

# An Adaptive Solution for Wireless LAN Distributed Power Saving Modes

Daniel Camps Mur<sup>\*</sup>, Xavier Pérez-Costa, Sebastia Sallent Ribes

*NEC Europe Laboratories, Germany, and Technical University of Catalonia (UPC), Spain*

---

## Abstract

The current trend to incorporate the Wireless LAN technology in increasingly smaller mobile devices poses new challenges, in terms of QoS and power consumption requirements, for the design of such devices. IEEE 802.11 and 802.11e define mechanisms to address these challenges but do not provide any guidance about how to design the algorithms required to make use of them to achieve an optimum performance. The work presented in this paper focuses on the design of an adaptive algorithm for the distributed power saving mechanisms of Wireless LANs when facing the challenge of providing QoS to devices in a power save mode. Our main contributions are i) an analytical model that captures the dependencies between the QoS experienced and the configuration of the Wireless LAN distributed power saving mechanisms, ii) a generic algorithm based on the steepest descent method that, using only information available in the MAC layer, adapts to the applications' characteristics and configures a power saving mechanism in order to provide a satisfactory QoS experience, iii) an analysis of the convergence properties of the proposed algorithm that provides the optimum values of its configurable parameters, and iv) a thorough simulative study that demonstrates the suitability of our adaptive solution in today's typical Wi-Fi deployments and its advantages in front of existent solutions in the state of the art.

*Key words:* Wireless LAN, QoS, Power Saving, Adaptive Configuration, Steepest Descent Algorithm

---

## 1. Introduction

The increasing popularity of wireless Internet access in mobile devices like PDA's or mobile phones is fostering their incorporation of the widely spread Wireless LAN (WLAN) technology. These devices though pose two fundamental challenges that have to be addressed in order to achieve a satisfactory user experience. First, users expect to be able to run applications that have stringent QoS requirements, e.g. VoIP. Second, users expect talk and standby times similar to those of cellular phones.

In order to address the need for QoS provisioning, IEEE developed the 802.11e [2] standard. This standard defines the Hybrid Coordination Function (HCF), that includes two different access methods: a contention-based channel access method called the Enhanced Distributed Channel Access (EDCA) and a contention-free channel access method referred to as HCF Controlled Channel Access (HCCA). While EDCA is a distributed scheme that provides prioritized QoS, HCCA is a centralized scheme that allows parameterized QoS provisioning. A thorough overview of the 802.11e QoS enhancements can be found in [4].

Regarding power saving mechanisms to extend battery life, IEEE 802.11 defines a power save mode that allows stations to switch off their radio during inactivity periods in order to save power. In the rest of the paper we will refer to this power save mode as *802.11 power save mode*. IEEE 802.11e defines an en-

---

<sup>\*</sup> Corresponding author. Address: NEC Europe Laboratories, Kurfuersten-Anlage 36, Heidelberg, Germany.

*Email addresses:* camps@neclab.eu (Daniel Camps Mur), perez@neclab.eu (Xavier Pérez-Costa), sallent@mat.upc.edu (Sebastia Sallent Ribes).

hancement of the 802.11 power save mode, Automatic Power Save Delivery (APSD), that takes advantage of the QoS mechanisms of 802.11e in order to provide an improved QoS experience when this power saving mode is used. Two modes of operation are available under APSD: *Unscheduled* and *Scheduled*. *Unscheduled* APSD (U-APSD) can be used only by stations accessing the channel using EDCA while *Scheduled* APSD (S-APSD) can be used with both access mechanisms, EDCA and HCCA.

New mobile devices incorporating 802.11e functionalities are more likely to include first the *distributed* mechanisms of 802.11e, i.e., EDCA and U-APSD, than the *centralized* ones, i.e., HCCA and S-APSD. This can be seen for instance in the fact that the Wi-Fi<sup>TM</sup> Alliance [3] has released first the Wi-Fi<sup>TM</sup> Multimedia (EDCA) and the WMM Power Save<sup>TM</sup> (EDCA plus U-APSD) certifications while HCCA and S-APSD certifications are being deferred. Based on this, we focus in this paper in the distributed mechanisms defined for Wireless LANs when facing the challenge of providing a seamless QoS experience to devices in a power save mode.

In this paper we design and analyze a generic solution to adaptively configure distributed power saving mechanisms in order to meet the expected QoS required by the applications using only information available at the MAC layer. Previous work in the area of power saving for WLANs has mainly targeted non real-time applications. In [7] and [8] different algorithms are proposed that adapt the waking up pattern of the stations assuming web-like traffic in the downlink. In [9] a power saving manager is defined that reduces power consumption during inactivity periods. Our work is orthogonal to many of these proposals in that we aim at trading off QoS and power consumption when there are active applications, not during inactivity periods, and target both real-time and non-real time traffic. In [10] several delivery strategies in the Access Point (AP) for U-APSD are studied but no U-APSD algorithm is proposed since only symmetric VoIP conversations and FTP sessions are considered which do not require one. In our previous work [5,6], we first analyzed the effect that 802.11 power save mode and U-APSD have on the QoS perceived by the application layer, and then in [11,12] considered the problem of adaptively configuring 802.11 power save mode and U-APSD providing heuristic solutions for each case.

The paper at hand extends our previous work and the results already existing in the literature by: i) Providing an analytical model of the effect of the WLAN distributed power saving mechanisms on the delay and

jitter experienced by applications, ii) designing an adaptive algorithm based on the steepest descent method that can be used to configure a WLAN distributed power saving mechanism such that QoS is provided without requiring knowledge from the applications, iii) analytically studying the convergence properties of the proposed algorithm and iv) evaluating by means of simulations the performance benefits of the proposed adaptive algorithm in front of existing solutions which require knowledge about the applications' characteristics.

Throughout this paper a basic knowledge of the 802.11 power saving mode and of 802.11e EDCA and U-APSD is assumed. An overview of these functionalities can be found in [4] for 802.11e EDCA and in [6] for 802.11 power save mode and U-APSD.

The rest of this paper is structured as follows. In Section 2 the need for an adaptive solution to configure distributed power saving mechanisms based only on information available at the MAC layer is motivated. Section 3 presents an analytical model of the effect of distributed power saving mechanisms on QoS. The requirements described in Section 2 and the model derived in Section 3 are used in Section 4 to design an algorithm that adapts to applications characteristics. The properties of the adaptive algorithm are analytically studied in Section 5 and a thorough performance evaluation is provided in Section 6. Finally, Section 7 summarizes the results and concludes the paper.

## 2. Problem Statement

In order to analyze the effect that distributed power saving mechanisms (U-APSD and 802.11 power save mode) have on the QoS perceived by applications, a deeper understanding on these schemes is needed. The main idea behind these mechanisms is the following. To increase battery life, stations switch off their radio transmitter and receiver to a sleep state of low power consumption whenever they have no pending transmissions in uplink or downlink. In the downlink direction (AP→station), the AP buffers the frames addressed to the power saving stations and only delivers them when the stations wake up and generate explicit requests for their transmission. In this paper we refer to these requests as *trigger* frames. The procedure of sending regular data frames in the uplink (station→AP) is not altered by the power saving mechanisms. The current WLAN distributed power saving schemes, i.e 802.11 power save mode and U-APSD, differ in the actual way the stations discover whether they have frames buffered in the AP's power save buffer and on the kind of frames that can

be used as trigger frames. In 802.11 power save mode, stations in power save mode wake up regularly to listen to Beacon transmissions and use the Traffic Indication Map (TIM) information to check whether there are any frames addressed to them buffered at the AP. The trigger frames issued by the stations are signaling frames called Power Save Polls (PS-Polls) and upon the reception of one poll the AP delivers one buffered frame to the station. A station can realize whether further frames are buffered in the AP by checking the More Data bit in the WLAN header of the received packets and issue PS-Polls accordingly.

In the case of U-APSD, the 802.11 power save mode capabilities are extended and enhanced in several ways. First, U-APSD allows for a reduction of the signaling introduced in 802.11 power save mode in a twofold manner: i) in addition to signaling frames, called QoS Nulls, data frames sent in the uplink by a station can be also configured to act as trigger frames and ii) a single trigger frame can retrieve from the AP not only one but up to a certain amount of buffered packets<sup>1</sup>. Second, the U-APSD mechanism allows a station to enable or disable U-APSD on a per Access Category (AC) basis.

Thus, it can be seen that the QoS experienced by applications in the downlink will become highly dependent on the algorithms that decide when to poll an AP in order to retrieve the buffered frames. On the one hand, if a station does not generate enough trigger frames, an application can suffer from an excessive delay in the downlink direction. On the other hand, if a station transmits too many unnecessary triggers, power saving is penalized and the congestion level in the network increases. As usual, the 802.11 and 802.11e standards leave open the algorithms to generate trigger frames to allow vendor differentiation.

Different approaches to solve this problem are considered today by WLAN mobile devices vendors. These approaches can be mainly classified into static and dynamic cross-layer approaches. Static approaches aim at bounding the delay of applications in the downlink using a *fixed polling interval*. Such a static approach though can not be optimal if different applications with different QoS requirements are used in the same mobile device. Dynamic cross-layer approaches can overcome this problem by re-configuring the MAC's layer polling interval every time a new application starts in order to meet its specific requirements. However, several issues need to be considered that limit the suitability of the

<sup>1</sup> The Max\_SP\_Length parameter configured at association defines how many packets the AP can deliver upon receiving a trigger frame: 2,4,6 or All

cross-layer approach for multi-purpose WLAN devices in practice:

- No generic framework is available for the application layer to become aware of the QoS requirements of the applications.
- No generic interface is available for communication between the application layer and the MAC layer.

Recently, the UPnP Forum [13] has started some work in the direction of achieving a standard cross-layer communication framework. However, this is a complex task that will not be completed in the near future because it requires the agreement of both software and hardware vendors in a technical specification and a certification program in order to ensure interoperability. Currently, due to the absence of a generic cross-layer framework, the dynamic cross-layer solution is not scalable because new functionality has to be added in the devices every time a new application needs to be supported.

The solution proposed in this paper solves the issue of dynamically configuring a Wireless LAN distributed power saving mechanism without requiring any cross-layer communication. In Section 4 we describe our proposed solution based only on information available at the MAC layer.

### 3. Modeling the effect of distributed power saving mechanisms on applications with QoS requirements

This section introduces an abstracted model that will allow us to quantify the effect that distributed power saving mechanisms have on the delay and jitter experienced in the downlink direction (AP→station) by applications with QoS requirements. The dependencies derived with this model will be used in the next section to design an algorithm that adapts to the traffic characteristics of the applications in order to meet their QoS needs. Although our focus is on guaranteeing that applications with QoS requirements can be run satisfactorily in mobile devices using a Wireless LAN distributed power saving mode, we will also show that the same algorithm can improve the performance of applications with lower QoS requirements, e.g., Web browsing and FTP downloads.

The main characteristics of Wireless LAN distributed power saving mechanisms are:

- i) Frames addressed to power saving stations are buffered in a network entity.
- ii) Stations are regularly informed through signaling messages about whether frames have been buffered for them.
- iii) Stations can request the delivery of their buffered

frames at any time by generating signaling triggers and in the U-APSD case also by reusing data frame transmissions in the uplink.

- iv) Stations' uplink data transmissions are not affected by the power saving mechanisms.

In order to simplify the derivation of an analytical model of the delay and jitter introduced in the downlink by the Wireless LAN distributed power saving mechanisms we make two approximations. The first approximation is that the traffic characteristics in the downlink of applications with QoS requirements can be modeled as a stream of packets, of fixed or variable size, that arrive at the AP with a constant interarrival time. Note that codec-based applications would fit in this model. In Section 5 we relax this assumption by considering jitter and drops in the incoming stream. The second approximation is that the delay experienced by a downlink frame is the time since the data frame was inserted in the AP's buffer until a station sent the corresponding trigger frame, i.e., the time to get access to the channel is negligible compared to the buffering time and the station trigger generation interval. This last approximation holds true when the level of congestion in the channel is moderated which is the working area of interest when QoS requirements need to be satisfied. The effect of congestion in the WLAN will be further studied in Section 6.

Two different cases are distinguished in the analysis. The case where uplink data frames can not be used as trigger frames (802.11 power save mode) and the case where uplink data triggers can be used as trigger frames (U-APSD).

### 3.1. No uplink data triggers

Let us consider the case where downlink frames arrive at the buffering entity with an interarrival time represented by  $\Delta_{DL}$  and stations in power saving poll the AP with a trigger interval equal to  $\Delta_{trig}$ .<sup>2</sup>

In order to model the delay experienced by the downlink stream the following can be derived. Given a trigger frame sent by a station and assuming that only one frame was retrieved from the AP as a result of that trigger, the delay experienced by this frame can be expressed as  $d(k) = t_{trig}(k) - t_{arr}(k)$ . Where  $t_{arr}(k)$  represents the arrival time of the  $k$ -th downlink data at

the AP, and  $t_{trig}(k)$  represents the reception time of the trigger sent by the station.

