)} > 0$) is

$$(i^*, j^*) = \underset{(l,k)\,\text{s.t.}\,(l_{l,k}>0\wedge\delta_{a,(l,k)}>0)}{\text{argmax}}\ \delta_{a,(l,k)}\delta_{\lambda,(l,k)} \quad (18)$$

and can be resolved by allocating new resources in $\delta_{a,(i^*,j^*)}$ and following the chain moving free resources to $(m, n) = n_{i^*,j^*}$ by swapping $\delta_{a,(i^*,j^*)}$ resources from user i to user m in slot n. Then, $(i^*, j^*) = (m, n)$ and the next chained swap is resolved until $n_{i^*,j^*} = (i^*, j^*)$, where the under-run in the last (i^*, j^*) is decreased.

Finally, Algorithm 4 lists the steps of the swap phase. Basically, each iteration checks whether a Type 1 or a Type 2 swap can be done and, in the positive case, it recomputes the allocation. Type 2 swaps are prioritized over Type 1, since it is more efficient to use remaining resources first and reducing lateness later.

Algorithm 4 Swap phase

Input: R, D, s_f, b_M, A, B.
Output: $[A, B, s_f] = \text{Swap}(R, D, s_f, b_M, A, B)$
  if Type 2 **then**
    $(l, k) = (i^*, j^*)$ as per Eq. (18)
    $\delta_a = \min\{\delta_{a,(l,k)},\ l_{l,k}/r_{l,k}\}$
    while $n_{l,k} \neq (l, k)$ **do**
      $(m, n) = n_{k,l}$
      $a_{l,k} = a_{l,k} + \delta_a$
      $a_{m,k} = a_{m,k} - \delta_a$
      $\delta_a = \delta_a\ r_{m,k}/r_{m,n}$
      $(l, k) = (m, n)$
    end while
    $a_{l,k} = a_{l,k} + \delta_a$
    Recompute B as per Eqns. (1)
  else if Type 1 **then**
    $(l, k) = (i^*, j^*)$ as per Eq. (14)
    $\delta_a = \min\{\delta_{a,(l,k)},\ l_{l,k}/r_{l,n}\}$
    while $n_{l,k} \neq (l, k)$ **do**
      $a_{l,n} = a_{l,n} + \delta_a$
      $a_{m,n} = a_{m,n} - \delta_a$
      $(l, k) = (m, n)$ and $(m, n) = n_{l,k}$
      $\delta_a = \delta_a r_{m,n}/r_{m,k}$
    end while
    recompute B as per Eqns. (1)
  end if
  $s_f = \text{NewStart}(A, B, R, b_M, s_f, T)$
  return A, B, s_f

In order to use the SS&S algorithm to solve the quality maximization problem, we can simply observe that the resources allocated for extra quality Q cannot modify those assigned to minimum requirements A, and the buffered data for extra quality must account for data already buffered for minimum requirements $B'' < b_M - B'$. Thus running

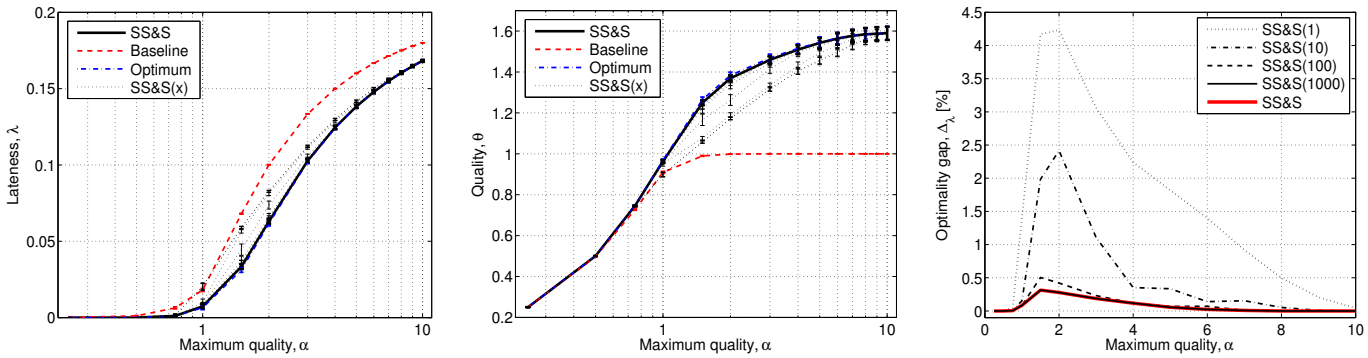


Fig. 2. Performance comparison among SS&S, optimal and baseline: the plots show the average lateness, the average total video quality normalized over the average capacity and the gap between the results obtained by SS&S and the optimal varying the number of iterations.

