A Power-Aware Multimedia Streaming Protocol for Mobile Users

G. Anastasi and A. Passarella
University of Pisa
Dept. of Information Engineering
Via Diotisalvi, 2 – 56126 Pisa, Italy
{g.anastasi, a.passarella}@iet.unipi.it

M. Conti, E. Gregori, and L. Pelusi
IIT Institute - CNR Research Area
Via G. Moruzzi, 1 – 56124 Pisa, Italy
{marco.conti, enrico.gregori, luciana.pelusi}@iit.cnr.it

Abstract

Smart environments are becoming popular and even more users are approaching their services through portable devices like PDAs, laptops, and mobile phones. These devices are generally battery-fed, thus, energy efficiency is surely a critical factor for the deployment of pervasive services.

In this paper we focus on the diffusion of multimedia streaming services in smart environments. Specifically, we investigate scenarios where mobile users who have a Wi-Fi access to the Internet receive audio files from remote streaming servers. We address energy saving by including periodic transmission interruptions in the schedule of audio frames at the server, so that the network interface card at the receiver can be set to a low-power consuming state.

Experimentation on a software prototype shows that our solution makes possible an energy saving ranging from 76% to 91% (depending on the data rate and the background traffic) of the total consumption due to the network interface. It also preserves a good user level QoS.

1. Introduction

The growing interest that people are showing in pervasive environments is encouraging migration of popular Internet services to mobile users. However, since deep differences exist between mobile devices and desktop computers, much effort must be spent to adapt the classical Internet applications to the world of wireless. Multimedia streaming services are envisioned to become very popular in mobile environments thus, in this paper, we focus on the development of streaming solutions for mobile users with emphasis on energy efficiency that we consider a key factor for the deployment of pervasive environments.

Streaming is a distributed application which provides on-demand transmission of audio/video files from a server to a client over the Internet while allowing playback of the arriving file chunks at the client. Playback starts a few seconds (*initial delay*) after the client receives the first chunk of the requested file and goes on while later parts of the file still arrive. Before playback, arriving chunks are temporarily stored at the client in a *buffer*.

The streaming service must guarantee a good playback quality. This fixes strict *time-constraints* to the transmission process because a timely delivery of packets is needed in order to ensure their availability at the client for playback.

Since constant-quality encoded audio/video files produce *variable-bit-rate (VBR) streams,* the resource allocation through the Internet during a streaming session is a hard task, and one of the main goals for a streaming server must be the design of a *transmission schedule* that tunes the outgoing traffic to the desired network and client resource usage, while obviously respecting the playback time constraints. A good scheduling algorithm is characterised by efficient use of the available network resources without overflowing or underflowing the client buffer. Moreover, it should minimize the delay before playback starts because users do not generally tolerate high initial delays (greater than 5-10s) especially when waiting for short sequences. Many smoothing algorithms have been proposed in the literature [1], however no one can be elected as the best because the performance metrics to be considered are conflicting and cannot be optimised all together. These metrics are: (i) peak bandwidth requirement, (ii) variability of transmission rate, (iii) number of rate changes, (iv) client buffer utilization, (v) additional computational burden to the server. Moreover, the same algorithm generally performs differently when applied to different media streams with different burstiness. As a result, the most suitable smoothing algorithm changes every time depending on the particular resource allocation model adopted at the server, at the client and at the network routers, as well as on the particular performance goals of the system components.

In this paper we focus on streaming services in wireless scenarios where servers are intended to be fixed hosts in the Internet whereas clients are on mobile devices (e.g., PDAs, laptops or mobile phones with wireless network interfaces) and access the Internet through an intermediate base station. We assume the presence of a *proxy* between the client and the server since proxies are commonly used to boost wireless network performances. In this environment we concentrate on energy efficiency because, since mobile devices are battery-fed, this is a key factor in the development of applications for mobile scenarios [12]. We will provide a solution that allows the energy consumption due to the Wireless Network Interface Card (WNIC) to go down to 9-24% of the energy consumption achieved by current systems.

