Abstract—Scheduling of real-time uplink flows in HSUPA is hampered by the fact that the NodeB does not know the size and dead-lines of the uplink packets, as the User Equipment (UE) only reports coarse-grained Scheduling Information (SI) advertising the overall backlog. In this paper, we describe how the NodeB can use the sequence of SI to reconstruct a UE queue, using affordable computations. This makes it possible to use scheduling algorithms employing packet sizes and deadlines in HSUPA, with obvious performance benefits. We present a general Virtual Queueing framework, that can be employed with any kind of traffic, and a specialized version of it for periodic (e.g., voice) traffic.

Keywords—scheduling, HSUPA, Real-time traffic

I. INTRODUCTION

3G+ cellular networks, such as High-Speed Packet Access (HSUPA) and Long Term Evolution (LTE) of the UMTS system, are and will be exploited to provide users with diverse services, therein including real-time ones, such as voice and video calls. Real-time services require a bounded end-to-end delay, which in turn cannot be enforced unless the delay at each scheduling hop is bounded as well, with access segments playing a key role because of their scarcer bandwidth. We focus on the uplink access segment of a cellular network, where scheduling is coordinated by a central entity, called NodeB, to which User Equipments (UEs, e.g. mobile phones or handheld devices) send their traffic. At every scheduling period, called Transmission Time Interval (TTI), the NodeB computes Serving Grants (SG) for each UE, which determine the amount of bits that an UE can transmit. The data transmitted across the radio interface are buffered in the layer-2 UE queues as Protocol Data Units (PDUs) of constant (configurable) length. Long data packets from higher layers are segmented into a number of PDUs, and padding can be added to fill up the last PDU. The NodeB obtains the backlog state of the UE queues from the Scheduling Information (SI), which are either piggybacked in the PDUs sent by the UE, if any, or sent in a standalone transmission otherwise. The SI are quantized according to a non-linear table (see [1]), so that reconstructing the exact backlog state is not always possible.

A clever scheduling algorithm should compute the SG based on the SI reported by the UEs. In fact, not doing so incurs the risk of issuing overlength SGs, thus actually wasting resources and reducing the number of UEs that can be served in a TTI. However, when real-time traffic is considered, one needs to know not only how many PDUs are in each queue, but also by what deadline they should be transmitted, which cannot be inferred from the SI alone.

Several schedulers have been proposed in the literature for HSUPA. The dual scheduler of Max CIR is usually referred to as Uplink CQI (UCQI). As for the downlink, some papers advocate variations of the PF approach (e.g.[10],[14]). [11] proposes to classify traffic into CBR-real-time, VBR-real-time, and non-real-time, and to give fixed grants to flows of the first class and variable grants to the other two. However, it does not specify neither how grants are computed, nor how the channel conditions are actually taken into account. A queuing-theoretical approach is presented in [13], where a maximum stability region is computed under the assumption of Poisson arrivals. In [12] the scheduling problem is formulated as an integer nonlinear optimization problem, keeping into account the channel conditions, the UE buffers and UE requested rates. An integer-linear approximated version is also proposed, which is likely to be too complex to be implemented nonetheless. None of the above schedulers take into account real-time constraints. A scheduler explicitly tailored for VoIP traffic is shown in [15], and henceforth referred to as UL-VoIP. The latter selects UEs in a round robin order, and allocated a fixed, pre-computed SG equal to 8db. Such value is judged to be optimal for serving VoIP traffic with 31 bytes packets according to link-level simulations. Note that, to the best of our knowledge, no HSUPA scheduler exists that exploits deadline information.

In this paper, we show that, using affordable computations, a NodeB can reconstruct a sufficiently accurate estimate of the backlog state of UE queues taking the SI as an input. Such estimate is called a Virtual Queue (VQ), and it is represented by a list of couples \([l_{0,hi}, d_{i}]\), where \(l_{0,hi}\) are the lower and upper bounds for the number of PDUs in the UE queue generated by time \(d\). This allows any scheduling algorithm that takes into account the above information (e.g., those used for real-time scheduling in the downlink direction, where queues are located at the NodeB – see for example [2]-[5]) to be adapted to working for the uplink via minor modifications. While the VQ algorithm does not require any assumption on actual traffic generation profile, thus being amenable to any kind of traffic, periodic or quasi-periodic real-time sources are indeed frequent. For instance, VoIP flows are quasi-periodic: they alternate “on” periods (talkspurts), when they generate fixed-length packets with a constant interarrival time, and – if they have Voice Activity Detection (VAD), as it normally happens – “off” or si-
lence periods, when they either do not generate packets at all, or generate smaller packets. Usually, information on the voice codec can be acquired from higher layers at the setup of a flow and conveyed to the NodeB. Such information can be used to identify the flow characteristics (i.e., the flow period and packet size, whether it has VAD or not, whether it generates packets during off periods, etc.). Based on the above information, the VQ algorithm can be refined to better match the flow characteristics, thus reducing the error in the reconstructed VQ. More to the point, if we know that traffic is periodic, we can predict the arrival of PDUs in the near future (i.e., the next 1-2 TTIs). This is particularly important since issuing SGs reactively, i.e. based on the SI reported by the UEs, undergoes a signaling delay equal to (at least) 2 TTIs, i.e. 20ms in a 10ms-TTI HSUPA system. Such delay could instead be avoided by predicting the backlog status of the UE queue at the time a possible SG would actually be used, i.e. in the subsequent TTI. This way, SGs can be issued proactively, i.e. based on the predicted status of the UE queue, before the SI is actually conveyed to the NodeB. Proactive scheduling can coexist with standard (reactive) scheduling of non-periodic uplink flows, and it can be turned on and off at will on the same flow, depending on whether a reliable estimate of the packet generation instants is available or not.