To understand the dynamics of  $d(k)$  we consider the delay experienced by a reference frame in the downlink,  $d_0$ , and observe that no delay above the station's polling interval,  $\Delta_{trig}$ , can be obtained in our model. Hence the delay experienced by the downlink frames can be expressed like:

$$d(k) = (d_0 + k\epsilon) \bmod \Delta_{trig} \quad (1)$$

Where  $k$  represents the  $k$ -th downlink frame arrived at the AP after the reference frame and  $\epsilon = \Delta_{trig} - \Delta_{DL}$ , represents the difference between the polling interval used by the station and the interarrival time of the downlink stream.

In order to prove Equation 1 consider that the  $k$ -th downlink frame is retrieved by the  $l$ -th trigger frame generated by the station. Applying the modulo function:

$$\begin{aligned} d(k) = t_{trig}(k) - t_{arr}(k) &= d_0 + l\Delta_{trig} - k\Delta_{DL} \leftrightarrow \\ &\leftrightarrow d(k) + (k-l)\Delta_{trig} = d_0 + k\epsilon \leftrightarrow \\ &\leftrightarrow d(k) = (d_0 + k\epsilon) \bmod \Delta_{trig} \end{aligned}$$

Equation 1 can be interpreted in the following way. First, it can be seen that the delay experienced by the downlink stream increases or decreases linearly with a slope equal to  $\epsilon$ . Second, the downlink stream never experiences a delay above the polling interval used in the station,  $\Delta_{trig}$ , resulting in a saw-tooth pattern. Figure 1(a) shows the downlink delay obtained in a simulation experiment<sup>3</sup> where a single WLAN station using U-APSD receives a CBR stream in the downlink. The results corroborate the correctness of Equation 1.

Based on Equation 1, the jitter in the downlink direction can be expressed as  $|j(k)| = |d(k) - d(k+1)| = \left| \left\lfloor \frac{d(k)+\epsilon}{\Delta_{trig}} \right\rfloor \Delta_{trig} - \epsilon \right|$ . The following bounds can be obtained considering  $|\epsilon| < \Delta_{trig}$ <sup>4</sup>:

$$|j(k)| = \begin{cases} |\epsilon| & \text{if } d(k) < \Delta_{DL}, d(k) \geq |\epsilon| \\ \Delta_{trig} - |\epsilon| & \text{if } d(k) \geq \Delta_{DL}, d(k) < |\epsilon| \end{cases} \quad (2)$$

Where the two different conditions on  $d(k)$  represent the cases of  $\epsilon \geq 0$  and  $\epsilon < 0$  respectively. Figure 1(a) depicts the downlink jitter experienced in the same experiment previously described. Notice in the figure how the spikes in jitter match the saw-tooth pattern observed in the downlink delay, and correspond to the instants

<sup>2</sup> In 802.11 PSM a station would generate extra PS-Polls when a frame with the More Data bit set is received. In the presented analysis we collapse these extra triggers into a single one, which is the case of U-APSD.

<sup>3</sup> The simulation framework used in this paper is detailed in Section 6.

<sup>4</sup> Note that this is the usual region of operation of an adaptive power save algorithm as the one presented in section 4.

where the station generates a trigger frame retrieving more than one data frame from the AP.

### 3.2. Uplink data triggers

Consider now the case where a station reuses uplink data frames as triggers. In this case a periodic trigger generation solution turns out not to be optimal since trigger frames would be generated more often than  $\Delta_{trig}$ . Thus, in order to avoid the introduction of unnecessary signaling, a *re-scheduling* mechanism should be used as it was proposed in [6]. This solution consists in scheduling a periodic generation of signaling triggers and re-scheduling the pending signaling trigger transmission every time a data trigger is sent in the uplink.

In order to model this solution, let us consider a WLAN station generating data triggers in the uplink every  $\Delta_{UL}$ , with  $\Delta_{UL} > \Delta_{DL}$  and  $\Delta_{UL} > \Delta_{trig}$  since otherwise the re-scheduling mechanism would avoid the transmission of signaling triggers, obtaining then a trigger generation pattern like in the previous section<sup>5</sup>. Thus, besides the uplink data triggers, a station schedules the transmission of signaling triggers every  $\Delta_{trig}$  as shown in Figure 2.

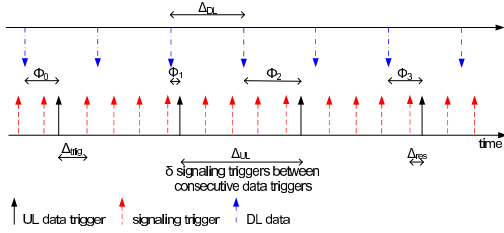


Fig. 2. Data and signaling trigger generation employing re-scheduling

Note that when the re-scheduling mechanism is applied,  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{trig}} \rfloor$  signaling triggers are always generated between two consecutive uplink data frames. In addition, the time between a signaling trigger and an uplink data trigger is  $\Delta_{res} = \Delta_{UL} \bmod \Delta_{trig}$ , with  $0 \leq \Delta_{res} < \Delta_{trig}$ .

Although the introduction of uplink data triggers increases the complexity of modeling the dynamic behaviour of the downlink delay  $d(k)$ , based on the results from the previous section it can be seen that the delay experienced by the downlink stream is still bounded by  $\Delta_{trig}$  and linearly increases or decreases between uplink data frames, with a slope equal to  $\epsilon = \Delta_{trig} - \Delta_{DL}$ . Whenever an uplink data frame is sent in the uplink, the

<sup>5</sup> A practical example of this setting could be a Video application generating periodic RTCP updates in uplink or a Voice conversation using different codecs in uplink and downlink.

downlink delay varies according to  $\epsilon_{res} = \Delta_{res} - \Delta_{DL}$ . Figure 1(b) shows the same experiment considered in the previous section but adding a periodic generation of data triggers in the uplink. Note that although  $\Delta_{trig}$  is above  $\Delta_{DL}$ ,  $\Delta_{res}$  is not and therefore some uplink data triggers find no packets in the AP's power save buffer.

Regarding the jitter of the downlink stream, the following can be said. The jitter continues to be constant and equal to  $\epsilon$  between consecutive uplink transmissions, and varies according to  $\epsilon_{res}$  when a station sends an uplink data trigger. A new situation has to be considered though, if an uplink data trigger finds no data buffered in the AP. In this case, the jitter is  $|j(k)| = |d(k) - d(k+1)| = |d(k) - (d(k) + \Delta_{res} + \Delta_{trig} - \Delta_{DL})| = \Delta_{res} + \epsilon$ , which could be above the maximum value found in the previous section. Figure 1(b) shows the downlink jitter in the experiment previously described.

Another interesting metric to consider in this case is the amount of uplink data triggers actually retrieving data from the AP. The former is a measure of how efficiently a station reuses these data frames as triggers. Ideally, we would like all data frames to be reused as triggers in order to minimize the amount of generated signaling triggers.

From Figure 2 it can be observed that an uplink data trigger retrieves a frame from the AP if and only if  $\phi_i < \Delta_{res}$ . Where  $\phi_i$  represents the time between an uplink data trigger transmission and the previous corresponding downlink frame that arrived at the AP.

In order to find how many uplink data triggers will find a downlink frame to download, it can be seen that the compound sequence of uplink triggers and downlink frames is periodic, with period  $T = lcm(\Delta_{UL}, \Delta_{DL})$ , where *lcm* stands for least common multiple.

Being such sequence periodic, the sequence of  $\phi_i$  values will also be periodic, with period  $N = \frac{T}{\Delta_{UL}}$ . Indeed, the actual  $\phi_i$  values can be expressed as:

$$\phi_i = (\phi_0 + i\Delta_{UL}) \bmod \Delta_{DL} \quad (3)$$

Where  $\phi_0$  is any of the values of the sequence taken as initial reference.

Since Equation 3 is a linear congruence, the *linear congruence theorem* can be applied to find that there will be integer solutions on  $i$  if and only if  $gcd(\Delta_{UL}, \Delta_{DL})$  divides  $\phi_i - \phi_{min}$ . Where *gcd* stands for greatest common divisor and  $\phi_{min} = \phi_0 \bmod gcd(\Delta_{UL}, \Delta_{DL})$ . Therefore, the elements contained in the set  $\{\phi_i\}$  are the same elements contained in the set  $\{\phi_j\}$  defined as:

$$\phi_j = \phi_{min} + j \cdot gcd(\Delta_{UL}, \Delta_{DL}) \quad j = 0 \dots N - 1 \quad (4)$$

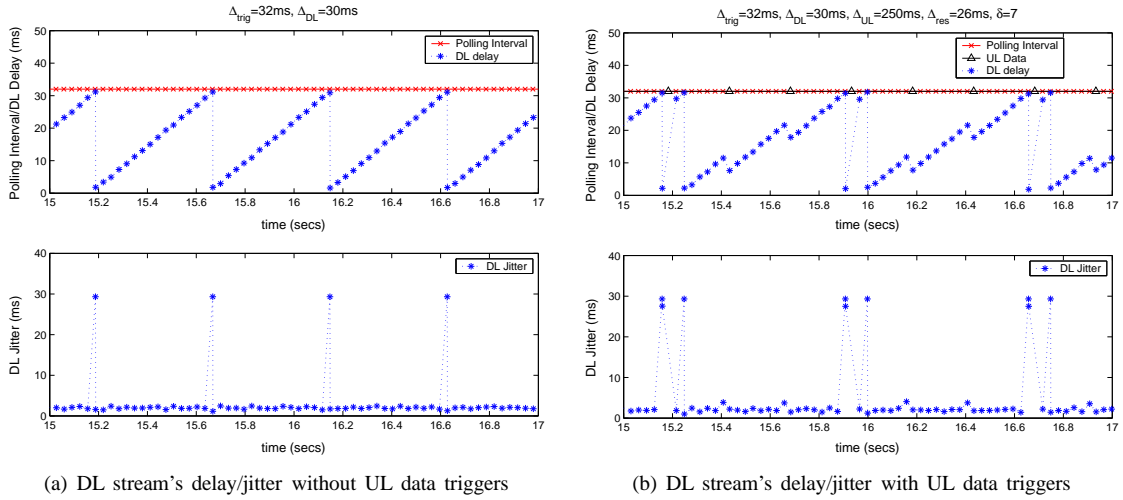


Fig. 1. DL stream's delay/jitter under a constant polling interval,  $\Delta_{trig}$

Recalling now that uplink data triggers will find data frames buffered in the AP's power save buffer if and only if  $\phi_i < \Delta_{res}$  and considering Equation 4, the number of uplink data triggers that, within a period  $T$ , retrieve frames from the AP is:

$$M = \lceil \frac{\Delta_{res} - \phi_{min}}{gcd(\Delta_{UL}, \Delta_{DL})} \rceil \quad 0 \leq M \leq N \quad (5)$$

#### 4. Algorithm Design

Sections 2 and 3 have motivated the need to configure the WLAN distributed power saving mechanisms according to the traffic characteristics of the applications based only on information available at the MAC layer and have shown how the delay and jitter experienced by an application in the downlink depend on the polling interval used by a station. In this section, to overcome the limitations of a static approach and avoid the problems that arise when using cross-layer mechanisms, we propose an adaptive algorithm that estimates the interarrival time of a downlink stream,  $\Delta_{DL}$ , and generates signaling trigger frames accordingly.

The reason for choosing this approach is that, according to the model derived in Section 3, if we would poll the AP at intervals equal to the downlink interarrival time,  $\Delta_{DL}$ , the downlink delay would be kept constant, bounded to the  $\Delta_{DL}$  value, and jitter would be eliminated. Note that the estimated value could also be used to poll the AP at multiples of the interarrival time depending on whether a reduction on delay/jitter or signaling load is preferred. The larger the polling interval the lower the signaling load but the larger the delay and jitter. Thus, an algorithm that estimates the downlink in-

terarrival time allows to efficiently address the trade-off between delay/jitter and signaling load.

We define the following objectives for the design of our adaptive algorithm:

1. Bound the delay of the downlink frames according to their interarrival time at the AP.
2. Minimize the amount of signaling load required to achieve 1.
3. Rapidly adapt to variations in the interarrival time.
4. Minimize the complexity, in terms of the number of updates required to follow changes in the interarrival time.

Given that the downlink interarrival time is unknown at the MAC layer of a station and can not be provided by the AP using standard mechanisms, an algorithm in a station that could meet the proposed objectives would be one that counts all the frames received from the AP,  $n_{fr\_rcvd}$ , during a time window,  $T_w$ , to afterwards update the downlink interarrival time estimation by computing  $\frac{T_w}{n_{fr\_rcvd}}$ . The bigger  $T_w$ , the better the updated value of the estimation since more frames will be received from the AP. The problem of increasing  $T_w$  though, is that sudden changes in the value of the interarrival time can not be followed immediately. In order to achieve better precision using small time windows, a moving average could be considered, but the problem again would be to quickly follow changes of the input because average filters introduce memory.

Therefore, instead of using a fixed  $T_w$  value, we propose an algorithm that updates the current value of the estimation according to the occurrence of specific events defined based on the following observations:

- If a station generates trigger frames at the same rate

that downlink frames arrive at the AP, each trigger frame will always result in a single frame delivered by the AP.