Another challenging goal, in this area, is certainly minimizing the client buffer size, as memory costs for mobile devices are still high. We will demonstrate that our solution achieves good energy savings even with small buffers, i.e., down to 50 Kbytes.

Hereafter we will refer to WLAN 802.11b environments therefore we will consider that access to the Internet is provided to mobile hosts by the means of Access Points. However, our solution is flexible enough to adapt to any type of infrastructure-based wireless LAN.

The remainder of the paper is organized as follows. In Section 2 we will investigate some related work on power management solutions with emphasis on those tailored on real-time streaming applications. Sections 3, 4 and 5 will be devoted to describing our power-saving solution. In Section 6 we will provide experimental results assessing the system effectiveness, and we will draw our conclusion in Section 7.

2. Related Work

Energy consumption issues in wireless networks have been addressed in many papers. [2] and [3] found that one of the most energy-consuming components in mobile devices is the WNIC since networking activities are responsible for up to the 50% of the total energy consumption (10% in laptops, up to 50% in hand-held devices). In order to reduce the activity of the WNIC, [4] targeted the reduction of transfer delays over the Internet and first proposed an alternative to the classical TCP-IP architecture for the management of communications between mobile hosts and fixed hosts: the *indirect-TCP*. It *splits the connection* between the mobile host and the fixed host in two parts, one between the mobile host and the Access Point, the other between the Access Point and the fixed host. At the Access Point a running daemon is in charge of relaying data between the two connections. The Indirect-TCP correctly handles wireless losses and avoids the throughput reduction due to the combined action of such losses and the TCP congestion control so the total transfer delay significantly reduces. Further improvements are possible by adding proxy facilities at the Access Point. The TCP proxy shields the wireless link from packet losses in the Internet leading to better link utilization and throughput [5].

Nevertheless, the reduction of the total transfer delay can only produce a slight energy saving in contrast to the significant gains that can be achieved by *switching the WNIC off* when not actually working and by exchanging messages at the *maximum rate* allowed by the network channel when working. As stated in [2], [8], during the total transfer delay the WNIC consumes almost the same both in receiving, transmitting and idle state. As a result, the burstiness of the traffic, which is responsible for even long idle periods, naturally leads to energy wastage. [2], [3], [6], and [7] handle power saving in an Indirect-TCP architecture by switching off the WNIC during inactivity time periods. [6] proposes an application dependent approach tailored on web applications and exploits the knowledge of statistical models for web traffic to *predict packet arrival times* and switch the WNIC off between consecutive arrivals. [7] proposes an application independent approach and suits much more traffic types.

Solutions mentioned so far are not good for streaming applications because of the time-constraints imposed to the transmission process. Existing power management solutions for real-time applications are targeted to both *switch the WNIC to a sleep (doze) mode* during inactivity time periods and *reduce the total transfer delay* of the streaming session.

Transcoding techniques have been introduced to shorten the size of audio/video streams so as to reduce the transfer delay. However, while leading to little energy savings (see above), these techniques may determine a significant reduction in the *stream quality*.

A power management system that exploits the sleep mode of WNICs is described in the IEEE 802.11 standard: it is the PSM (Power Saving Mode) protocol [13]. However, as stated in [10], it only performs well when the arriving traffic is regular, whereas it is not suitable for popular multimedia streams; therefore, hereafter we will consider the PSM not being active.