VQ can be applied to several cellular technologies, e.g. HSUPA, and LTE, and in general wherever coarsened buffer occupancy reports are issued to a central scheduler that also assigns SGs for the users. For the sake of concreteness, we describe it with reference to its HSUPA embodiment. Defining criteria for selecting users and allocating SGs to them is not within the scope of this paper. Rather, the framework described herein can be used with any scheduler. However, for the sake of demonstrating its effectiveness, we present performance results when it is employed in conjunction with the Hybrid Real-Time and Channel-Aware (HY-CART) scheduler [5], recently proposed for HSPA. The latter is known to achieve better performances than other schedulers known in the literature with real-time traffic.

To the best of our knowledge, the idea of virtualizing the uplink queues at the NodeB has not received attention in the literature so far. A previous work of ours [5] employs a vanilla version of the VQ scheme to adapt HY-CART to the uplink segment. The algorithm used therein, which, is only briefly outlined, is however considerably less effective than those presented in this paper.

The rest of the paper is organized as follows: we provide background on HSUPA in Section 2. The VQ framework is described in Section 3. Section 4 describes performance evaluation results, and highlight conclusions in Section 5.

II. HIGH-SPEED UPLINK PACKET ACCESS

This section reports background on those aspects of HSUPA connected to the VQ mechanism, notably the reporting of Scheduling Information and the timing of Service Grants.

In HSUPA the NodeB coordinates scheduling decisions at TTIs of 2 or 10ms. Hereafter, we assume for simplicity that UE transmissions are directed to a single NodeB. Transmissions are acknowledged and protected by a Hybrid ARQ. A single UE can open up to eight flows simultaneously. According to the standard, the latter are arbitrated by an internal strict-priority scheduler. The PDUs are physically stored at the UEs. However, the NodeB obtains the backlog state of the UE queues from the Scheduling Information (SI), which include the total backlog on the UE and the percentage of the latter accounted to the highest priority flow (i.e., the one which will be selected by the internal UE scheduler). Both are quantized, using five and four bit indexes respectively, according to two non-linear tables (see [1]), so that reconstructing the exact backlog state is not possible. The UE always reports an overestimated backlog to the NodeB.

We assume that A generic UE has one real-time flow, which is internally scheduled at the highest priority. The QoS class of the flow is known to the NodeB from the setup negotiation. Some other information, hereafter called flow information, may (or may not) be available at the NodeB. This information includes the voice codec employed, one in a finite number of possibilities.

A fragment of the standard SI quantization table [1] is reported in Table 1. The table reports a dimension in bytes, whereas the queues actually contain a number of fixed-length PDUs, whose length is selected by the NodeB. A common value, to which we will stick in the following unless otherwise stated, is 40 bytes. We remark that the algorithms described herein do not depend on a particular PDU size, although their performance and effectiveness may vary with the latter. The number of 40-byte PDUs associated to each SI value is also shown in Table 1. For instance, the UE reports a value of 18 if the queue is buffering 24 to 31 PDUs. For a given PDU size, the quantization table is divided in two zones: a non ambiguous zone, consisting in the set of SI that allows one to infer an exact backlog (i.e., , 0 ≤ SI ≤ 12 ) and an ambiguous one, where the quantization intervals are larger than the PDU size. Furthermore, note that — in the non ambiguous region — not all the SI values are actually possible. The width of the ambiguous region depends on the PDU size. The larger the PDU size, the smaller the ambiguous region, and, within the latter, the width of each quantization interval. Note, however, that the exact shape of the quantization table (e.g., the fact that quantization intervals are increasing) is not a prerequisite for the algorithms we present.