- If a station generates trigger frames at a rate that is above the rate of arrivals at the AP, each trigger frame will result in either one or no frames delivered by the AP.
- If a station generates trigger frames at a rate below the rate of arrivals at the AP, each trigger frame will result in either one or more frames delivered by the AP.

According to the above observations, two types of events can be defined that univocally identify whether the estimation in the station is above or below the actual interarrival time of the downlink packets at the AP.

- *No Data* event: If a trigger is sent by a station and no frame is delivered by the AP. This event implies that the station holds an estimation below the actual downlink interarrival time.
- *More Data* event: If a trigger is sent by a station and more than one frame is delivered by the AP. This event implies that the station holds an estimation above the actual downlink interarrival time.

These two events can be recognized at the MAC layer of the stations using any of the distributed power saving mechanisms available today. The *No Data* event can be recognized because the AP delivers an empty frame, QoS Null, when a trigger is received but no data is buffered. A more efficient method to signal this event has been proposed in [14], where the need to send explicit signaling is avoided by making use of a bit available in the ACK frames. Regarding the *More Data* event, it can be recognized by simply counting the number of frames received before getting a frame with the EOSP bit<sup>6</sup> set to 1 in the case of U-APSD, or monitoring the *More Data* bit in the case of 802.11 power save mode.

An algorithm that aims to estimate the downlink interarrival time can be therefore proposed that increases its current estimation whenever a *No Data* event occurs, and decreases its current estimation whenever a *More Data* event occurs. The problem hence is to investigate how the current value of the estimation,  $\Delta_{trig}(n)$ , has to be updated every time an event occurs in order to converge to the actual downlink interarrival time. Where  $\Delta_{trig}(n)$  represents the polling interval used by a station after the  $n$ -th estimation update,  $n \in \mathbb{Z}$ .

Our proposal is to use an *steepest descent* algorithm to update the estimation and converge to  $\Delta_{DL}$ . The idea behind the steepest descent algorithm is to esti-

mate at each iteration the error between the current estimation,  $\Delta_{trig}(n)$ , and the actual downlink interarrival time,  $\Delta_{DL}$ , and use this information to update  $\Delta_{trig}(n)$  in order to get closer to  $\Delta_{DL}$ .

A steepest descent algorithm for a single variable, in this case the variable  $\Delta_{trig}$ , can be expressed as:

$$\Delta_{trig}(n+1) = \Delta_{trig}(n) - \gamma \frac{\partial}{\partial \Delta_{trig}} J(\Delta_{trig}(n)) \quad (6)$$

Where  $\Delta_{trig}(n)$  represents the current value of the estimation,  $\gamma > 0$  is the adaptation step that controls the speed of convergence and  $J(\Delta_{trig})$  is a derivable and convex error function that has a minimum at the point where the algorithm converges. To avoid converging to a value other than  $\Delta_{DL}$  we select an error function that has only one minimum:

$$J(\Delta_{trig}(n)) = (\Delta_{trig}(n) - \widehat{\Delta_{DL}})^2 \quad (7)$$

Where  $\widehat{\Delta_{DL}}$  is an estimator of  $\Delta_{DL}$ , that has to be used because the actual value of  $\Delta_{DL}$  is the unknown value we are looking for. Thus, the successful convergence of the algorithm will depend on whether  $\widehat{\Delta_{DL}}$  is a good estimation of  $\Delta_{DL}$ . We consider:

$$\widehat{\Delta_{DL}} = \frac{\Delta_t(n)}{n\_fr\_rcvd(n)} \quad (8)$$

Where  $\Delta_t(n)$  is the time measured at a station between the current event and the previous one and  $n\_fr\_rcvd(n)$  is the number of frames that a station has retrieved from the AP during  $\Delta_t(n)$ , using a polling interval  $\Delta_{trig}(n)$ .

Based on the previous equations, our proposed algorithm will update the estimation every time an event occurs according to:

$$\Delta_{trig}(n+1) = \Delta_{trig}(n) - \gamma \left( \Delta_{trig}(n) - \frac{\Delta_t(n)}{n\_fr\_rcvd(n)} \right) \quad (9)$$

The routine described in Algorithm 1 illustrates our proposed implementation. This routine is executed by the power saving stations every time a service period (SP<sup>7</sup>) completes.

Several points deserve special attention in Algorithm 1. First, it can be observed in steps 16 and 25 that a different value for the adaptation step is used depending on whether the station experiences a *More Data* event,

<sup>6</sup> The End Of Service Period (EOSP) bit signals the end of a Service Period and allows a station to go back to sleep mode in U-APSD.

<sup>7</sup> We extend the U-APSD definition of Service Period in the context of this paper to be used also in the case of 802.11 power save mode. Service Period is defined as the period of time that starts when a station sends a trigger frame and ends when a frame with EOSP=1 in case of U-APSD or MD=0 in case of legacy power save mode is received from the AP.

**Algorithm 1** Routine executed in the station to update its current polling interval,  $\Delta_{trig}(n)$

---

```

 $\Delta_t \leftarrow$  Time between the current event and the previous one
 $n\_fr\_rcvd \leftarrow$  Number of frames received between the current
event and the previous one
 $m \leftarrow$  Number of frames received during the Service Period (SP)
 $\Delta_{trig} \leftarrow$  Polling interval used in the station
1: if  $m = 1$  then
2:    $n\_fr\_rcvd \leftarrow n\_fr\_rcvd + 1$ 
3: else if  $m, n\_fr\_rcvd = 0$  and last trigger was signaling then
4:    $\Delta_{trig}(n+1) \leftarrow \beta \Delta_{trig}(n)$ 
5:    $n\_fr\_rcvd \leftarrow 0$ 
6: else if  $m > 2$  then
7:   if  $consec\_long\_burst > cons\_long\_burst\_MAX$  then
8:      $\Delta_{trig}(n+1) \leftarrow \frac{\Delta_{trig}(n)}{m}$ 
9:      $consec\_long\_burst \leftarrow 0$ 
10:  else
11:     $consec\_long\_burst \leftarrow consec\_long\_burst + 1$ 
12:  end if
13: else
14:  if  $m > 1$  then
15:    if  $More\_Data\_update$  is  $TRUE$  then
16:       $\Delta_{trig}(n+1) \leftarrow \Delta_{trig}(n) - \gamma_{MD}(\Delta_{trig}(n) - \frac{\Delta_t}{n\_fr\_rcvd})$ 
17:      Reschedule next signaling trigger transmission
18:    else
19:       $More\_Data\_update \leftarrow TRUE$ 
20:    end if
21:     $No\_Data\_update \leftarrow FALSE$ 
22:     $n\_fr\_rcvd \leftarrow 0$ 
23:  else if  $m = 0$  and last trigger was signaling then
24:    if  $No\_Data\_update$  is  $TRUE$  then
25:       $\Delta_{trig}(n+1) \leftarrow \Delta_{trig}(n) - \gamma_{ND}(\Delta_{trig}(n) - \frac{\Delta_t}{n\_fr\_rcvd})$ 
26:      Reschedule next signaling trigger transmission
27:    else
28:       $No\_Data\_update \leftarrow TRUE$ 
29:    end if
30:     $More\_Data\_update \leftarrow FALSE$ 
31:     $n\_fr\_rcvd \leftarrow 0$ 
32:  end if
33: end if
34:  $m \leftarrow 0$ 

```

---

$\gamma_{MD}$ , or a No Data event,  $\gamma_{ND}$ . The reason for doing so, is that the value of  $\gamma$  can be used to control whether the estimation,  $\Delta_{trig}(n)$ , converges to  $\Delta_{DL}$  remaining always in the More Data zone, i.e., the zone where  $\Delta_{trig}(n) > \Delta_{DL}$ , or in the No Data zone, i.e., the zone where  $\Delta_{trig}(n) < \Delta_{DL}$ <sup>8</sup>. The choice of configuring the algorithm to operate in the More Data zone or in the No Data zone represents a trade-off between the possibility of generating an excess of signaling load, if it operates in the No Data zone, or temporarily allowing a delay for some frames above the downlink interarrival

<sup>8</sup> A more detailed definition of the More Data and No Data zones is provided in Section 5.

time, if it operates in the More Data zone. In order to avoid useless signaling that might increase the congestion in the network and harm the power consumption, we propose to configure the algorithm to operate in the More Data zone.

Second, it can be observed between steps 3 and 12, that a special behaviour is considered if no frames have been received between the current event and the previous one ( $n\_fr\_rcvd = 0$ ), or if more than two frames have been retrieved from the AP during this SP ( $m > 2$ ). Both cases represent the situation of having a current estimation,  $\Delta_{trig}(n)$ , far from  $\Delta_{DL}$ . In the former case the current value of the estimation is below the downlink interarrival time but, since no frames have been retrieved between events, the station has no information to estimate  $\Delta_{DL}$ . In this case the current estimation is updated by doing  $\Delta_{trig}(n+1) \leftarrow \beta \Delta_{trig}(n)$ , where  $\beta > 1$ . In the latter case, the current estimation,  $\Delta_{trig}(n)$ , is above  $\Delta_{DL}$ . Thus, after applying a memory to reduce spurious cases ( $consec\_long\_burst$ ), the estimation is quickly decreased by doing  $\Delta_{trig}(n+1) \leftarrow \frac{\Delta_{trig}(n)}{m}$ . Note that if  $m > 2$  the station knows that it is holding an estimation at least  $m$  times above the downlink interarrival time.

Finally, the boolean variables introduced in steps 15 and 24,  $More\_Data\_update$  and  $No\_Data\_update$ , ensure that the estimation is updated always between consecutive events of the same zone (More Data zone or No Data zone). The reason for this is that a too short interval,  $\Delta_t$ , between events can occur when the value of the estimation moves from one zone to the other. As it will be seen in the next section, this could lead to a misleading estimation of the error between  $\Delta_{trig}(n)$  and  $\Delta_{DL}$ .

We consider now the case where EDCA is the channel access function in use and various applications, with different delay requirements, run in the station at the same time through different Access Categories (AC). In this case, the trigger generation algorithm would only generate triggers at the estimated rate of the application sending frames more often, i.e.  $min\{\Delta_{DL}[i]\}$ . The former is possible because the algorithm can rely on the More Data bit and Max\_SP\_Length capabilities of the existent power saving mechanisms in order to retrieve frames from other downlink streams. To make the overall system adaptive, a simple procedure can be implemented in the station that by keeping track of the frames received from each AC decides which is the AC sending frames more often. The station would then run the algorithm over the selected AC that would also be used to generate signaling triggers.

The complete cycle of activity in a station can be summarized in the following terms. When a station is idle no trigger frames are generated. Once a Beacon frame is received announcing the presence of buffered frames in the AP, a station starts generating trigger frames with an initial polling interval,  $\Delta_{trig}(0)$ , that is then dynamically adjusted by means of Algorithm 1. After sending a certain number of consecutive trigger frames receiving only empty frames from the AP a station considers that the application has finished and switches back to idle mode again.

Finally, we conclude this section discussing a simple heuristic that can be used to improve the performance of the algorithm when applications sending large packets are considered. Large packets in the application layer, e.g. like those generated by video codecs, are in many cases fragmented resulting in bursts of packets arriving at the AP which can mislead the algorithm into false More Data events. However, if the minimum MTU between the application generating packets and the AP is assumed to be known at the terminal, which with a reasonable probability can be considered to be the Ethernet MTU, the misleading effect of large packets can be reduced by remembering the size in the MAC layer of the packets downloaded during a service period, and considering as a single packet adjacent packets of size equal to the expected MTU<sup>9</sup>.

#### 4.1. Convergence to a multiple of the interarrival time: $\Delta_{trig} \rightarrow k\Delta_{DL}$

If a delay and jitter above  $\Delta_{DL}$  can be afforded, further power saving and signaling load enhancements can be obtained by modifying Algorithm 1 to converge to a multiple of the downlink interarrival time,  $\Delta_{trig} \rightarrow k\Delta_{DL}$ , where  $k$  is an integer bigger than 1. The larger the value of  $k$  can be set, the larger the potential increase in power saving and reduction of signaling load.

In order to achieve this, Equation 9 has to be modified such that the equilibrium state is reached when the trigger interval used by a station,  $\Delta_{trig}$ , equals the desired multiple of the interarrival time:

$$\Delta_{trig}(n+1) = \Delta_{trig}(n) - \gamma(\Delta_{trig}(n) - k \frac{\Delta_i(n)}{n_{fr\_rcvd}(n)}) \quad (10)$$

In addition, the *No Data* and *More Data* events generation, which trigger an update of the current polling

<sup>9</sup> This heuristic would fail if packets of size equal to the MTU minus overheads are generated by the applications, but we consider this probability to be small.

interval, also need to be adjusted accordingly. In this case, *No Data* events will be generated when less than  $k$  frames are received within one service period while *More Data* events will be generated when more than  $k$  frames are received within one service period.

In the rest of the paper we focus in the case of the polling interval converging to the downlink interarrival time,  $k = 1$ , since it is the one with the most stringent QoS requirements and thus, the most demanding for the proposed adaptive algorithm.