Application layer techniques have been proposed in order to make predictions on the arriving traffic so as to switch the WNIC on when a packet is expected to arrive and to the sleep mode when an idle period is expected to start. [9], [11] propose solutions with client side predictions of the inter-arrival periods based on the history of the experimented inter-arrival periods in the past. Wrong predictions result in packet loss and in the worsening of the quality perceived during playback. Moreover, low packet loss probability and high energy saving are conflicting targets and a tradeoff must be met. Techniques based on client predictions can lead to energy saving ranging between 50% and 80% [9]. As for the PSM case, however, the best performances are achieved when the incoming traffic is regular. Traffic shaping can help predictions at the client become effective. However, an intermediate proxy must be introduced because transmission over the Internet change the traffic profile by adding jitter to the packets, thus performing traffic shaping at the server is useless. The power saving solutions proposed in [10] and [11] perform *traffic shaping at the proxy* and make use of client side predictions on idle periods in order to set the WNIC to the sleep mode. [11] also proposes a second solution where *predictions are proxy-assisted.* The proxy sends to the client all the packets of an interval in a single burst, and then informs the client that the transmission will go on after a sleep time period. The client can set the WNIC to the sleep mode for all that entire time period. [11] achieves a total energy saving which ranges from 65% to 85% when using client side predictions and from 80% to 98% with proxy-assisted predictions. These results are worked out by summing up contributions from both the WNIC, which is set to the sleep mode during inactivity time periods, and the CPU whose utilization (for decoding) is decreased thanks to transcoding techniques used to shrink the stream.

The power management solution that we propose makes use of a proxy-based architecture where the proxy is in charge of planning transmission to the client: it alternates periods of transmission and periods of non-transmission and warns the client of a non-transmission period when it starts. It also informs the client of the duration of the next non-transmission period so as it can set the WNIC' sleep mode for all that entire period. As novelty we propose that plans of transmission are dynamically obtained by considering the available bandwidth on the wireless link together with the current client buffer level.

We concentrate on *Stored Audio Streaming* applications and mainly refer to audio files which are MP3 coded. Since MP3 compression already reduces the quality of the audio stream to a boundary limit, we decide not to make use of transcoding techniques so as to preserve the remaining audio quality. We believe that only video streams can take advantage from transcoding techniques because they allow significant shrinkage of the file without severely affecting the playback quality (see [11]).

3. Proxy-based Architecture and Power Saving Strategies

The proxy-based architecture we refer to in our solution is depicted in Figure 1: the connection between the mobile host and the fixed host is split like in the Indirect-TCP model. Moreover, on the wired and wireless connections two different goals are targeted: on the wireless link the major goal is to reduce the energy consumption whereas on the wired connection a classical approach to the streaming with smoothness goals is provided.

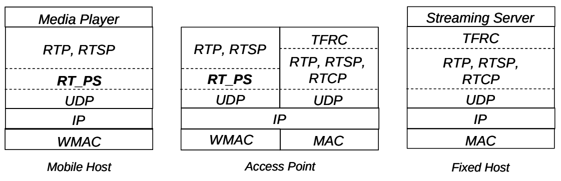

Figure1. Streaming architecture for a wireless scenario.

This differentiation is due to the fact that by reducing the peak transmission rate the traffic smoothing also increases the total transfer delay over the Internet thus conflicting with the power saving objectives.

The streaming server is located at the fixed host and sends streams to the Access Point. Its scheduling policy is smoothness-driven. Real-time data are encapsulated in *RTP* (*Real Time Protocol* [14], [15]) packets while the *Real Time Streaming Protocol* (*RTSP*) [19] is used in order to exchange playback control information with the other streaming peer.

Since RTP uses UDP as underlying transport protocol, which provides no built-in congestion control mechanism, the presence of multimedia traffic in the Internet can potentially lead to congestion. The *TCP-*

Friendly Rate Control (*TFRC* [21]) protocol adaptively establishes an upper bound to the server transmission rate thus leading to congestion avoidance. RTP and RTCP (*Real Time Control Protocol* [21]) packets are used in order to carry TFRC information [22]. Proxy facilities are located at the Access Point as well as a running daemon which is in charge of relaying streams arriving from the server to the mobile host.

The *RT_PS (Real Time and Power Saving)* protocol that we have designed is located on top of the UDP transport layer on both the Access Point and the mobile host and is responsible for the power management during a streaming session. Since the main contribution of this paper is energy efficiency, hereafter we will only concentrate on the mobile host and the Access Point and will assume that an application layer with the streaming server is present at the Access Point.