A UE application generates packets so that, at some time instants, a number of PDUs is inserted in the (FIFO) UE queue. When the UE receives a SG from the NodeB, it transmits one MAC-E PDU, which includes zero or more PDUs and SI, in a single TTI. Those PDUs may belong to different packets, as original packets are reassembled at the NodeB. This means that the scheduler in the latter may be made aware of packet boundaries when it receives a set of PDUs. However, we do not rely on this assumption here. Packets queued at the UE can be removed either by scheduling decisions taken at the NodeB, or dropped by the UE after a known dropping timeout. Without loss of generality, we assume that one packet is generated at a TTI. In fact, the UE is not required to transmit packets atomically, and it can mix PDU of several packets (however maintaining the FIFO ordering) in the same transmission. Further-
more, it is not possible to pin down events with a smaller resolution than the TTI, from a real-time scheduling point of view. Packets of a flow generated in the same TTI would have the same deadline nonetheless. From now on, for ease of exposition, we normalize times to the TTI length, so that TTIs are natural numbers. The SI issued at time $T$ by the UE, reporting the state of its queue up to that time, arrives at the NodeB at $T + 1$. Assuming (for the sake of discussion) that the NodeB makes a scheduling decision in zero time, the NodeB might be able to keep that SI into account when computing the subsequent SG. The latter in turn would arrive at the UE at time $T + 2$. Therefore, in the very best case, we have a signaling delay equal to 2 TTIs. Figure 1 reports an example of the above signaling sequence. Note that, in a 10ms TTI deployment, such a signaling delay is indeed attainable, since the SG is actually 2ms long, and they are repeated five times in the 10ms period. It is thus foreseeable that the NodeB might take, e.g., 2ms for computing the SGs for the UEs in the cell, leave the first 2ms instance blank, and actually repeat the correct SG four times. The UEs would therefore be able to decode the correct SG simply by skipping the first instance. In a 2ms TTI deployment, instead, there seems to be no other way than wasting at least another TTI for allowing the NodeB to make scheduling decisions, which brings the signaling delay to 3 TTIs.

Note that information received at the NodeB at $T$ refers to quantities generated before $T − 1$. Henceforth, we will assume the NodeB as a time reference, as decisions are taken by the latter. Therefore, we refer to “SI at time $T$” to define SI received at that time (hence generated at time $T − 1$ at the UE). Finally, note that, for the above reason, a packet has an intrinsic minimum delay between one and two TTIs, depending on its arrival offset with respect to a TTI boundary.

III. Virtual Queuing

The VQ scheduling framework is shown in Figure 2. The Virtual Queuing block collects the sequence of SI and transmitted PDUs from the UE, and uses this information to reconstruct the state of the UE queue in the most accurate possible way, using affordable computations. A basic algorithm, which does not make any assumption on the traffic profile, will be described in a minute. In some cases, we might know more about the flow, and exploit this knowledge to improve the VQ estimate. We define a flow profile as a set of information such as: whether the traffic is periodic, and whether the generated packets have a constant length. The above information can be collected at flow setup. We assume that the NodeB is configured with a table which associates codecs with the following information: i) a flow type (e.g., CBR, or CBR on/off or non CBR), ii) a flow period, and iii) a packet length. This information can be conveyed to the NodeB at the time of flow setup, through means which are outside the scope of this paper. If a flow is known to be CBR or CBR on/off, this information can be used both to specialize the VQ algorithm and to enable proactive SG assignment. As for the first issue, knowing its packet length and period allows one to overcome the uncertainties in the VQ estimate that arise due to the SI quantization. Furthermore, as it allows one to predict the size and generation instant of packets, it enables the NodeB to assign suitably large SGs proactively to periodic flows.

A. Basic algorithm for Virtual Queuing

The goal of VQ would be to enable the NodeB to reconstruct the exact backlog state on the UE. Unfortunately, this is not always possible due to the quantized nature of the SI. Although an exact packet length for every generated packet cannot be inferred from the SI in general, we show that by considering the sequence of reported SI and transmitted PDUs we can increase the accuracy of the VQ estimate, thus reducing the impact of the SI quantization. We first present the general algorithm. Later on, in Section 3.B, we show how to refine it if flow profile information is available.
A VQ is a FIFO queue of items \(\{(lo, hi), d\}\), where \(d\) is an estimated packet generation time, and \(lo, hi\) are the lower and upper bounds for the overall VQ backlog including all packets generated until time \(d\). Figure 2 reports a possible snapshot of a VQ: 1 to 3 PDUs that are still sitting in the UE queue were generated at time 1, whereas 5 to 7 PDUs were generated up to time 3 (which implies that 5-3=2 to 7-1=6 PDUs were generated at time 3) and so on. While one can easily infer lower and upper bounds for each packet in the VQ from the above information, we prefer to store the queue length at time \(t\) in the VQ, since the above information is more precise than the length of single packets. In fact, for the latter the uncertainty in the estimation of the queue length for two consecutive packets is summed up. This means that we can estimate the length of single packets from the cumulative lengths if need be, but not vice versa (at least, not without introducing further errors). Furthermore, the length associated to the head of the VQ still describes the head-of-line packet, which in fact enables deadline-based scheduling.