## 5. Algorithm Analysis

In this section we use the model presented in Section 3 in order to gain a deeper understanding on the properties of the adaptive algorithm presented in the previous section. First, we study under which conditions the algorithm converges. Second, the convergence speed of the algorithm is analyzed providing a worst case bound on the required number of iterations. Finally, we discuss how the performance of the algorithm degrades when the assumptions of the model do not hold.

### 5.1. Proof of Convergence

For the analysis, we define *convergence* as the process of making the estimation,  $\Delta_{trig}(n)$ , continuously approach at each iteration the actual downlink interarrival time,  $\Delta_{DL}$ , while always guaranteeing  $\Delta_{trig}(n) \geq \Delta_{DL}$ . Some of the reasons for preferring to keep  $\Delta_{trig}(n) \geq \Delta_{DL}$  have already been provided in Section 4. Additionally, further reasons will be given in this section. The same analysis presented here though, could be also applied if it would be preferred that the algorithm converges to  $\Delta_{DL}$  guaranteeing  $\Delta_{trig}(n) \leq \Delta_{DL}$ .

Throughout the analysis we refer to the set of values of the estimation,  $\Delta_{trig}(n)$ , that are above  $\Delta_{DL}$  as the *More data* zone, since only More Data events can be observed by the station in this zone. Similarly, the set of values of the estimation below  $\Delta_{DL}$  are referred to as the *No Data* zone. The different zones of convergence are depicted in Figure 3.

To demonstrate the convergence of the algorithm we will prove that if the current value of the estimation,  $\Delta_{trig}(n)$ , is above the downlink interarrival time,  $\Delta_{DL}$ , the next estimation update will result in a smaller value,  $\Delta_{trig}(n+1)$ , either above or equal to  $\Delta_{DL}$ . It is important to note though that when the algorithm starts the station's initial polling interval,  $\Delta_{trig}(0)$ , could be below the downlink interarrival time. If the initial value is

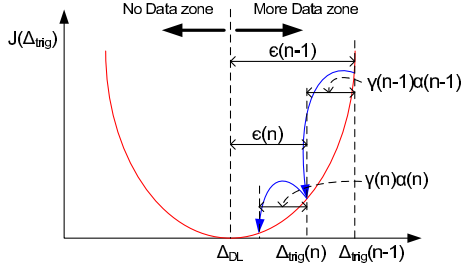


Fig. 3. Evolution of  $\epsilon(n)$  along the error function

below the downlink interarrival time, appropriate mechanisms can be designed to ensure that we will move to the More Data zone. The proof of convergence of this section guarantees that once we enter into the More Data zone we will not leave it anymore and at each iteration the estimation will get closer to the actual  $\Delta_{DL}$  value.

We start considering the more general case where the station reuses uplink data frames as triggers and the re-scheduling mechanism is used. Results regarding the no data triggers case can be obtained particularizing the presented analysis. The error incurred by a station at each estimation update is defined as  $\epsilon(n) = \Delta_{trig}(n) - \Delta_{DL}$ . Hence, if it holds that  $0 \leq \frac{\epsilon(n+1)}{\epsilon(n)} < 1 \forall n$ , the estimation will get closer to  $\Delta_{DL}$  every time it is updated. The former can be proved by separately demonstrating:

1.  $\frac{\epsilon(n+1)}{\epsilon(n)} < 1$ . This condition ensures that the error  $\epsilon$  reduces at each iteration.
2.  $\frac{\epsilon(n+1)}{\epsilon(n)} \geq 0$ . This condition ensures that the algorithm remains always in the same zone, either the More Data zone or the No Data zone.

We start proving that  $\frac{\epsilon(n+1)}{\epsilon(n)} < 1 \forall n$ . Figure 4 shows two consecutive events that update the estimation. In the figure,  $\Delta_{DL}$  represents the interarrival time of the downlink stream and  $\Delta_{trig}$  and  $\Delta_{UL}$  represent the generation interval of signaling and data triggers, respectively. Note that  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{trig}} \rfloor$  signaling triggers are sent between two consecutive data triggers in the uplink and the time between a signaling trigger and an uplink data frame is  $\Delta_{res} = \Delta_{UL} \bmod \Delta_{trig}$ .

In Figure 4 the two signaling triggers marked with a circle represent More Data events that trigger an estimation update. In the first event,  $d_{01}$  represents the time between the trigger frame sent by the station and the arrival time of the last frame retrieved by this trigger from the AP. Note that since this is a More Data event and the station polls the AP every  $\Delta_{trig}(n-1)$ ,  $0 \leq d_{01} < \epsilon(n-1)$ . An analog definition is used for  $d_{02}$  in the second event, the difference in this case is that  $0 \leq d_{02} < \epsilon(n)$ . In addition,  $\Delta_t$  represents the time

between the two consecutive events,  $\Delta'_t$  represents the time between the first of the events and the last uplink data trigger sent before the second event, and  $k$  represents the number of triggers sent after  $\Delta'_t$  until the second event occurred, with  $1 \leq k \leq \delta$ <sup>10</sup>.

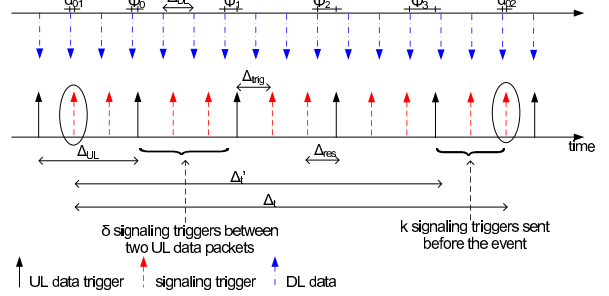


Fig. 4. Timing between consecutive events

Recalling now the definition of  $\epsilon(n)$ , Equation 9 and considering the current estimation in the More Data zone, it can be observed that  $\frac{\epsilon(n+1)}{\epsilon(n)} < 1 \forall n$  if and only if:

$$\Delta_{trig}(n+1) < \Delta_{trig}(n) \leftrightarrow \Delta_{trig}(n) > \frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$$

An upper bound on  $\frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$  expressed in terms of  $\Delta_{trig}(n)$  is needed in order to prove the previous statement. Considering again Figure 4,  $\frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$  can be upper bounded in the following way:

$$\frac{\Delta_t}{n\_fr\_rcvd} = \frac{\Delta'_t + k\Delta_{trig}}{\lfloor \frac{d_{01} + \Delta'_t}{\Delta_{DL}} \rfloor + k + 1} \leq \frac{\Delta'_t + k\Delta_{trig}}{\lceil \frac{\Delta'_t}{\Delta_{DL}} \rceil + k} \quad (11)$$

Where the temporal index  $n$  has been omitted for clarity. Thus,  $\Delta_{trig}(n) > \frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$  can be proved by comparing  $\Delta_{trig}(n)$  with the upper bound for  $\frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$  obtained in Equation 11:

$$\frac{\Delta'_t + k\Delta_{trig}}{\lceil \frac{\Delta'_t}{\Delta_{DL}} \rceil + k} < \Delta_{trig} \leftrightarrow \frac{\Delta'_t}{\Delta_{trig}} < \lceil \frac{\Delta'_t}{\Delta_{DL}} \rceil \quad (12)$$

Where the previous inequality is always satisfied because we are considering the case where  $\Delta_{trig}(n) > \Delta_{DL}$ .

So far it has been proved that the algorithm always updates the estimation in the correct direction when a More Data event is observed. Next, we prove that the adaptation step,  $\gamma$ , can ensure that the estimation,

<sup>10</sup>We assume in this case that the second More Data event is triggered by a signaling frame. The same analysis applies if this event is triggered by a data frame, replacing  $k\Delta_{trig}$  by  $\delta\Delta_{trig} + \Delta_{res}$ .

$\Delta_{trig}(n)$ , does not bounce between the More Data and the No Data zones, i.e.,  $\frac{\epsilon(n+1)}{\epsilon(n)} \geq 0$ .

Considering Equation 9, it can be seen that the *effective* step taken by the algorithm to update the current estimation is  $\gamma(n)\alpha(n)$ , with  $\alpha(n) = \Delta_{trig}(n) - \frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$ . From Figure 3 and assuming a value of the estimation in the More Data zone, it can be observed that the updated value of the estimation,  $\Delta_{trig}(n+1)$ , will remain in the More Data zone if and only if the step taken by the algorithm at this iteration is smaller than the current error, i.e.,  $\gamma(n)\alpha(n) \leq \epsilon(n)$ .

In order to obtain an upper bound on  $\gamma(n)$  that ensures  $\Delta_{trig}(n+1) \geq \Delta_{DL}$ , note first that based on the relations illustrated in Figure 4,  $\frac{\Delta_t(n)}{n\_fr\_rcvd(n)}$  can be expressed as:

$$\frac{\Delta_t(n)}{n\_fr\_rcvd(n)} = \Delta_{DL} + \frac{d_{02} - d_{01}}{n\_fr\_rcvd(n)} \quad (13)$$

Recalling the definition of  $\alpha(n)$  and the previous equation we can express  $\alpha(n)$  as:

$$\begin{aligned} \alpha(n) &= \Delta_{trig}(n) - \frac{\Delta_t(n)}{n\_fr\_rcvd(n)} = \\ &= \epsilon(n) - \frac{d_{02} - d_{01}}{n\_fr\_rcvd(n)} \end{aligned} \quad (14)$$

Considering now that  $0 \leq d_{01} < \epsilon(n-1)$  and that, from Equation 9,  $\epsilon(n-1)$  can be expressed as  $\epsilon(n-1) = \epsilon(n) + \gamma(n-1)\alpha(n-1)$ , the following upper bound on  $\alpha(n)$  can be obtained:

$$\alpha(n) < \epsilon(n) \frac{n\_fr\_rcvd(n) + 1}{n\_fr\_rcvd(n)} + \frac{\gamma(n-1)\alpha(n-1)}{n\_fr\_rcvd(n)}$$

Thus, the previous upper bound on  $\alpha(n)$  can be turned into a lower bound on  $\epsilon(n)$  and recalling the condition to keep the next value of the estimation in the More Data zone,  $\gamma(n)\alpha(n) < \epsilon(n)$ , a conservative  $\gamma(n)$  can be found,  $\gamma_{CONS}(n)$ , that keeps the estimation always in the desired zone of convergence:

$$\gamma(n) < \frac{n\_fr\_rcvd(n) - \gamma(n-1)\frac{\alpha(n-1)}{\alpha(n)}}{n\_fr\_rcvd(n) + 1} = \gamma_{CONS}(n) \quad (15)$$

Continuing with the previous reasoning and going back to Equation 14 while recalling that  $0 \leq d_{02} < \epsilon(n)$ , a lower bound on  $\alpha(n)$  can be obtained. Applying the same analysis to this bound, an aggressive bound on  $\gamma(n)$  that forces the estimation to jump from one zone to the other can also be obtained, being:

$$\gamma(n) > \frac{n\_fr\_rcvd(n)}{n\_fr\_rcvd(n) - 1} = \gamma_{AGGR}(n) \quad (16)$$

Although the previous bounds,  $\gamma_{CONS}(n)$  and  $\gamma_{AGGR}(n)$ , have been derived under the assumption of having the estimation in the More Data zone, Equations 15 and 16 hold true also in the No Data zone.

To illustrate how the algorithm updates the estimation while keeping always a value in the More Data zone, Figure 5(a) depicts the delay and polling interval experienced by a WLAN station using U-APSD that retrieves a CBR stream from the AP. The values of  $\gamma$  in the More Data and No Data zone,  $\gamma_{MD}$  and  $\gamma_{ND}$ , are constant and chosen below and above the corresponding  $\gamma_{CONS}$  and  $\gamma_{AGGR}$  thresholds. From the figure, it is worth noting how the time between events increases as the error in the estimation reduces, as predicted by the model introduced in Section 3. This property of the algorithm has the interesting consequence that, in the limit, the sequence of uplink triggers generated by the station and downlink frames arriving at the AP would synchronize, i.e.,  $\lim_{t \rightarrow \infty} d(k) = 0$ . Note though, that if the algorithm would converge from the No Data zone, the behaviour in the limit would be different, i.e.  $\lim_{t \rightarrow \infty} d(k) = \Delta_{DL}$ . Hence this is another reason to make the algorithm converge from the More Data zone.

It has been proved so far that  $\epsilon(n)$  reduces at each estimation update. In order to update the estimation though, the station has to keep observing new events. Hence, a new question arises that is whether the station will always observe enough events to make the estimation converge to  $\Delta_{DL}$ . We prove in the following paragraphs that:

1. If no data triggers in the uplink are considered, the proposed algorithm always converges to  $\Delta_{DL}$ .
2. If data triggers in the uplink are considered, the proposed algorithm can converge to a compact set of values around  $\Delta_{DL}$ . In this case though, all the algorithm's requirements described in section 4 are also fulfilled.

To prove the previous statements, the case where the station generates uplink data triggers is considered. The results obtained are afterwards particularized to the case with no data triggers in the uplink.

The *converged* values of  $\Delta_{trig}(n)$  are defined as those values that prevent the station from observing any further event. In order to find those values we define  $\phi_i$ ,  $0 \leq \phi_i < \Delta_{DL}$ , which represent the time difference between an uplink data trigger sent by a station and the last downlink arrival at the AP. See Figure 4 for a graphical representation.