According to the RT_PS protocol the proxy schedules packets to the client in an on-off fashion. During a streaming session, the time can be subdivided, from the wireless net standpoint, in transmission periods, i.e., on-periods, and non-transmission periods, i.e., off-periods. During on-periods the proxy transmits frames to the mobile host *at the highest possible rate* whereas during off-periods the mobile host sets the *WNIC to the sleep mode*. The traffic shaping that we propose exploits *a priori* knowledge of the frame lengths, the knowledge of the *client buffer size* and an estimate of the current *available bandwidth* on the wireless link which corresponds to the maximum throughput that the proxy can exploit for transmission. During an on-period the proxy decides whether to stop transmitting or not depending on the available bandwidth: when high it continues transmitting until it fills up the client buffer; when low, it stops transmitting in order to avoid increasing congestion and transfer delay. The duration of an off-period is decided by the proxy when it stops transmitting and depends on the client buffer level: an off-period ends when the client buffer level falls down a dynamic threshold, *low water* level, that warns the proxy about the risk of a playback starvation. The low water level depends on the available bandwidth too: when the current available bandwidth is low then the low water level is high in order to preserve a large supply for the playback process; when the current available bandwidth is high the low water level is lower since the proxy can quickly feed the buffer and the risk of underflow is very low. Finally, in order to avoid bandwidth wastage, only frames that are expected to arrive in time for their playback are delivered to the client whereas the others are *discarded*. The computation of frame arrival times makes use of the estimate of the available throughput on the wireless link.

Together with the presence of a buffer at the client, where arriving frames are stored in advance of their playback times, our solution also exploits an initial playback delay and the amount of pre-buffered data depends on its duration. Anyway, it is worth noticing that an initial playback delay and a client buffer are always needed in case of streaming applications in order to consider the transfer delay and absorb the jitter.

4. The RT_PS Protocol

The *RT_PS* protocol is distributed on a *client* at the mobile device and on a *server* at the Access Point. The *RT_PS server* main facilities are: transmitting the audio frames to the mobile host; calculating the client buffer level by keeping into consideration the progression of both the transmission and the playback processes; evaluating the possibility to stop transmitting and let the WNIC of the receiving mobile device assume a low power consuming state; calculating the duration of the sleep periods for the WNIC of the mobile device while preventing the playback process to starve due to the client buffer underflow.

The *RT_PS client* main facilities are: sending the request for the audio file to the proxy also specifying the available buffer size; receiving and interpreting the arriving messages from the proxy containing either audio content or commands; collecting the audio frames in a buffer where the media player can take and play them out; evaluating the available bandwidth on the wireless channel and forwarding its estimates to the RT_PS server; updating the playback start point upon proxy notification; triggering the playback process when the start point elapses; commanding for the WNIC to be set to the sleep mode upon receipt of a sleep command from the proxy and for its raising up when the sleep time finishes.

Details on the key components of the RT_PS protocol follow in Sections 5 and 6. Due to the lack of space, the complete description of the RT_PS server and client behaviours is omitted here, however it can be found in [16].

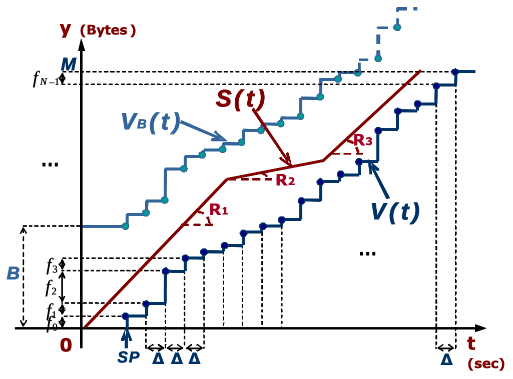

Figure2. Streaming model under real-time constraints: graphical model.