Call \(S_i, D_i, q\) the SI, the number of transmitted PDUs and the queue length at time \(i\), and call \(L(\cdot), H(\cdot)\) the functions that report the lower and upper bounds on the UE queue length from the SI given as a parameter. For instance, with reference to Table 1, we have \(L(15)=11, H(15)=13\). Note that, according to the standard, when a UE sends both PDUs and SI, the backlog reported by the latter does not take into account the PDUs just transmitted.

At a high level, the VQ estimation works as follows:

- Every time a set of \(k\) PDUs is transmitted by the UE to the NodeB, the number \(k\) is subtracted from the upper and lower bounds of every entry in the VQ, and when the upper bound of the head-of-line entry reaches zero the latter is removed from the front.

- Every time a new SI arrives at the NodeB, the new information is used to either improve the length estimate for the tail of the VQ, or to detect the generation of a new packet at the UE.

- When a packet generation is detected at time \(t\), a new entry \(\{L(S_i), H(S_i), t\}\) is added to the tail of the VQ.

- In order to explain the algorithm in more detail, let us first start with describing what happens with an initially empty queue, assuming that a packet is generated at time 0. After describing that, we will move to the general case. At every subsequent time instant \(i\), the following relationships hold:

\[
\forall i \geq 0, \quad L(S_i) \leq q_i \leq H(S_i) \tag{1}
\]

Assume that no other packet is generated at the UE after time 0 for a while. Call \(i\) a generic instant. Then, for any time \(j\) between 0 and \(i\), it is easy to see that:

\[
\forall i \geq 0, \forall j: 0 \leq j \leq i, \quad q_j = q_i - \sum_{r=j+1}^{i} D_r \tag{2}
\]

This means that the queue is not growing. In fact, some PDUs might be transmitted by the UE. If this happens, then the SI at time \(j\) will reflect the new state of the queue, which will generally be smaller than at a previous instant (assuming that no packets are generated after time 0). By merging (1) and (2), the following expression is obtained:

\[
Q^L_i = \max_{0 \leq j \leq i} \left\{ L(S_j) - \sum_{r=j+1}^{i} D_r \right\} \leq q_i \leq \min_{0 \leq j \leq i} \left\{ H(S_j) - \sum_{r=j+1}^{i} D_r \right\} = Q^H_i \tag{3}
\]

\(Q^L, Q^H\) are the most accurate bounds on the length of the queue available at time \(i\) given the above information. Now, in the absence of a new packet arrival, \(Q^L, Q^H\) are increasing and decreasing respectively, thus narrowing down the uncertainty for \(q_i\). Therefore, in order to provide the scheduler with a consistent view at every TTI, both \(Q^L, Q^H\) should be recomputed and the corresponding VQ entry updated. However, if another packet is generated at the UE at some time \(k\), then \(Q^L_k > Q^H_k\) might possibly take place for some \(h \geq k\). We detail this using a simple example.

Suppose that a 20-PDU packet is generated at time 0. Then we have \(S_0 = 17, L(S_0) = 18, H(S_0) = 23\). Thus, an entry is inserted in the VQ: \(\{18, 23, 0\}\). At time 1, no PDUs are transmitted and \(S_1 = 17\). At time 2, the UE transmits four PDUs, and sends \(S_2 = 17\) again. According to (3), we now have:

\[
Q^L_2 = \max \left\{ L(S_2) - 4, L(S_1) - 4, L(S_0) \right\} = 18
\]

\[
Q^H_2 = \min \left\{ H(S_2) - 4, H(S_1) - 4, H(S_0) \right\} = 19
\]

This means that the number of PDUs in the queue at time 0 was at least \(q_0 \geq 18 + 4 = 22\), and \(q_0 \leq 23\). Note that the initial uncertainty on the number of PDU included was \(23 - 18 + 1 = 6\) PDUs at time 0, and it is now \(19 - 18 + 1 = 2\) PDU, i.e. it decreases over time. Suppose now that, at time 3, two more PDUs are received at the NodeB, and the UE still reports \(S_3 = 17\). Clearly, this implies that another packet must have been generated: if it had not, then the UE queue would be between 16 and 17 PDUs, and the reported SI would be \(S_i = 16\). If we instantiate again (3) at time 3, we get:

\[
Q^L_3 = \max \left\{ L(S_3) - 6, L(S_2) - 6, L(S_1) - 2, L(S_0) \right\} = 18
\]

\[
Q^H_3 = \min \left\{ H(S_3) - 6, H(S_2) - 6, H(S_1) - 2, H(S_0) \right\} = 17
\]