As mentioned in Section 3, the compound sequence of uplink triggers and downlink frames in our model

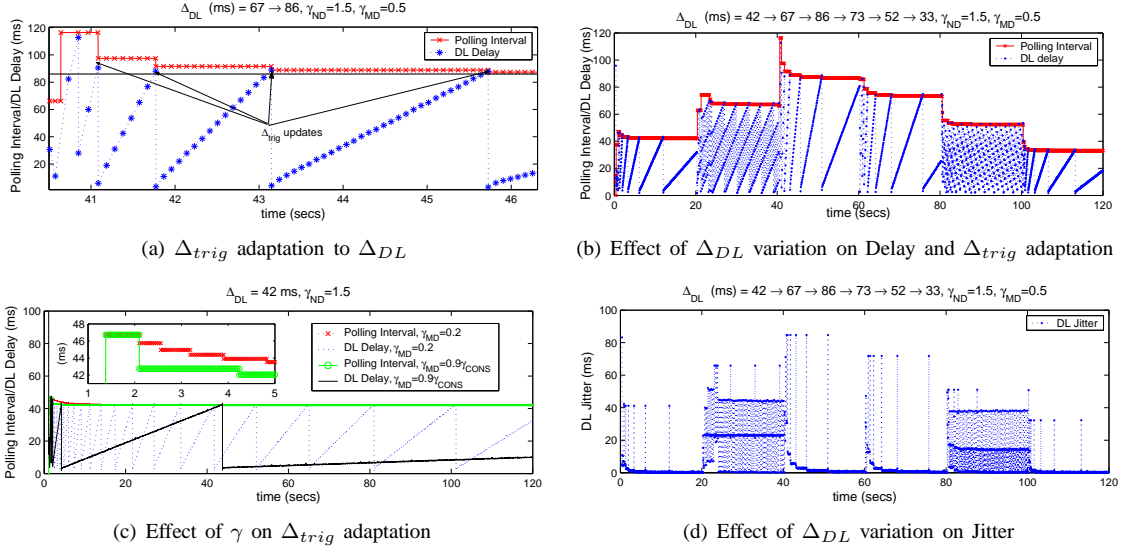


Fig. 5.  $\Delta_{trig}$  dynamics

is periodic, with period  $T = lcm(\Delta_{UL}, \Delta_{DL})$ .  $T$  can hence be understood as composed of  $N = \frac{T}{\Delta_{UL}}$  consecutive uplink subperiods, where each uplink subperiod is formed by an uplink data trigger and  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{trig}} \rfloor$  consecutive signaling triggers and is shifted  $\phi_i = (\phi_0 + i\Delta_{UL}) \bmod \Delta_{DL}$  with respect to the sequence of downlink data frames. Notice that  $\phi_0$  corresponds to the shift experienced by a reference uplink subperiod.

If the station holds a value of  $\Delta_{trig}(n)$  such that no event is observed within  $T$ ,  $\Delta_{trig}(n)$  is a converged value of the estimation. Indeed, not experiencing any event in  $T$  means not experiencing any event in any of the  $N$  uplink subperiods that conform  $T$ .

It is proved in the Appendix that in the general case where  $\Delta_{UL}$  is not multiple of  $\Delta_{DL}$ , the station has to generate  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor$  signaling triggers between uplink data triggers to avoid observing any event during  $T$ . Hence, a necessary condition for  $\Delta_{trig}(n)$  being a converged polling interval is  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{trig}(n)} \rfloor = \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor$ , or:

$$\frac{\Delta_{UL}}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor + 1} < \Delta_{trig}(n) < \frac{\Delta_{UL}}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor}$$

Considering that  $\Delta_{trig}(n) > \Delta_{DL}$ , we continue proving that a  $\Delta_{trig_{max}}$  exists such that if  $\Delta_{DL} < \Delta_{trig}(n) < \Delta_{trig_{max}}$  the station will never observe an event.

In order to find  $\Delta_{trig_{max}}$ , recall from Equation 1 that if no event is observed, the delay experienced by consecutive downlink frames can be expressed like  $d_0 + k\epsilon(n)$ , with  $\epsilon(n) = \Delta_{trig}(n) - \Delta_{DL}$ . Therefore, if  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor$  signaling triggers are sent between two consecutive data

triggers, the following condition must hold in order to avoid More Data events in any uplink subperiod:

$$\phi_i + \epsilon(n) \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor < \Delta_{DL} \iff \Delta_{trig}(n) < \Delta_{DL} + \frac{\Delta_{DL} - \phi_i}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor}$$

The previous expression implicitly assumes that the time between a signaling trigger and an uplink data trigger,  $\Delta_{res}$ , is below  $\Delta_{DL}$ . Otherwise, applying the same analysis, a tighter bound on  $\Delta_{trig}(n)$  with the same structure would be found. The most restrictive condition on  $\Delta_{trig}(n)$  is hence imposed by the uplink subperiod with the biggest  $\phi_i$ . Recalling Equation 4,  $\phi_{max}$  can be found to be  $\phi_{max} = \Delta_{DL} - (gcd(\Delta_{UL}, \Delta_{DL}) - \phi_{min})$ . Thus,  $\Delta_{trig_{max}}$  can be written as:

$$\Delta_{trig_{max}} = \min\left\{ \frac{\Delta_{UL}}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor}, \Delta_{DL} + \frac{\Delta_{DL} - \phi_{max}}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor} \right\}$$

Applying the same analysis when  $\Delta_{trig}(n) < \Delta_{DL}$ , it can be found that no event ever occurs if  $\Delta_{trig_{min}} < \Delta_{trig}(n) < \Delta_{DL}$ , where:

$$\Delta_{trig_{min}} = \max\left\{ \frac{\Delta_{UL}}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor + 1}, \Delta_{DL} - \frac{\phi_{min}}{\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor} \right\}$$

Note that in the general case,  $\Delta_{UL}$  not multiple of  $\Delta_{DL}$ , the amount of uplink data triggers that will download a data packet from the AP, i.e., those uplink data packets that are effectively reused as triggers, will be always the same and correspond to the number of up-

link subperiods where  $\lceil \frac{\Delta_{UL}}{\Delta_{DL}} \rceil$  downlink frames arrive at the AP. This number is found in the Appendix.

In the particular case where  $\Delta_{UL}$  is a multiple of  $\Delta_{DL}$ , the same analysis can be applied to find a new converged value of  $\Delta_{trig}$  located above  $\Delta_{DL}$ ,  $\Delta'_{trig_{max}}$ , such that only  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor - 1$  signaling triggers are generated by a station between uplink data triggers. This is again another argument to prefer the convergence of the algorithm from the More Data zone.

Note that regardless of the final converged value estimated in the station, the original requirements of the algorithm are always being fulfilled by the way the events have been defined. First, if a More Data event never occurs, the delay is effectively bounded to the downlink interarrival time. Second, if a No Data event never occurs, no useless signaling is generated.

The converged values of  $\Delta_{trig}(n)$  when no uplink data frames are reused as triggers can be obtained from the expressions of  $\Delta_{trig_{min}}$  and  $\Delta_{trig_{max}}$  by observing that:

$$\lim_{\Delta_{UL} \rightarrow \infty} \Delta_{trig_{min}} = \lim_{\Delta_{UL} \rightarrow \infty} \Delta_{trig_{max}} = \Delta_{DL}$$

Therefore, without uplink data triggers, the only converged value of  $\Delta_{trig}(n)$  that refrains the station from observing any event is  $\Delta_{trig}(n) = \Delta_{DL}$ .

Finally, to illustrate the analysis of convergence introduced in this section, Figures 5(b) and 5(d) depict the delay and jitter experienced by a downlink stream whose interarrival time is changed in a step-like fashion. The WLAN station is configured as in the experiment for Figure 5(a) and generates data triggers in the uplink every 90ms during the time intervals (20,40) and (80,100) seconds.

As predicted in the analysis, it can be observed by the constantly decreasing slope of the delay curves that, when no uplink data triggers are considered, the algorithm constantly approaches  $\Delta_{DL}$ . In the time intervals where the station sends uplink data triggers though, the predicted periodic situation in the channel is observed. Note that the different jitter levels observed in Figure 5(d) correspond to the different values predicted in the model introduced in Section 3.

## 5.2. Speed of Convergence

In the previous section it has been proved that the proposed algorithm always converges to a value that fulfills our design objectives. In this section we study the speed of the algorithm to converge to such a value.

Our goal is to find an upper bound on the number of updates needed in order to reduce an initial error  $\epsilon(0)$

to  $\epsilon(n) < \alpha \Delta_{DL}$ , where  $\alpha$  is a variable defined for the purpose of this analysis,  $0 < \alpha < 1$ .

Assuming that our algorithm operates in the More Data zone, as recommended, and recalling Equation 13, the error experienced in the next estimation update can be expressed and upper bounded in the following way:

$$\begin{aligned} \epsilon(n+1) &= \epsilon(n)(1-\gamma) + \gamma \frac{d_{02} - d_{01}}{n_{-fr\_rcvd}(n)} < \\ &< \epsilon(n)(1-\gamma\Psi(n)) \end{aligned}$$

Where  $\Psi(n) = \frac{n_{-fr\_rcvd}(n)-1}{n_{-fr\_rcvd}(n)}$  and  $0 \leq d_{02} < \epsilon(n)$ . Note that  $\Psi(n)$  is a monotonically growing function of  $n_{-fr\_rcvd}(n)$  and since the minimum number of frames that a station can retrieve from the AP between two consecutive More Data events is 2 then  $\frac{1}{2} < \Psi(n) < 1$ .

Based on that, a lower bound on the speed with which the error decreases can be obtained as:

$$\frac{\epsilon(n+k)}{\epsilon(n)} < \prod_{j=0}^{k-1} (1-\gamma\Psi(n+j)) < (1-\frac{1}{2}\gamma)^k$$

Where a constant adaptation step,  $\gamma$  has been assumed. If  $\gamma$  would be adaptive, its minimum value should be considered in order to obtain the previous bound.

Note that the adaptation step,  $\gamma$ , can be effectively used to control the speed of convergence. Nevertheless, an upper bound,  $\gamma_{CONS}$ , was found in the previous section in order to guarantee the convergence keeping the value of the estimation always in the same zone. A trade-off between speed of convergence and stability controlled by the adaptation step is an intrinsic characteristic of the steepest descent algorithm.

If the algorithm converges faster than  $(1-\frac{1}{2}\gamma)^k$ , an upper bound on the number of updates needed to reduce the error of the estimation from  $\epsilon(0)$  to  $\alpha \Delta_{DL}$ , can be found as:

$$K_{max} = -\log\left(\frac{\epsilon(0)}{\alpha \Delta_{DL}}\right) \frac{1}{\log(1-\frac{1}{2}\gamma)}$$

The former upper bound holds true regardless of whether the station generates uplink data triggers or not. The difference will be on the time needed to reach the critical number of estimation updates,  $K_{max}$ . The reason is that the time between consecutive events is different depending on whether uplink data triggers are considered or not. In any case though, the time between events increases when the error in the estimation,  $\epsilon(n)$ , decreases.

The benefits of updating the estimation in a way proportional to  $\frac{1}{\epsilon(n)}$  are indeed twofold. On the one

hand, the algorithm will react fast when there are sudden changes in the downlink interarrival time, because the error,  $\epsilon(n)$ , will increase. On the other hand, if the downlink interarrival time remains stable, the algorithm minimizes the required computational load by relaxing the frequency of updates as  $\epsilon(n)$  decreases.

Finally, to illustrate how the adaptation step can be used to tune the speed of convergence, in Figure 5(c) we show how a WLAN station using U-APSD estimates an interarrival time of 42ms using two different values of  $\gamma_{MD}$ ,  $\gamma_{MD} = 0.2$  and  $\gamma_{MD} = 0.9\gamma_{CONS}$ . A noticeable faster estimation is observed when  $\gamma_{MD} = 0.9\gamma_{CONS}$ , while keeping the estimation always in the More Data zone.

### 5.3. Relaxing the Assumptions of the Model

In order to derive the convergence properties of our adaptive algorithm we have assumed so far that the downlink and uplink data streams were essentially periodic. However, this strong assumption only holds in reality when congestion in the wired and wireless parts of the network is kept very low. In this section we take a simulative approach in order to analyze the degradation experienced by the algorithm when the previous assumptions are not valid.

We consider the same scenario used for Figure 5(a), but having a downlink stream with an interarrival time of 30ms and introducing behind the AP a virtual node which delays every incoming packet by a random value uniformly distributed between 0 and  $J$ ms, and drops every incoming packet with probability  $P$ . Thus, we want to study the robustness of the adaptive algorithm to increasing values of  $J$  and  $P$ .