Table1. Symbols used in the Streaming Model

Symbol	Meaning
Δ	The time interval between two consecutive frames.
N	The number of frames in the multimedia file.
$f_i,\ 0 \le i < N$	The size of the i-th frame in the multimedia file.
$M = \sum_{i=0}^{N-1} f_i$	The size of the multimedia file.
$B \ge \max_{0 \le i < N} f_i$	The size of the client buffer,
$S(t)$	Scheduling function.
R_i	i-th transmission rate.
SP	Start Point: when the playback begins.
$V(t)$	The playback curve.
$V^B(t)$	The upper bound constraint to the scheduling function.

5. Computation of Start Point and Sleep Time Intervals

Before describing the computation details that this section is devoted to, an introduction is needed on the specific terminology that will be used. It follows.

5.1. Analytical model

An MP3 audio file is a sequence of frames and its playback begins at the *start point* (*SP*) which specifically is the time when the first frame of the file is played out. From then on, each subsequent frame must be played out starting at fixed, equally spaced, times called *playback times*. In Figure 2 the playback process is represented in a graphical model ([1], [23]) whose symbols are explained in Table 1.

Given Δ the time that a single frame playback lasts[1], *V(t)* is the playback function and represents the total number of bytes played out by the time *t*:

$$V(t) = \begin{cases} 0 & \text{when} \quad t < SP \\ \sum_{i=0}^{k} f_i & \text{when} \quad t \in \left[SP + k\cdot\Delta,\ SP + (k+1)\cdot\Delta\right) \end{cases}, \qquad (1)$$

where k = 0..N-1. During a streaming session, frames arriving to the client for playback are collected in a buffer: let the size of the buffer be *B*. The playback process extracts one single frame at a time from the client buffer at their playback times. Therefore the client buffer level, at the time instant *t*, is the difference between the total number of bytes arrived at the client by the time *t*, and the total number of bytes already played back, i.e., *V(t)*. Playback times are actual deadlines for frames arriving from the server to the client and consequently for frames delivered at the server.

S(t) gives the total number of bytes which the server has already sent to the client by the time *t*. Since the server must transmit enough data to the client in order to support the playback process and avoid client buffer underflows, the relation: $S(t) \ge V(t)$ must be true. $V^B(t)$ is an upper bound for the server delivery process (being *B* the client buffer size): whenever the server sends more than $V^B(t)$ bytes to the client by the time *t* an overflow occurs at the client buffer.

$$V^B(t) = \begin{cases} B & \text{when} \quad t \le SP \\ B + V(SP + (k-1)\cdot\Delta) & \text{when} \quad t \in \left(SP + (k-1)\cdot\Delta,\ SP + k\cdot\Delta\right] \end{cases}, \qquad (2)$$

where k = 1..N-1. The schedule *S(t)* followed by the server in order to transmit packets to the client must be within the two mentioned curves:

$$V(t) \le S(t) \le V^B(t)\ . \qquad (3)$$

When traffic-smoothing goals are targeted, the scheduling curves are polyline functions with line segments of different slopes corresponding to different transmission rates (R_i is the i-th transmission rate). Our scheduling curve is also a polyline and it alternates line segments with zero slope (off-periods) to line segments with non-zero slope (on-periods). The function *A(t)* –omitted in Figure 1- giving the number of bytes already arrived at the client by the time *t* is such that $S(t) \ge A(t)$ and it strongly depends on the transfer delay between the server and the client and on the jitter. However, when investigating streaming issues it is generally assumed that *S(t)=A(t)*; variations between the two curves are finally taken into account by slightly enlarging the client buffer size so as it can *absorb* both the jitter and the transfer delay. Therefore

[1] Δ is the number of samples inside a single frame divided by the sampling frequency.