The inconsistency is revealed by the fact that \(Q^L_3 > Q^H_3\). When this happens, the most accurate estimate for the length of the first entry in the VQ is \(\{Q^L_3, Q^H_3\}\), and a new entry \(\{L(S_4), H(S_4)\}, h\) has to be added to the tail of the VQ. From then on, the estimate for the previous entry (i.e., the last-but-one) is frozen, and every subsequent SI can only be used to improve the last entry in the VQ. At TTI \(h\), the computation of \(Q^L, Q^H\) is restarted, i.e., we rewrite (3) replacing 0 with the arrival time of the most recent packet. \(\forall i \geq h\),
\[ Q^i = \max_{k \leq j \leq d} \left\{ L(S_j) - \sum_{r=j+1}^i D_r \right\} \leq q_i \]
\[ \leq \min_{k \leq j \leq d} \left\{ H(S_j) - \sum_{r=j+1}^i D_r \right\} = Q^H_i \]

Note that \( Q^i, Q^H_i \) can be computed in constant time, as (4) boils down to:
\[ Q^i = \max \left\{ L(S_j) - D_j \right\}, \quad Q^H_i = \min \left\{ H(S_j) - D_j \right\} \]

We observe that it is not always possible to identify the exact TTI when the new packet is generated, unless additional information on the packet generation process are available. Thus, we decide to acknowledge that a packet has been generated when \( Q^i > Q^H_i \).

Assume now that the VQ includes \( N \) entries \( \left\{ \left[m', n', t' \right] \right\} \) (we use a superscript for the entries in the VQ and a subscript for time instants): when a set of PDUs is transmitted by the UE, the following actions should take place:

\[ \forall i, \ 1 \leq i \leq N-1, \ m' = \max \left\{ 0, m' - k \right\}, \ n' = n' - k \]

Furthermore, if \( n' \leq 0 \) the head-of-line entry has to be removed. This implies that we may consider more PDUs than necessary as being generated at a given time instant. However, this cannot be helped without additional information. If, e.g., the NodeB sees packet boundaries and we can be sure that only one packet is generated at a TTI, this last part of the algorithm could be refined. Note that the last entry in the VQ (i.e., the \( N^{th} \)), is instead updated through (5).

Now, computing (6) is not expected to be too costly, since the number of packets in a UE queue should be small enough, especially with real-time traffic. However, one might also trade accuracy for speed as follows:

- when PDUs are transmitted by the UE, (5) is computed for the last entry and the transmitted PDUs are subtracted from the head-of-line entry only;
- each VQ entry is augmented with a counter \( q' \). The latter is set to \( n' - 1 \), i.e. the upper bound on the estimate of the previous entry;
- once \( n' \) becomes null or negative, the first entry is removed, and for the new head-of-line entry we subtract \( q' \) from both the lower and upper bounds, as well as any remainder from the previous transmission.

This makes it possible to update the VQ in constant time, without worsening the length estimate of either the head-of-line packet or the whole queue, which are normally the most important information taken into account by a real-time scheduler. However, it is not possible anymore to correctly assess the length of the queue up to the \( x^{th} \) packet, \( 1 < x < N \), at least not without tolerating \( O(N) \) operations.

The alert reader will notice that three issues influence the accuracy of the VQ algorithm: first of all, the PDU length. In fact, the uncertainty in the estimate is inversely proportional to the PDU size. Second, the scheduling algorithm. The process of populating the VQ, although logically disjoint from SG selection, cannot be considered in isolation with respect to the latter. In fact, since the quantization intervals in the SI table get larger as the size of the UE queue increases, the uncertainty in the VQ state gets larger if we allow the UE queue to build up. For instance, if a UE is not served for a very long time, and its queue derives persistently towards a region where a packet arrival might go undetected due to SI quantization, then VQ proves less effective. It is therefore a task of the scheduler to ensure that this does not happen in normal operating conditions. Last, but not least, the traffic pattern: it is perfectly possible to devise a traffic generation and scheduling pattern that systematically baffles out the VQ algorithm. As an example, assume that a packet of length \( x = L(S_h) \) is generated at time \( h \), such that \( L(S_h), H(S_h) \) is a large interval. The UE then generates one PDU at every TTI. In these conditions, it might take the NodeB up to \( H(S_h) - L(S_h) \) TTIs before detecting that new traffic has been generated. On the other hand, if the UE just generates one 1-PDU packet at time \( h + 1 \) the latter might go altogether undetected, as if the UE had generated a single packet whose length is \( x + 1 \) at time \( h \). In other words, the VQ algorithm works best if the PDUs are not too small, the traffic is sufficiently regular, and scheduling is performed such that the length of the UE queue does not grow large.