Figure 6(a) depicts the dynamics of the algorithm in the presence of jitter (upper graph) and packet drops (lower graph). The first remarkable point is that as observed in both graphs the essential behavior predicted in Section 3, a saw-tooth pattern for delay, is maintained in the presence of jitter and packet drops. The actual dynamics though slightly differ in either case. In both cases, jitter or drops, if a trigger is sent and no frame is present in the AP the algorithm will experience a No Data event. On the one hand, if the missing frame was delayed due to jitter, the next time the station triggers the AP it will probably retrieve two frames, thus experiencing a More Data event. Since the algorithm is designed to only update the estimation between consecutive events of the same type (see Algorithm 1), No Data events and More Data events cancel out in the case of jitter resulting in a very stable estimation. On the other

hand, if the No Data event was due to a packet drop the algorithm will not experience a More Data event in the next trigger which results in sporadic increases of the estimation above the target polling interval.

To better quantify the effects of jitter and packet drops on the algorithm performance we increase  $J$  from  $0ms \rightarrow 50ms$  and  $P$  from  $0.1\% \rightarrow 20\%$ . This range for jitter and drops covers the margins recommended by ITU-T in [15]. Figure 6(b) illustrates for this experiment the worst case downlink delay and the amount of *null* triggers (triggers sent by the station which find no data in the AP). The results show how the algorithm can maintain a bounded delay for the downlink stream for a large range of jitter and drops,  $J < 40ms$  and  $P < 10\%$ . However, as the jitter and packet drops increase, degradation unavoidably appears in terms of null triggers which could result in an increased level of congestion in the network.

## 6. Performance Evaluation

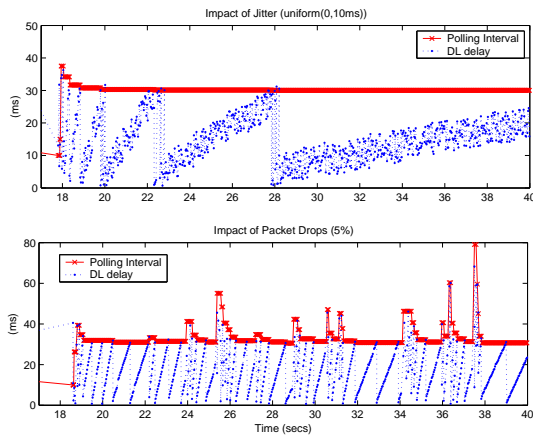
In this section we present a performance evaluation of our proposed adaptive algorithm. We divide this performance evaluation in two subsections. First, subsection 6.1 discusses the behavior of our adaptive algorithm when used with different applications in a typical Wi-Fi deployment. Second, subsection 6.2 analyzes the scalability and performance of our adaptive algorithm as congestion in the WLAN increases.

The analysis has been performed via simulations. We extended the 802.11 libraries provided by OPNET [16] to include the power saving extensions defined in 802.11 and 802.11e and our proposed adaptive trigger generation algorithm.

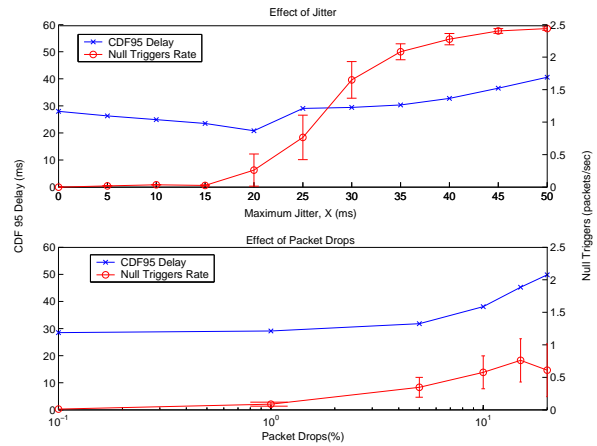
We consider infrastructure mode WLANs, with an AP beacon interval of 100ms. The physical layer chosen in our simulations is 802.11b with all stations transmitting at 11Mbps.

Next, we present a definition of a *traffic mix* that we consider representative of today's typical Wi-Fi deployments. Throughout this section we will consider stations using one/various of the applications that conform this traffic mix. Sometimes we will use the concept of a *cluster* of stations, which we define as a group of four stations where each station runs one of the applications defined in our traffic mix. Together with each application we indicate the EDCA access category used to transmit traffic of this application:

- AC\_VO: G.711 Voice codec with silence suppression. Data rate: 64kbps. Frame length: 20ms. Talk spurt exponential with mean 0.35s and silence spurt expo-



(a) Dynamics of the algorithm in presence of jitter or drops



(b) Performance with varying jitter and drops

Fig. 6. Behavior of the algorithm when relaxing the assumptions of the model

ponential with mean 0.65s.

- AC\_VI: Streaming video download. MPEG-4 real traces of the movie 'Star Trek: First Contact' obtained from [17]. Target rate: 64kbps. Frame generation interval: 40ms.
- AC\_BE: Web traffic. Page interarrival time exponentially distributed with mean 60s. Page size 10KB plus 1 to 5 images of a size uniformly distributed between 10KB and 100KB.
- AC\_BK: FTP download. File size of 1 MB. Interrequest time exponentially distributed with mean 60s.

For our experiments we consider that stations in the WLAN communicate through the AP with stations in a wired domain. The wired domain is modeled as an Ethernet segment behind the AP and a virtual node which represents the backbone of the wired network and introduces jitter and drops. Based on [15] we consider that the Voice and Video traffic traversing the backbone network suffer a maximum delay variation of 5ms and a 0.1% packet drop probability. Additionally Web and FTP applications communicate with a server using TCP New Reno and experiencing an RTT of 20ms.

As previously mentioned the channel access function considered in our simulations is EDCA. We assume a fixed configuration of the EDCA QoS parameters based on the 802.11e standard recommendation [2]. The parameters used are detailed in Table 1.

Throughout this section our adaptive algorithm will be configured in the following way. The initial polling interval,  $\Delta_{trig}(0)$ , is set to 10ms, *consecutive\_long\_burst* is 2,  $\beta$  is set to 1.5 and the algorithm switches off after sending three consecutive triggers without finding data in the AP.

In addition we will consider that all the trigger gen-

EDCA	AIFS	CWmin	CWmax	TXOP length
AC_VO	2	31	63	3.264 ms
AC_VI	2	63	127	6.016 ms
AC_BE	3	127	1023	0
AC_BK	7	127	1023	0

Table 1  
EDCA configuration for the different ACs

eration algorithms studied in this section use U-APSD, configured with *Max\_SP\_Length* set to *All*, instead of Legacy Power Save Mode, due to its better support for QoS.

### 6.1. Dynamics of the algorithm with realistic applications

In order to study how our adaptive algorithm behaves with different applications we consider the following scenario. A set of five clusters, which results in a total of 20 stations in the network, generate traffic according to our defined traffic mix. This traffic is considered to be background load in the WLAN. In addition we consider a tagged station configured to run the specific application that we want to study. In this section we will consider this station running the following applications: i) VoIP with silence suppression, ii) Video Streaming, iii) TCP download and iv) a mix of different applications.

#### 6.1.1. VoIP with VAD

A G.711 voice codec with voice activity detection (VAD) has been considered in our experiments.

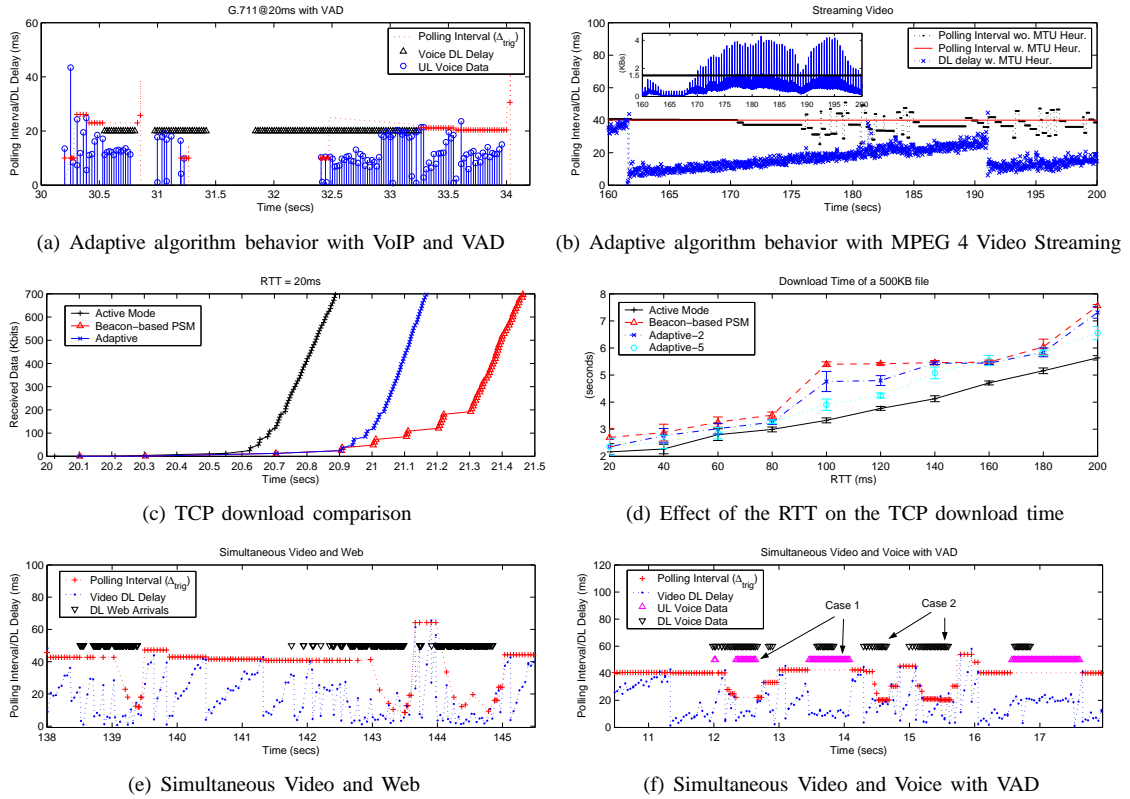


Fig. 7. Instantaneous Adaptive algorithm behaviour

The instantaneous behaviour of the adaptive algorithm is illustrated in Figure 7(a). The dynamics of the algorithm can be summarized in the following terms. First, when there is a burst of voice packets in the downlink, e.g. around 30.5 seconds, the station applies the adaptive algorithm to adjust its polling interval that converges to 20ms. Second, after sending 3 consecutive triggers without response, the algorithm infers that there is a silence gap and cancels the generation of triggers, as observed at 34 seconds. Switching off the algorithm though, can turn into an increased delay for the first voice samples when the activity restarts, as seen at 30 seconds in the trace. Finally, when the application generates data frames in the uplink, the signaling triggers are deferred by means of the re-scheduling mechanism.

### 6.1.2. Video Streaming

The behaviour of the adaptive algorithm in case of Video Streaming is depicted in Figure 7(b). The MPEG-4 codec generates frames at a nominal interarrival time of 40ms. These frames though, may need to be fragmented (MTU of Ethernet) and as result the AP can receive bursts of frames every 40ms.

Figure 7(b) illustrates the instantaneous polling inter-

val used in the station with and without the MTU heuristic defined in Section 4, the MAC delay experienced by the video frames retrieved from the AP and in a sub-graph, the instantaneous evolution of the frame sizes generated by the MPEG-4 codec. It can be seen that before 170 seconds, when the size of the video frames is most of the time below the fragmentation threshold, the algorithm maintains a very stable value of the estimation regardless of whether the MTU heuristic is used or not. From 170 seconds on though, the codec generates a burst of frames above the fragmentation threshold, making the frame interarrival time at the AP deviate from 40ms. In this case, if the MTU heuristic is not applied the estimated polling interval shakes around 40ms. If the MTU heuristic is applied though, the estimation remains perfectly stable at 40ms.

### 6.1.3. TCP application

In order to illustrate the effect of the adaptive algorithm on applications running over TCP, we consider in this section that our station under study performs a TCP download of a 500KB file.

In [7] an interesting interaction between TCP and

the Beacon-based<sup>11</sup> power saving protocols used in WLAN was introduced. This interaction is as follows. The first TCP packets sent by the TCP connection in slow start are buffered in the AP and only retrieved by the station after a Beacon frame. While retrieving these frames, the station generates TCP ACKs in the Uplink and immediately goes back to the sleep state. After an RTT the next window of TCP packets sent by the server will reach the AP and be buffered there until the next Beacon frame. This process will repeat until the time required by the station to retrieve the whole window of packets stored in the AP after a Beacon gets above the RTT between the AP and the server; from that moment on the station will remain awake until the TCP connection closes. Notice though that the station may never enter this 'active-mode' like behavior if its TCP advertised window is smaller than the required critical value.

This effect can be altered in a beneficial way when our adaptive algorithm is used. Figure 7(c) depicts the instantaneous traces of a TCP download between a WLAN station and a server experiencing a 20ms RTT with the AP in the case of the adaptive algorithm, a traditional Beacon-based power save mode and a station in active mode. The previous interaction is clearly observed in the case of the Beacon-based power save mode. However, in the case of the adaptive algorithm, the length of the initial RTTs can be highly reduced because the additional triggers generated by the algorithm after a TCP burst can capture the new burst of TCP packets arriving from the server before the next Beacon frame. Hence, the actual improvements obtained in the case of the adaptive algorithm will depend on the value of the RTT experienced by the connection, the number of additional triggers generated by the algorithm before switching off and the value of  $\beta$  in Algorithm 1. We do not study in this work how to optimally tune these two parameters.