hereafter we will assume that *S(t)=A(t).*

5.2. Initial delay computation

A deep investigation on properties of on-off schedules when transmitting VBR traffic at a constant rate *R* can be found in [23] and [24]. They demonstrate that, given the constrained region delimited by the functions *V(t), V^B(t), t=0, y=0* and *y=M* (see Figure 1), it is not always possible to schedule a file in an on-off fashion but when the transmission rate is greater than a minimum value, hereafter referred to as $\overline{R}$ (see below). Moreover when $R \geq \overline{R}$, given *V(t)* the playback function for the file to be played out and *B* the client buffer size, many on-off schedules are possible differing for the start point value and the number, position and duration of the off-periods. Anyway, a *minimum* on-off schedule can be found, *R-envelope E_R(t)* of the sequence *V(t),* and for every possible on-off schedule *S_R(t)* the following is true:

$$\forall t\,,\ 0 \leq t < SP + N \cdot \Delta\,,\ S_R(t) \geq E_R(t)\,; \tag{4}$$

where *N* is the total number of frames which compose the file, and Δ is the time the playback process spends playing a single frame. *E_R(t)* is the nearest possible on-off schedule to the playback curve *V(t)* and its formal definition can be found in [23], however for our purposes the definition of the $E_R[k]$ sequence may suffice [25] which is composed by the samples picked up by the *E_R(t)* continuous function at regular time intervals. It can be easily obtained starting from the end: the last value corresponds to the end of the playout curve at the last playback time when $t = SP + (N-1)\cdot\Delta$ and *V(t)=M,* so $E_R[N-1] = M$. From there, a descending line segment with slope *R* must be drawn which ends at the previous playback time, i.e., Δs before, when $t = SP + (N-2)\cdot\Delta$. If the ending point of the segment is higher than the value of the playout curve *V(t)* (at that playback time) then the ending point itself is the new sample $E_R[N-2]$. Otherwise the value of the playout curve is taken as $E_R[N-2]$. Again, starting from the new sample, a new *R*-sloped line segment must be drawn till the previous playback time. The new sample, again, is the maximum between the ending point of the line segment and the playback curve. By repeating this procedure until the time *t=0* is reached, all the samples of the sequence $E_R[k]$ are obtained. More formally:

$$E_R[k] = E_R(SP + k\cdot\Delta) = \\ = \begin{cases} V(SP + (N-1)\cdot\Delta); & k = N-1 \\ \max\{E_R[k+1] - R_\Delta, V(SP + k\cdot\Delta)\}; & 0 \leq k < N-1 \\ 0 & k < 0 \end{cases}, \tag{5}$$

where $0 \leq k < N$ and R_Δ is the total number of bytes which can be transmitted during the time interval Δ when the transmission rate is *R*.

The minimum rate $\overline{R}$ that allows an on-off schedule to be traced is exactly the minimum rate for which an *R-envelope E_R(t)* of the sequence *V(t)* exists inside the constrained region:

$$\overline{R} = \min\{R \mid \exists E_R(t)\}\ . \tag{6}$$

At the beginning of the streaming session, the RT_PS server works out the minimum rate $\overline{R}$ for the audio sequence to be transmitted and contrasts it with the available bandwidth *R* on the wireless link: the streaming goes on only in case $R \geq \overline{R}$. Each possible on-off schedule *S_R(t)* for the sequence *V(t)* lies above the *R-envelope E_R(t)* of the same curve at a larger distance. This causes an increase in the need for buffer space being the minimum client buffer size [23]:

$$B_R = \max_{0 \leq i < N}\{E_R(SP + i\cdot\Delta) - V(SP + i\cdot\Delta) + f_i\}\ . \tag{7}$$

The playback *start point* is the time instant when playback starts. Assuming that the time when the transmission starts is the time 0, the start point corresponds to the initial delay. Though having a long initial delay grants more time to do pre-buffering at the client, and thus prevents the playback process to starve due to the buffer underflow, some evidences suggest keeping the initial delay short. Firstly, an increase in the initial delay determines a decrease in the QoS that the user perceives because he has to wait more to start listening the selected song. Secondly, as stated in [23] an increase in the initial delay causes an increase in the total idle period (i.e., the sum of all the off-periods) while the total on-period remains unmodified; this leads in our solution to a higher power consumption due to the energy, though minimum, consumed by the WNIC in sleep mode.