B. Specialized VQ for CBR on/off flows
The VQ algorithm described in the previous section can be employed with any kind of traffic, as it does not rely on additional information on the flow profile (e.g., whether the traffic is periodic or not). However, if the flow profile shows the flow generation and scheduling algorithm that systematically bafles out the VQ algorithm. As an example, assume that a packet of length \( x = L(S_h) \) is generated at time \( h \), such that \( L(S_h), H(S_h) \) is a large interval. The UE then generates one PDU at every TTI. In these conditions, it might take the NodeB up to \( H(S_h) - L(S_h) \) TTIs before detecting that new traffic has been generated. On the other hand, if the UE just generates one 1-PDU packet at time \( h + 1 \) the latter might go altogether undetected, as if the UE had generated a single packet whose length is \( x + 1 \) at time \( h \). In other words, the VQ algorithm works best if the PDUs are not too small, the traffic is sufficiently regular, and scheduling is performed such that the length of the UE queue does not grow large.

Now, computing (6) is not expected to be too costly, since the number of packets in a UE queue should be small enough, especially with real-time traffic. However, one might also trade accuracy for speed as follows:

\[ \forall i, \ 1 \leq i \leq N-1, \ m' = \max \left\{ 0, m' - k \right\}, \ n' = n' - k \]

Furthermore, if \( n' \leq 0 \) the head-of-line entry has to be removed. This implies that we may consider more PDUs than necessary as being generated at a given time instant. However, this cannot be helped without additional information. If, e.g., the NodeB sees packet boundaries and we can be sure that only one packet is generated at a TTI, this last part of the algorithm could be refined. Note that the last entry in the VQ (i.e., the \( N^{th} \)), is instead updated through (5).

Now, computing (6) is not expected to be too costly, since the number of packets in a UE queue should be small enough, especially with real-time traffic. However, one might also trade accuracy for speed as follows:

- when PDUs are transmitted by the UE, (5) is computed for the last entry and the transmitted PDUs are subtracted from the head-of-line entry only;
- each VQ entry is augmented with a counter \( q' \). The latter is set to \( n' - 1 \), i.e. the upper bound on the estimate of the previous entry;
- once \( n' \) becomes null or negative, the first entry is removed, and for the new head-of-line entry we subtract \( q' \) from both the lower and upper bounds, as well as any remainder from the previous transmission.

This makes it possible to update the VQ in constant time, without worsening the length estimate of either the head-of-line packet or the whole queue, which are normally the most important information taken into account by a real-time scheduler. However, it is not possible anymore to correctly assess the length of the queue up to the \( x^{th} \) packet, \( 1 < x < N \), at least not without tolerating \( O(N) \) operations.

The alert reader will notice that three issues influence the accuracy of the VQ algorithm: first of all, the PDU length. In fact, the uncertainty in the estimate is inversely proportional to the PDU size. Second, the scheduling algorithm. The process of populating the VQ, although logically disjoint from SG selection, cannot be considered in isolation with respect to the latter. In fact, since the quantization intervals in the SI table get larger as the size of the UE queue increases, the uncertainty in the VQ state gets larger if we allow the UE queue to build up. For instance, if a UE is not served for a very long time, and its queue derives persistently towards a region where a packet arrival might go undetected due to SI quantization, then VQ proves less effective. It is therefore a task of the scheduler to ensure that this does not happen in normal operating conditions. Last, but not least, the traffic pattern: it is perfectly possible to devise a traffic generation and scheduling pattern that systematically bafles out the VQ algorithm. As an example, assume that a packet of length \( x = L(S_h) \) is generated at time \( h \), such that \( L(S_h), H(S_h) \) is a large interval. The UE then generates one PDU at every TTI. In these conditions, it might take the NodeB up to \( H(S_h) - L(S_h) \) TTIs before detecting that new traffic has been generated. On the other hand, if the UE just generates one 1-PDU packet at time \( h + 1 \) the latter might go altogether undetected, as if the UE had generated a single packet whose length is \( x + 1 \) at time \( h \). In other words, the VQ algorithm works best if the PDUs are not too small, the traffic is sufficiently regular, and scheduling is performed such that the length of the UE queue does not grow large.
The only problem here is to detect the exact TTI when a packet is generated, from which all the packet generation instants can be estimated. This can be done if one receives two consecutive SI which reveal a packet generation. For instance, assuming a 2 TTI period and 1-PDU packets, if $SI(i) = 4$, $SI(i+1) = 5$, then packets are generated at time $5 + 2 \cdot k$, $k \geq 0$.

If, as it happens with some voice codecs, packets have a non-integer period (e.g., 33.3ms, i.e. 10/3 of a 10ms TTI), then the computations - although conceptually simple - are slightly more involved, and are omitted due to space limitations. In fact, in this case, knowing the period and the TTI of the first packet of a talkspurt is not sufficient to determine the TTIs when packets are generated: we also need to determine the offset with respect to the TTI boundary, i.e. whether $0 \leq o < 1/3$, $1/3 \leq o < 2/3$, $2/3 \leq o < 1$ in this case, as shown in Figure 3.