To better illustrate the effects of the adaptive algorithm on the lifetime of TCP connections, in Figure 7(d) the download time of a 500KB file in our reference scenario is depicted while increasing the value of the RTT between the AP and the TCP server. As in [7] the TCP advertised window in the WLAN station is configured to be 240 Kbits. Four different cases are considered, a station in active mode, a Beacon-based power save algorithm, the adaptive algorithm generating two triggers before switching off (*Adaptive-2*) and the adap-

tive algorithm generating 5 triggers before switching off (*Adaptive-5*). The results corroborate that the adaptive algorithm can reduce the download time with respect to the traditional Beacon-based algorithm and that this reduction increases with the number of triggers generated before switching off.

#### 6.1.4. Simultaneous Applications

Until now we have always assumed that our WLAN station under study was running only one application at a time. This is often not the case in realistic scenarios where a terminal generates traffic from more than one application at once. In this section we study how the adaptive algorithm reacts to traffic generated by multiple applications at the same time.

Figure 7(e) illustrates the behavior of the algorithm in a terminal running a Video Streaming application, through AC\_VI, while performing Web browsing, through AC\_BE. The Figure depicts the polling interval estimation kept by the algorithm, the delay experienced by the Video frames and the presence of Web data packets buffered in the AP (black markers). The dynamics of the algorithm are as follows. When no Web traffic is present in the AP the algorithm converges to the interarrival time of the Video codec, i.e. 40ms. However, when a burst of Web traffic arrives at the AP, the algorithm suddenly sees an increased amount of data and reduces the estimation in order to quickly retrieve the extra data. When the burst of Web data packets is retrieved, the polling intervals returns to 40ms.

Figure 7(f) illustrates the case where a terminal simultaneously generates traffic from a Video Streaming application, through AC\_VI, and a Voice codec with silence suppression, through AC\_VO. The behavior of the adaptive algorithm can be understood in the following terms. When the Voice codec is in silence in both uplink and downlink directions, as observed in the trace before 12 seconds, the polling interval converges to the video codec interarrival time, i.e. 40ms. When Voice packets are generated in the uplink, as observed in the trace in the areas labeled as *Case 1*, the rescheduling mechanism avoids the generation of signaling triggers, and the uplink Voice packets download both Voice and Video packets buffered in the AP.<sup>12</sup> Finally, when downlink Voice packets are buffered in the AP but no Voice data packets are generated in uplink, as observed in the trace in the areas labeled as *Case 2*, the polling interval estimation converges to the most stringent delay requirement, in this case Voice with a 20ms interarrival time.

<sup>11</sup>We consider a Beacon-based power saving protocol as a protocol where the station only triggers the AP for data after receiving a Beacon frame with the TIM bit set to 1.

<sup>12</sup>Notice that we configure all ACs as trigger and delivery enabled in U-APSD.

As shown in the previous experiments the algorithm performs as desired when simultaneous applications are considered. Signaling load is generated according to the application showing the most stringent delay requirements, and the algorithm is fast enough to quickly adapt to bursty applications, e.g. Web and Voice with silence suppression.

## 6.2. Performance under congestion

In this section we study how the adaptive algorithm behaves when the congestion in the WLAN increases. For this purpose we consider a set of  $N$  clusters in the WLAN, one cluster being composed of four stations and each station running a different application from our defined traffic mix, and we increase  $N$  until congestion appears.

In addition we compare the adaptive algorithm with two other algorithms existent in the state of the art. Unlike our adaptive proposal though, these algorithms require knowledge about the application characteristics. We will assume that these algorithms are perfectly configured according to the application characteristics in order to set an upper bound on the expected performance of the adaptive algorithm. Next, we describe the mentioned algorithms that will be hereafter referred to as the *Ideal Static* and the *Ideal Static+Signaling Reduction(SigRed)* algorithms.

The *Ideal Static* algorithm configures the operation of U-APSD depending on the traffic generated by the stations. When the WLAN terminal runs a Voice or Video application, U-APSD is statically configured to periodically generate trigger frames with an interval adapted to the nominal application interarrival time<sup>13</sup>. However if the terminals use applications running over AC\_BE or AC\_BK, like Web or FTP in our experiment, the *Ideal Static* algorithm configures U-APSD to behave like a Beacon-based power save mode.

The *Ideal Static+SigRed* algorithm is an improvement over the previous algorithm inspired on the BSD protocol proposed in [7] that reduces the amount of introduced signaling load when the application is in a silence period. The idea is the following. When the terminal runs a Voice or Video application a nominal polling interval is defined that matches the nominal application characteristics. However, if the terminal sends a signaling frame and observes that no data was buffered in the AP the algorithm infers that the application is in a silence period and increases the polling interval like

$T_{int}(n+1) = T_{int}(n)(1+p)$ , where  $p > 0$ . Whenever traffic is again detected in the downlink direction the trigger interval is returned to its nominal value. The parameter  $p$  can be used to trade-off the amount of signaling load introduced during silence periods and the extra delay experienced by the downlink stream if wrong silence periods are inferred. For our evaluation we consider  $p = 0.5$  and in order to keep a bounded delay  $T_{int_{MAX}} = 100ms$ . Regarding applications running over AC\_BE or AC\_BK we consider that the *Ideal Static+SigRed* algorithm behaves in the same way than the *Ideal Static* one.

Next, we present the performance of the *Adaptive*, *Ideal Static* and *Ideal Static+SigRed* algorithms in terms of QoS, power consumption and introduced signaling load. In addition we finalize this subsection providing a deeper insight on how congestion in the WLAN affects the dynamics of the adaptive algorithm.

The simulations presented in this section have a length of 300 seconds with a warm-up phase of 30 seconds. The graphs showing average values are plotted with the corresponding 95% confidence intervals<sup>14</sup>. The graphs regarding delay show the 99% percentile of the delay computed considering together the samples obtained from all simulation runs.

### 6.2.1. QoS Performance

In this section we analyze the Throughput and Delay metrics. Since the distributed power saving mechanisms degrade the performance mainly in the downlink direction (AP  $\rightarrow$  STA), we focus on the downlink results which is the performance bottleneck of the system.

Figure 8(a) shows for each AC the average MAC throughput experienced in the downlink direction. The results show that the *Adaptive* (solid line) and the *Ideal Static+SigRed* (dash-dot line) algorithms outperform the *Ideal Static* algorithm (dotted line) in all ACs except AC\_VO, which still does not reach saturation due to the small size of the VoIP packets compared to the AC\_VO TXOP length and the amount of buffering available in the AP. The main reason for the improved performance observed in the case of the *Adaptive* and *Ideal Static+SigRed* algorithms is their ability to reduce signaling load when the application is in a silence period. Notice that in the case of Voice, detecting silence periods in the downlink results in a reduction of the amount of high priority load (AC\_VO) introduced in the channel, which benefits all other ACs. In addition, a slightly better performance under congestion is observed in the

<sup>13</sup>If Uplink data triggers are sent signaling triggers are recheduled.

<sup>14</sup>Note that the confidence intervals are present in all graphs although they might not be visible in some cases due to its small size

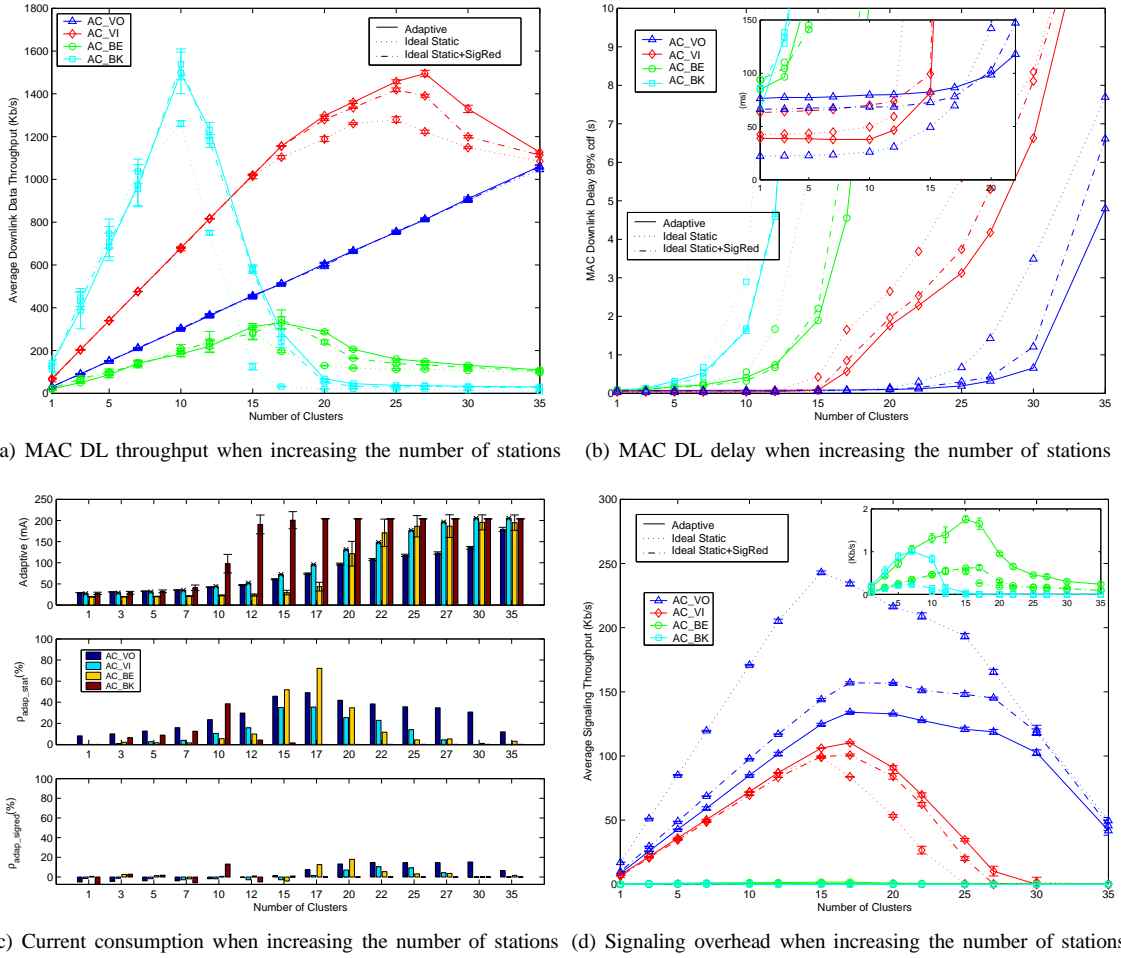


Fig. 8. Overall Network Performance

case of the Adaptive algorithm with respect to the Ideal Static+SigRed algorithm. The reason is the particular configuration of both algorithms which results in the Adaptive algorithm introducing slightly less signaling load during Voice silence periods.

In Figure 8(b) the worst case (99% percentile) delay is depicted for the different algorithms under study and each AC. The delay results closely resemble the throughput ones, with the Adaptive and Ideal Static+SigRed algorithms clearly outperforming the Ideal Static one, the only difference being that now differences are observed also for AC\_VO. When the network is not yet fully congested (<15 stations), the Adaptive algorithm bounds the downlink delay to the interarrival time in the case of AC\_VI but not in the case of AC\_VO, where a higher value is observed. This is the price to pay in order to save signaling load because when the trigger generation stops the next voice

samples can only be recovered with the Beacon frame, experiencing hence a higher delay. The same effect is observed in the case of the Ideal Static+SigRed algorithm for AC\_VO, due to the silence periods of the codec, and AC\_VI, because of spurious increments in the trigger interval due to jitter in the Video stream. Notice that considering a higher value for  $p$  in the Ideal Static+SigRed algorithm would result in higher worst case delays when the network is not congested.

Finally, we conclude that in terms of QoS our proposed Adaptive algorithm can provide a performance equivalent to that of optimally configured dynamic approaches existent in the state of the art (Ideal Static+SigRed) and a much superior performance than static approaches (Ideal Static), with the advantage of not requiring a priori knowledge about the applications characteristics.

### 6.2.2. Power Saving Efficiency

The main objective of any power saving mechanism is to reduce the power consumption of a station. In this section, we analyze the power consumed by the stations in order to study whether the benefits exhibited by the adaptive algorithm have a price in terms of power consumption.

The power consumption model used for the evaluation has been derived based on current WLAN cards/chipsets available in the market and consists of four states: Sleep, Listen, Reception and Transmission. In order to obtain the power consumed by the stations, we first compute the percentage of time spent during an active session in each state by the stations per AC. Then, we translate this generic metric into *mA* weighting the current consumed by a station in each state. This information has been obtained from the product datasheet of a common PCMCIA WLAN card [18]. Thus, the metric represents the average instantaneous current consumption (in *mA*) experienced by the stations in each AC. The current consumption values used are shown in Table 2<sup>15</sup>

Cisco Aironet <sup>TM</sup>	Sleep	Listen	Rx	Tx
Current (mA)	15	203	327	539

Table 2  
Current consumption levels of a popular PCMCIA card

The upper part of Figure 8(c) illustrates the average current consumption levels for each AC for the adaptive algorithm. It can be observed how the average current consumption increases with the congestion in the network. These results though, confirm that when the network is not congested, a noticeable current consumption reduction can be achieved during an active session if a distributed power saving mechanism is used. Looking for instance at the case of VoIP, less than 50mA are consumed on average when the network is not congested, in front of almost 200mA when the network is highly congested and the stations spend almost all the time awake.