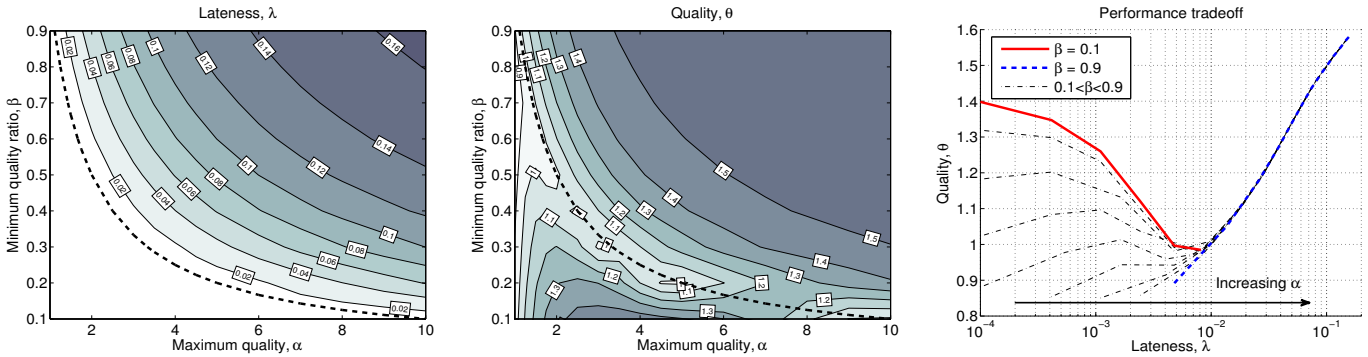


Fig. 3. Contour plots of the average lateness (left), average total quality (center) and trade off plot (right) between lateness and quality varying α and β.

$[Q, B''] = \text{SS\&S}(R, U, b_M - B')$ will provide the desired solution.

IV. Simulation Results

This section evaluates the performance of SS&S and its result after x iterations (SS&S(x)) against the optimal solution (Optimum) and the unoptimized performance (Baseline). The baseline is computed assuming proportionally fair scheduling is allocating resources [16] by allowing each users $1/K$-th of the time. While, the optimal performance is obtained as the solution of the optimization problem in Eq. (7) by means of standard solvers, such as GUROBI [15].

Each simulation is run over traces generated according to the LTE model in [16] assuming random cell deployment with average cell distance of 500 meters, users moving according to a random waypoint mobility model with an average speed of 10 meters per second, 10 active users are considered in each simulation and the video is 180 seconds long. In each trace the capacity oscillates according the the distance from the users to the base station and has 4 maxima, which are approximately 50 seconds apart from each other.

In each simulation, all traces are normalized so that the average capacity is $C = 1$ and, thus, equal for all users. In all simulation the buffer size is set to last at least 50 seconds at the maximum quality, so that a full buffer allow a user to playing the video without interruptions even if no download is made between two capacity maxima. Finally, each parameter combination is averaged over 50 runs and error bars are plotted for this averages at 95% confidence.

In order to systematically study various rate requirements for video streaming, we define two additional parameters: $\alpha \in (0, \infty)$ and $\beta \in [0, 1]$. The parameter $\alpha = K(d_{i,j} + u_{i,j})/C$ represents the maximum video bitrate requested by all users for every user i and slot j. Even though minimum and maximum requirements are assumed constant for the whole video and equal across the users, this only simplifies the numerical study and does not limit for the algorithm. Also, $\alpha = 1$ means that the demand is equal to the average offer. Instead, $\beta = d_{i,j}/(d_{i,j} + u_{i,j})$ represents the ratio between the minimum and the maximum video bitrate. Thus, $0 < \beta < 1$ means that the video quality may be as low as β in order to stream the video without interruptions.