Given the initial transmission rate R^0, the minimum possible initial delay corresponds to the distance, measured on the X-axis (*t*), between the *R^0–envelope* $E_{R^0}(t)$ to the *V(t)* function and the *V(t)* function itself. Using the *R^0/2-envelope* $E_{R^0/2}(t)$ of the *V(t)* function instead of the *R^0–envelope* $E_{R^0}(t)$ for the calculation, the initial delay becomes longer and more guarantees are offered against the playback process starvation. The motivation for this assumption comes from the following property of the *envelope* curves by [23]: given two transmission rates R_1, R_2, $0 \le R_1 \le R_2$,

$$\forall t,\ 0 \le t < SP + N \cdot \Delta,\ E_{R_1}(t) \ge E_{R_2}(t)\ . \qquad (8)$$

Therefore, with a higher rate the envelope curve is closer to the playout function and the risk of underflow greater. The lower the rate the farther the *rate-envelope*, the less the underflow risk. An on-off schedule starting with a transmission rate equal to R^0, and an initial delay computed by using the $R^0/2$ envelope to $V(t)$, does not cause buffer underflow unless the transmission rate falls down $R^0/2$. This will be our choice for the initial delay hereafter. However, when the initial transmission rate is particularly low, it is much more convenient setting the initial delay to a boundary limit tailored on the user QoS needs.

5.3. Sleep time-periods computation

The procedure we have just described to calculate the initial delay is also used to calculate the duration of an off-period. Suppose that the server has just scheduled and sent to the client $S(t^*)$ bytes. Suppose that, according to the last estimation provided by the client, the transmission rate is R^*. Then, the maximum possible sleep time duration corresponds to the distance, measured on the $y=S(t^*)$ horizontal axis, between $S(t^*)$ and the R^*–*envelope* $E_{R^*}(t)$ to the $V(t)$ function. Intuitively, by assuming that R^* will be available also in the near future, the mobile host can sleep until the scheduling function reaches $E_{R^*}(t)$. In order to reduce the risk for starvation of the playback process, we use the $R^*/2$-*envelope* $E_{R^*/2}(t)$ curve instead of the R^*–*envelope* $E_{R^*}(t)$. A buffer underflow does not occur unless the transmission rate falls down $R^*/2$. An off-period can start if the calculated off-period duration is greater than the time that the WNIC needs to switch to the sleep mode and subsequently resume: we have considered this time is 1ms. When this condition is not verified then the client buffer level is considered to be a *low water* level because the scheduling function is very close to the playback curve and a transmission interruption is too dangerous. In this case, the transmission must continue even if the available bandwidth on the wireless channel is low. Otherwise the client buffer level is considered safe and an off-period can start. As a final remark, it is worth noticing that the procedure for off-periods' calculation is invoked when the client buffer fills up as well as when the transmission rate becomes low (half the initial value) so stopping the transmission can avoid increasing the congestion and the total transfer delay.

6. Available Bandwidth Estimation

The transmission rate that the RT_PS protocol largely uses for its computations corresponds to the available bandwidth on the wireless channel. This is why the RT_PS server when transmitting (on-periods) forwards packets to the RT_PS client with the highest possible rate so as to nearly saturate the bandwidth available on the wireless channel (though for small periods of time).

We use a very simple and fast technique for the available bandwidth estimation with an accuracy that suffices to our purposes. The RT_PS server sends trains of back-to-back packets to the RT_PS client and for each train the client works out the total number of bytes it contains and the total transfer time since the first packet has been sent to the time the last packet is received. Therefore, the transfer time represents the total time the channel spends to transfer a complete packet train from the sender to the receiver. Once received a complete packet train the client divides the train length (the number of bits arrived) by its transfer time and the result is the current available bandwidth. For a good estimation two key factors must be considered: the packet train length and the size of each single packet. For best accuracy the packet size should be 1472 bytes and a single packet train should include at least 50 packets. The value 1472 corresponds to the maximum possible UDP payload size which does not suffer the MAC layer fragmentation[2]. Experimental results show that under these best conditions the maximum available bandwidth on a WLAN network is about 6.35 Mbps when the WNIC data rate is 11 Mbps, 3.95 Mbps when the data rate is 5.5 Mbps and 1.69 Mbps when the data rate is 2 Mbps.