Regarding the current consumption differences between the different algorithms, the middle and lower part of Figure 8(c) illustrate the relative current consumption reduction of the adaptive algorithm with respect to the other ones ( $\rho_{Adaptive\_AlgX} = \frac{AlgX(mA) - Adaptive(mA)}{AlgX(mA)}$ ). We can see that with respect to the Ideal Static algorithm (middle figure) a significant power consumption reduction is achieved by the

<sup>15</sup>For the sleep mode we used the value of a previous model of a Cisco PCMCIA card (Cisco Aironet 350) since no information was available for the current one

adaptive algorithm in all ACs. With respect to the Ideal Static+SigRed algorithm (lower figure) though, a slight power consumption reduction is observed only when the network is congested. The main reason for the reduced current consumption is again on the smaller amount of signaling load introduced by the adaptive algorithm, which results in less congestion in the network.

### 6.2.3. Signaling Overhead

The main factor determining the scalability of the different algorithms under study is the amount of signaling load introduced. The challenge is to minimize the amount of signaling load while at the same time provide the required QoS to the applications. In this section we validate the previous statements related to the amount of signaling introduced by each algorithm by showing in Figure 8(d) the total signaling both in uplink and downlink introduced by each AC.

Regardless of the algorithm considered, the first notable effect is a reduction in the amount of signaling introduced in all ACs when the congestion in the channel increases. This effect is intrinsic to the U-APSD protocol and is due to the following two facts: i) the AP delivers all buffered data for a station when receiving a trigger frame, and ii) the station does not generate new trigger frames if there is an ongoing service period. Thus, when the network gets congested, the time that the AP needs to deliver a frame in a service period becomes longer, which in turn increases the chances that a new frame addressed to the station arrives at the AP within an ongoing service period. This translates into an increase of the stations' listen time and in a reduction of the number of triggers generated.

Focusing on AC\_VO, we can see the effective reduction in signaling load achieved by the Adaptive and Ideal Static+SigRed algorithms. Indeed, stopping the trigger generation when there is no activity in the downlink, saves both the uplink signaling generated by the station and the downlink signaling generated by the AP in order to end the service period. In addition, a slightly higher amount of signaling is introduced by the Ideal Static+SigRed algorithm which could be reduced, at the price of accepting higher delays, by increasing the parameter  $p$ .

Regarding the AC\_VI application, when the network is not saturated a very similar amount of signaling is introduced by all the algorithms, which saturates before in the case of the Ideal Static algorithm due the higher level of congestion in the network in this case.

Finally, looking at the AC\_BE and AC\_BK results, we can see that a very small amount of signaling is

introduced as compared to the other ACs. The reason is that TCP traffic arrives in bursts that can be retrieved by the stations with a single trigger thanks to the configured *Max\_SP\_Length* value. The subgraph in the graph shows that a slightly higher amount of signaling is introduced in the case of the adaptive algorithm. The reason is that once the burst of packets is downloaded, the adaptive algorithm generates some additional triggers prior to switching off which as previously seen in subsection 6.1 can help to improve the performance of TCP.

#### 6.2.4. Understanding the effect of congestion

We finish this section providing an analysis of how congestion in the WLAN affects the dynamics of the adaptive algorithm.

Figure 9 depicts the CDF of the trigger interval estimation for the stations running Video considering three different levels of congestion, i.e. 5, 15 and 25 clusters (5x4, 15x4 and 25x4 competing stations). As congestion increases, e.g. 15 clusters, the estimation becomes more unstable but is still kept around the desired nominal interarrival time (40ms in the case of Video). The reason is that congestion in the WLAN translates into packet drops or extra delays which distort the interarrival time of the downlink stream perceived by the algorithm.

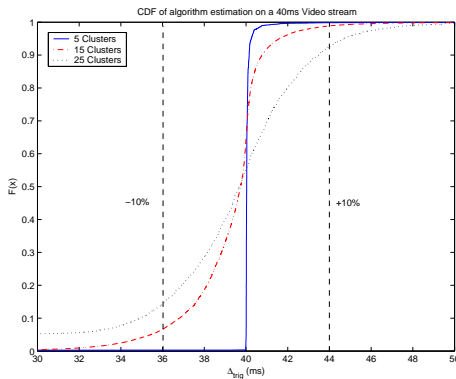


Fig. 9. Effect of congestion on the Adaptive Algorithm

These results show that the estimation kept by the algorithm becomes less accurate as congestion in the WLAN increases but it is still kept in 80% of the cases within a  $\pm 10\%$  range even for a significantly high congestion level as 25x4 competing stations. Note from Figure 8(a) that Video traffic has already started to experience congestion at 25x4 competing stations. Hence we conclude that the proposed algorithm is able to cope with moderate to high levels of congestion in the WLAN.

## 7. Conclusions

Designers of mobile devices incorporating the Wireless LAN technology face new challenges with respect to the QoS and power saving requirements to be met to satisfy users' expectations. IEEE 802.11 and 802.11e provide some mechanisms to address these challenges but the algorithms required to achieve the desired performance are not defined to allow for vendor differentiation. The focus of our work in this paper has been the study of the dependencies between the configurable parameters of the distributed power saving mechanisms and its impact over the resulting QoS performance. Based on this, we designed a power saving algorithm that satisfies users' expectations both in QoS and power saving terms and at the same time makes an efficient use of the network resources.

Our contributions are as follows. First, we have derived an analytical model of the dependency of the downlink delay and jitter with the configuration of the power saving mechanism in use. Second, we have designed an adaptive algorithm, based on the steepest descent method, that determines the appropriate wake up rate for requesting frames from the AP according to the instantaneous application offered load, based only on information available at the MAC layer. Third, we have analyzed the convergence properties of our adaptive algorithm, providing guidelines to set its configurable parameters and a worst case bound on the number of updates required to converge. Finally, we have compared, by means of simulations, the performance of our approach with approaches existent in the state of the art which require knowledge of the applications' characteristics.

The main conclusions that can be drawn from our results are i) the downlink delay is bounded by the trigger generation interval and increases or decreases linearly according to the difference between the station's polling interval and the downlink interarrival time following a saw-tooth pattern, ii) the jitter keeps constant with the exception of the case when a trigger frame sent by the station finds either no frames or more than one in the AP's power save buffer which results in a sharp increase in the experienced jitter, iii) an adaptive algorithm for a station can be designed, based only on information available at the MAC layer, such that the delay of the downlink frames is bounded according to a desired multiple of the interarrival time at the AP and iv) an adaptive solution that dynamically adapts to the instantaneous traffic characteristics of the applications matches or outperforms existent state of the art approaches but

with the significant advantage of not requiring any a priori knowledge of the applications' characteristics.

## 8. APPENDIX

Assuming the definitions introduced in Section 5,  $K_i$  frames arrive at the AP during uplink subperiod  $i$ , where  $K_i = \lfloor \frac{\Delta_{UL} + \phi_i}{\Delta_{DL}} \rfloor$ . Hence:

$$K_i = \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor + \lfloor \frac{\Delta_{UL} \bmod \Delta_{DL} + \phi_i}{\Delta_{DL}} \rfloor$$

Thus:

$$K_i = \begin{cases} \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor & \text{if } \phi_i < \Delta_{DL} - \Delta_{UL} \bmod \Delta_{DL} \\ \lceil \frac{\Delta_{UL}}{\Delta_{DL}} \rceil & \text{if } \phi_i \geq \Delta_{DL} - \Delta_{UL} \bmod \Delta_{DL} \end{cases}$$

Assuming the general case where  $\Delta_{UL}$  is not multiple of  $\Delta_{DL}$  ( $\Delta_{UL} \neq n\Delta_{DL}$ ,  $n \in \mathbb{Z}$ ), and recalling  $0 \leq \phi_i < \Delta_{DL}$ , the channel situation periodic  $T = \text{lcm}(\Delta_{UL}, \Delta_{DL}) = M\Delta_{DL} = N\Delta_{UL}$ , and  $T$  composed by  $N$  consecutive uplink subperiods, during  $l > 0$  and  $j > 0$  uplink subperiods  $\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor$  and  $\lceil \frac{\Delta_{UL}}{\Delta_{DL}} \rceil$  downlink frames are received at the AP respectively, with:

$$\begin{cases} l \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor + j \lceil \frac{\Delta_{UL}}{\Delta_{DL}} \rceil = M \\ l + j = N \end{cases}$$

Therefore:

$$\begin{aligned} l &= N(\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor + 1) - M \\ j &= M - N \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor \end{aligned}$$

Thus, within  $T$  there are both uplink subperiods receiving  $\lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor$  and  $\lceil \frac{\Delta_{UL}}{\Delta_{DL}} \rceil$  data frames in the downlink. Therefore the station must generate  $\delta = \lfloor \frac{\Delta_{UL}}{\Delta_{DL}} \rfloor$  signaling triggers between two consecutive data triggers in order not to generate any event in any uplink subperiod. Hence, the reuse of uplink data triggers in this case is  $\frac{j}{N}$ .

In the case that  $\Delta_{UL} = n\Delta_{DL}$ ,  $n \in \mathbb{Z}$ , the same number of data frames arrive at the AP in all uplink subperiods,  $\frac{\Delta_{UL}}{\Delta_{DL}}$ , and a converged value of  $\Delta_{trig}$  can be obtained generating  $\delta = \frac{\Delta_{UL}}{\Delta_{DL}}$  or  $\delta = \frac{\Delta_{UL}}{\Delta_{DL}} - 1$  signaling triggers between uplink data triggers. Thus, the reuse of uplink data triggers in this case will be 0 or 1 respectively.

## References

- [1] IEEE 802.11 WG, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11, June 1999.
- [2] IEEE 802.11 WG, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*, IEEE Standard 802.11e, November 2005.
- [3] <http://www.wi-fi.org>.
- [4] S.Mangold and S.Choi and G.R.Hiertz and O.Klein and B.Walke, *Analysis of IEEE 802.11e for QoS Support in Wireless LANs*, IEEE Wireless Communications Magazine, December 2003.
- [5] X.Pérez-Costa, D.Camps-Mur and T.Sasihara. *Analysis of the Integration of IEEE 802.11e Capabilities in Battery Limited Mobile Devices*. IEEE Wireless Communications Magazine (WirComMag), special issue on Internetworking Wireless LAN and Cellular Networks, Volume 12, Issue 6, December 2005.
- [6] X.Pérez-Costa, D.Camps-Mur, A. Vidal. *SU-APSD: Static IEEE 802.11e Unscheduled Automatic Power Save Delivery*. Computer Networks Volume 51, Issue 9, 20 June 2007, Pages 2326-2344.
- [7] R.Krashinsky and H.Balakrishnan, *Minimizing energy for wireless web access with bounded slowdown*, Wireless Networks archive Volume 11, Issue 1-2 (January 2005).
- [8] D.Qiao and K.G.Shin, *Smart Power Saving Mode for IEEE 802.11 Wireless LANs*, Proceedings of IEEE INFOCOM, March 2005.
- [9] G. Anastasi, M. Conti, E. Gregori, A. Passarella *802.11 powersaving mode for mobile computing in WiFi hotspots: limitations, enhancements and open issues*, Wireless Networks Volume 14, Issue 6 (December 2008).
- [10] Y.Chen and S.Emeott, *Investigation into Packet Delivery Options for WLAN Access Points Implementing Unscheduled Power Save Delivery*, Proceedings of IEEE Globecom, November 2005.
- [11] X.Pérez-Costa and D.Camps-Mur, *APSM: Bounding the Downlink Delay for 802.11 Power Save Mode*, In Proceedings of IEEE International Conference on Communications (ICC), May 2005.
- [12] X.Pérez-Costa and D.Camps-Mur, *AU-APSD: Adaptive IEEE 802.11e Unscheduled Automatic Power Save Delivery*, In Proceedings of IEEE International Conference on Communications (ICC), June 2006.
- [13] UPnP Forum, *UPnP QoS Architecture V2.0*, October 2006.
- [14] Xavier Pérez Costa, Daniel Camps Mur. *A protocol Enhancement for IEEE 802.11 Distributed Power Saving Mechanisms. No Data Acknowledgement*. IST summit 2007, Budapest, July 2007.
- [15] ITU-T Recommendation Y.1541
- [16] OPNET Technologies, <http://www.opnet.com>.
- [17] F.H.P. Fitzek and M.Reisslein, *MPEG-4 and H.263 Video Traces for Network Performance Evaluation*, IEEE Network, Vol. 15, No. 6, pages 40-54, November/December 2001.
- [18] Cisco Aironet 802.11a/b/g Wireless CardBus Adapter, <http://www.cisco.com/>.