Fig. 2 shows the first set of plots, that illustrate, from the left to the right, the average lateness λ, the average total video quality θ normalized over the average capacity and the gap between the results obtained by SS&S and the optimal.

The first plot is obtained with $\alpha \in [0.25, 10]$ in logarithmic steps and $\beta = 1$. This setup is meant to study λ as a function of the ratio between demand and offer, thus no extra quality is considered. The results obtained by SS&S and the optimal are plotted as black solid and blue dash-dotted lines respectively and they are very close to each other confirming that SS&S

obtains almost optimal performance. SS&S(x) performance are plotted for $x \in [1, 1000]$ as dotted black lines and they are increasingly close to the optimal performance with increasing x. Finally, the baseline performance is shown as a red dashed line. Notably, the baseline performance is always worse than the others and it is obtaining an average lateness more than twice (2.45) as long as SS&S when $\alpha = 1$.

The second plot, in the center, is obtained for $\beta = 0$ with everything else unchanged. This set of experiments is meant to study the maximum average bitrate achievable varying α. Again SS&S reaches almost optimal performance and both solutions outperform the baseline by up to 60% and, starting from $\alpha \leq 1.5$ of as much as 25%. As in the previous graph, increasing the number of iterations reduces the distance from SS&S(x) to the optimum. Notably, both here and in the previous graph, a larger number of iterations is needed for $\alpha \in [1, 3]$: in fact, out of this region a completely greedy solution is already good enough as the minimium requirements are either very low ($\alpha < 1$) and they can be greedily allocated to all the users or very high ($\alpha > 3$) so that only the user with the highest capacity is being allocated.

The third plot of Fig. 2 shows the difference between the lateness obtained by SS&S(x) and the optimal $\Delta_\lambda = \lambda_{\mathrm{SS\&S}(x)} - \lambda_{\mathrm{Opt}}$ varying $x \in \{1, 10, 100, 1000\}$. Again the gap is larger for fewer iterations and in the region $1 < \alpha < 3$. Also, for $x >= 1000$ are sufficient to achieve the best performance of SS&S, which, in turn, are very close to the optimal ($\Delta_\lambda < 5 \cdot 10^{-3}$). Finally, even with a single iteration the gap is smaller than 5% ($\Delta_\lambda \approx 0.043$).

The second series of plot in Fig. 3 represents from left to right: contour plots of the average lateness, contour plots of the total average quality and the trade off between lateness and quality varying both $\alpha \in [1, 10]$ and $\beta \in [0.1, 0.9]$. In the first two plots a black dashed contour is plotted to mark the boundary between the region where minimum requirements for an uninterrupted streaming are lower (bottom-left part) or larger (upper-right part) than the average capacity.

The lateness results (left) are quite intuitive as below the dashed border SS&S mostly streams the video uninterrupted at the desired minimum quality. However, crossing this border causes an increasingly higher lateness up to 20% of the video duration. Conversely, the quality contours (center) are slightly more complex: in fact the quality increases both above and below the dashed border. The quality increase when the resources are scarcer (top-right part) is justified by the fact that the system is trading lateness for quality allocating only users that can obtain higher quality. Conversely, the quality increase in the lower-left part of the figure is obtained without almost no lateness (compare to the left figure): in fact, in this region the minimum requirements $d_{i,j}$ are small compared to the average capacity, thus allocation computed by SS&S allow all the users to receive an uninterrupted stream at a quality which is at least equal to $\alpha\beta C/K$.

The trade off between lateness and quality is clearly illustrated in the right plot if Fig. 3, where λ and θ are plotted on the x and y axes respectively. The different curves are plotted for different β and each curve is obtained for varying α. Notably, the system is bound between $\beta = 0.1$ on the top-left part and $\beta = 0.9$ on the right. All the curves are quite close when $\alpha\beta = 1$. This plot can be used to estimate the expected system performance when adopting SS&S: for instance, in a system where the minimum requirements are 20% of the maximum quality (e.g.: 400 kbps and 2 Mbps) the second curve from the top can be used to decide how many users to allow in the system as a function of the desired average video bitrate; as an example, if $C = 20$ Mbps and the desired average video bitrate should not be lower than 1.5 Mbps, the user number should be $K \approx 16$, obtaining an average lateness lower than $\lambda < 10^{-4}$; more users can be traded for lower average quality, lower minimum quality or higher lateness.