Note that the condition used to detect the generation of a packet is an increase in the reported SI. This means that we implicitly assume that, during a silence period, the UE queue can be drained sufficiently, so that there is no ambiguity in the detection of the first packet. Such assumptions are reasonable, since the duration of a silence period is considerably larger than the packet period and the ambiguous region starts when there are several (e.g., more than 5) packets in the queue. Similarly, the onset of a silence period is detected through mismatch between the SI and the VQ length. Again, this condition might be detected with some delay if the UE queue is in the ambiguous region when this happens. However, late detection of a silence period is not particularly harmful, since the VQ is rapidly reset to the correct length, and virtual packets entered by mistake have little, if any, chance of being seen by the scheduler.

C. Proactive VQ

As shown in Section 2, the signaling delay associated to SI reporting and SG scheduling amounts to at least two TTIs, assuming that at time $T+1$ the NodeB can issue a SG which takes into account the SI sent by the UE at time $T$. This delay is unavoidable if reactive SG assignment is used, even if a UE queue is always emptied right after each non-zero SI report. As the TTI duration in HSUPA can be as high as 10ms, 20ms of added delay in the uplink direction are not negligible, especially with voice applications. However, such delay can be removed by employing a proactive SG assignment scheme, exploiting the VQ algorithm and the prediction of the time instants at which packets are generated. In fact, if the packet length and generation instants are known, SG can be scheduled based on the presumed backlog state of the UE at the time when the SG will actually be used. As an example, if the NodeB knows that voice packets are generated each 20 ms (i.e., every other TTI) starting from time 0, it may schedule a SG large enough to hold two packets at time 2, and periodically schedule another SG for one packet, as shown in Figure 4.

This way the delay of each packet (under ideal scheduling conditions) would be reduced to 1.5 TTI on average, which is actually the lower bound. All it takes to make it possible is to run the VQ algorithm at $T$, however predicting the SI that the UE is going to send at $T+1$ (in fact it will use the grant that the NodeB is computing), thus enabling a scheduler to assign SGs based on the estimated state of the VQ at time $T+1$. In this example, assuming a CBR on/off flow, a proactive scheduling would reduce the delay of each packet in a talkspurt except the first one. In fact, the beginning of a talkspurt can only be detected by looking at the SI (i.e., reactively), so that the first packet actually has a higher delay (3.5 TTI on average) than the rest. Likewise, since the onset of a silence period can only be detected reactively, i.e. through a mismatch between the VQ and the SI, an SG might be wasted at the end of each talkspurt (unless the UE has lower priority traffic to send). Given that the average number of voice packets in a talkspurt is rather large, and that the SG required for servicing a voice packet is normally small, this results in a negligible waste of resources. Moreover, if a scheduler takes into account channel conditions (e.g., favors UEs with a better channel), being proactive would increase by two (i.e., the number of TTIs in the signaling delay) the number of scheduling opportunities for a given packet, thus possibly allowing for a better exploitation of the channel characteristics.

Proactive VQ can be enforced only if the traffic profile is predictable (i.e., CBR on/off). Let us look into this in more detail, with the help of Figure 5. We want to enable a scheduler to compute an SG at time $T$. At that time, it has $SI(T)$, generated at time $T-1$ by the UE. On the other hand, the SG will be used at time $T+1$ by the UE. Thus, the SG has to be computed taking into account:

- Packets generated between $T-1$ and $T+1$ at the UE
- PDU transmitted by the UE at time $T$, according to the SG that was sent at time $T-1$.

1 In the figure, we slightly abuse the notation for conciseness’s sake. $SG(2p)$ means an SG large enough to transmit two packets. Non relevant quantities (e.g., the SI generated at the UE at time 2) are omitted for ease of reading.
Therefore, at time $T$ the NodeB should do what follows:

1. update the VQ including $S(t)$;
2. predict the VQ state at time $T^+$, $VQ_{T^-}$ (which would be known at $T+1$), by predicting the arrivals in $[T-1,T^+]$ according to the flow profile;
3. starting from $VQ_{T^-}$, estimate the PDUs that the UE will transmit at time $T$ as the minimum between those allowed by the SG issued at time $T-1$ and those in $VQ_{T^-}$. Note that those PDUs will not be reported anymore once they have been sent to the H-ARQ process for transmission (even if they need retransmitting). This allows one to compute $VQ_T$;
4. predict the VQ state at $(T+1)^+$, $VQ_{T+1^-}$ (which would be known at $T+2$), by predicting the arrivals in $[T,T+1)$ according to the flow profile.

Obviously, if at time $T$ the UE is not eligible for transmitting new PDUs (e.g., due to a retransmitting H-ARQ process), the NodeB simply skips it when allocating SGs.