V. Conclusions

In this paper we addressed the optimization problem of network resource allocation for mobile video streaming with minimum lateness and maximum average quality. We described the optimization problem as a piecewise linear program and we developed Split, Sort & Swap, a heuristic algorithm achieving close to optimal performance.

The algorithm features the following interesting characteristics. It solves the problem by first providing minimum lateness and, then, uses the remaining resources for extra quality. Thus the video playback is delivered with minimal interruptions. Being anticipative, this solution allows to preserve a smooth playback in situations of low wireless capacity by buffering video segments at the minimum quality, while improving the quality if the capacity is predicted to be sufficient. Finally, the algorithm can be stopped after any number of iterations, while always returning a feasible solution that represents an improvement to the previous iteration.

Our simulation results show that the result obtained by SS&S is very close to the optimal solution and we used the algorithm performance to draw some preliminary consideration about the performance of a system adopting SS&S for resource allocation. As a future work, we will study the algorithm with imperfect prediction in realistic environments.

Acknowledgments

The research leading to these results was partly funded by the European Union under the project eCOUSIN (EU-FP7-318398).

References

[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update," 2014.

[2] Sandvine Incorporated ULC, "Global Internet Phenomena 1H 2014 Report," 2014.

[3] S. Wang, Y. Xin, S. Chen, W. Zhang, and C. Wang, "Enhancing spectral efficiency for LTE-advanced and beyond cellular networks [Guest Editorial]," *IEEE Wireless Communications*, vol. 21, no. 2, pp. 8–9, April 2014.

[4] U. Paul, A. P. Subramanian, M. M. Buddhikot, and S. R. Das, "Understanding traffic dynamics in cellular data networks," in *IEEE INFOCOM*, Shangai, China, April 2011, pp. 882–890.

[5] M. Z. Shafiq, L. Ji, A. X. Liu, and J. Wang, "Characterizing and modeling internet traffic dynamics of cellular devices," in *ACM SIGMETRICS*, New York, NY, USA, 2011, pp. 305–316.

[6] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453, no. 7196, pp. 779–782, 2008.

[7] M. Ahmed, S. Spagna, F. Huici, and S. Niccolini, "A peek into the future: predicting the evolution of popularity in user generated content," in *ACM WSDM*, Rome, Italy, February 2013, pp. 607–616.

[8] M. Draxler and H. Karl, "Cross-layer scheduling for multi-quality video streaming in cellular wireless networks," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE, 2013, pp. 1181–1186.

[9] Z. Lu and G. de Veciana, "Optimizing stored video delivery for mobile networks: The value of knowing the future," in *IEEE INFOCOM*, Turin, Italy, April 2013, pp. 2706–2714.

[10] H. Abou-zeid, H. Hassanein, and S. Valentin, "Optimal predictive resource allocation: Exploiting mobility patterns and radio maps," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, 2013, pp. 4714–4719.

[11] ——, "Energy-efficient adaptive video transmission: Exploiting rate predictions in wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 5, pp. 2013–2026, June 2014.

[12] R. Margolies, A. Sridharan, V. Aggarwal, R. Jana, N. Shankaranarayanan, V. A. Vaishampayan, and G. Zussman, "Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms," in *Proc. IEEE Int. Conf. on Computer Commun.(INFOCOM)*, 2014.

[13] N. Bui, F. Michelinakis, and J. Widmer, "A model for throughput prediction for mobile users," in *European Wireless*, Barcelona, Spain, May 2014.

[14] R. Pantos and W. May, "HTTP live streaming," *IETF Draft, June*, 2010.

[15] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2014. [Online]. Available: http://www.gurobi.com

[16] O. Østerbø, "Scheduling and capacity estimation in lte," in *IEEE ITC*, San Francisco, CA, USA, September 2011, pp. 63–70.