IV. PERFORMANCE EVALUATION

We evaluate the VQ algorithms through simulation, using HY-CART [5] as a scheduler and comparing their joint performance with the UL-VoIP scheduler [15]. The latter selects UEs in a round robin order, and allocates a fixed, pre-computed SG. HY-CART, instead, exploits the VQ by computing urgent data, i.e., those whose deadline expires within few TTI. It then computes the UE priority by summing up its urgency, i.e., a function of $i$) the amount of the urgent data, and $ii$) of how near their deadline is, and its channel state, and sorts UEs accordingly. When a UE is selected, its SG is computed as so as to serve its urgent data only.

We have implemented the HSPA framework in a quasi-static system-level network simulator, including full NodeB and RNC functionalities. The following VQ algorithms were coded: HY-CART [5] standard, the basic VQ, and the specialized and Proactive VQ for CBR on/off flows. The physical layer is capable of simulating a single HSPA cell in a wide-area scenario with ITU standard multipath profiles, including HSUPA power control functionalities. The application layer is simulated on every UE, in which we implemented a VoIP traffic generator.

We simulate a single cell of 425m radius, with the NodeB at its center. The NodeB is connected to the RNC node via an optical fiber connection (IuB interface), whose bandwidth is 622 Mbps, large enough not to be a bottleneck. The RNC is connected to a simulated Core Network, which introduces a constant delay equal to 40ms. As far as scheduling constraints are concerned the maximum cell load for HSUPA is set equal to 90%. As for physical layer, the simulated channel profile is a pedestrian A3 multipath profile [6] with shadowing modeled by a log-normal distribution whose standard deviation is 8db.

The mobility model we adopted is the same one employed in the EURANE simulator [9]: each UE moves at a constant speed of 3 km/h around the NodeB. The inter-cell interference is taken into account thanks to specific models. The simulated NodeB has a transmission power of 43 dBm, while the UEs are Class 3 HSPA devices (thus supporting 10ms TTIs in HSUPA), with a maximum transmission power of 21 dBm. UEs have a drop timer equal to 400ms. Simulation time is 200 seconds.

VoIP traffic is modeled according to the VoIP ns-2 application [7], whose set of parameters is summarized in Table 2. At the receiver, an optimal playout buffer is assumed [7], whose performance upper bounds that of any real-life playout buffer.

Figure 6 reports the average per-talkspurt Mean Opinion Score (MOS) [8] for each conversation in a scenario with 50 VoIP users, i.e. a high load. The MOS is computed on a true end-to-end basis, including the application layer (i.e. encoding delay and playout buffer delays and losses). The figure shows concrete benefits for the three algorithms, with proactive VQ performing slightly better than the specialized version for CBR traffic (gest) for most of the UEs. Both in turn fare sensibly better than the basic VQ. The original algorithm proposed in [5] for HY-CART, instead, has a low performance, comparable to that of the UL-VoIP scheduler. Note that, in this scenario, proactive VQ does not bring significant improvements. This is because, when the load is high, it is likely that the UE queues are always backlogged. Therefore, the additional PDUs estimated by proactive VQ have a small probability to be taken into account by a scheduler such as HY-CART, which sizes its SGs on urgent data only. The results are confirmed by Figure 7 and Figure 8, which show the end-to-end delay and the queuing delay at the UE respectively. Proactive VQ warrants the lowest delay. On the other hand, the basic VQ entails a lower delay than queue estimation for CBR traffic. This is because, as the former has a less precise knowledge of the UE buffer (due to the absence of any hypothesis regarding traffic patterns), it sometimes overestimates the urgent data at the front of the queue. As a consequence, it makes HY-CART give away larger SGs, which reduces the one-way delay, albeit without necessarily increasing the MOS.

V. CONCLUSIONS

In this paper, algorithms for virtual queueing, i.e. for reconstructing the state of the UE queues at the NodeB, were proposed. We designed a basic algorithm that works without any assumption on the traffic profile. A more accurate algorithm has instead be devised for periodic traffic, such as VoIP. The above algorithm allow a scheduler to get full knowledge of the uplink queues, thus enabling deadline-based scheduling and resource allocation in HSUPA. The results show concrete benefits for using advanced VQ techniques (e.g., both the specialized VQ for periodic traffic and its proactive extension).
with respect to the basic version. Furthermore, using deadline-based schedulers, such as HY-CART, even with the basic version of VQ, improves significantly over traditional scheduling for VoIP in HSUPA when the cell load is high.

This work can be extended in several directions. First of all, the alert reader will have noticed that the algorithms of sections 3.B and 3.C can be easily generalized to other types of traffic (e.g., voice codecs that transmit at different rates during on and off periods). It could be possible to design versions of them suitable for compressed video traffic, where a stochastic description of the frame length of each frame type in a Group of Pictures might be exploited. A research direction being explored at the time of writing is how to infer flow profile information online, from a sequence of packets received by the NodeB. This would make it possible to use the basic VQ as a bottom line, and switch to specialized versions when enough information is gathered.

REFERENCES