However, in our implementation, estimates are worked out on the RT_PS traffic so a packet is an RT_PS message which includes an RTP packet. The RTP packet is further composed by *ADU descriptor - ADU frame* couples which cannot be fragmented for the sake of loss tolerance [19]. This leads to a great variety of packet lengths and a decrease in estimation accuracy. We have collected an integer value (greater

[2] When packets transmitted during the streaming session are shorter than 1472 bytes the overhead times introduced at the MAC layer (according to the 802.11b standard) for each single transmission have a greater impact on the total transmission time and thus on the resulting calculated throughput.

than one) of *ADU descriptor - ADU frame* couples into a single RTP packet in order to keep the RT_PS packet lengths as close as possible to the ideal value and better exploit the wireless channel capacity. We have also increased the packet train length to 60 instead of 50 in order to make the total train length, in byte, closer to the best value. This is generally possible in our scenario except when the client buffer is small; in this case the estimation accuracy decreases.

Nevertheless, even when working in absence of competing traffic the streaming session cannot experience the best throughputs mentioned above. In fact, according to the RT_PS scheduling algorithm, packets cannot be transmitted back-to-back due to the evaluation of the sleep conditions in between them. Hence, the bandwidth estimation that the client provides during the streaming session is necessarily error-prone. Results show that a streaming session, in absence of background traffic, is able to exploit up to 5.84 Mbps (instead of 6.35 Mbps) with 11 Mbps data rate, up to 3.72 Mbps (instead of 3.95 Mbps) with 5.5 Mbps data rate, and 1.63 Mbps (instead of 1.69 Mbps) with 2 Mbps data rate.

When changing the client buffer size another factor appears to affect the bandwidth estimation: the number of control messages exchanged between the client and the server. When the client buffer is large, the RT_PS schedule produces a small number of very long off-periods. This results in a very bursty traffic where transmissions are concentrated in small periods alternated with long interruptions. As the client buffer size decreases the number of interruptions increases whereas their lengths decrease. In order to manage this schedule, more control messages have to be sent, especially SLEEP messages [16]: they are shorter than the AUDIO messages and therefore responsible for the throughput to decrease. This behavior is much more evident when the data rate is 2 Mbps because the number of off periods is even greater. With a 5.5 Mbps data rate, variations in the available bandwidth estimates range from 9 to 15%; with a 2 Mbps data rate, they range from 4 to 29%. When adding background traffic to the streaming flow the estimated available bandwidth decreases accordingly. In our case, i.e., with only one background flow, even when the requested background throughput grows up, the available bandwidth estimated by the RT_PS client, never goes down 900 Kbps, with a data rate of 2Mbps, 1.8 Mbps, with a data rate of 5.5 Mbps, or 3.2 Mbps with a data rate of 11 Mbps, thus resulting in a fair bandwidth allocation between the two competing traffics.

7. Performance Evaluation

In order to evaluate the effectiveness of the *RT_PS* protocol, we have implemented a software prototype and tested it. An experiment consisted of a single streaming session where the server had to transmit to the client the Mp3 audio file it had previously requested. We have repeated the experimentation with different client buffer sizes from 1 Mbytes down to 50 Kbytes and we have evaluated the impact of some background traffic by adding a competing CBR traffic flow during the streaming session. Moreover, we have repeated the same experiments by changing the data rate of the WNIC. During each single experiment we have manually set the WNIC to work at a constant data rate of either 2 Mbps or 5.5 Mbps or 11 Mbps.

Table2. I_PS vs. Client Buffer Size
Best Case: 11Mbps Data Rate, No Background Traffic

Client Buffer Size (Kbytes)	Average Transmission Rate (Mbps)	$\frac{T_{SLEEP}}{T_{SLEEP}+T_{ON}}$ (%)	*I_ps* (%)
1000	5.74	96.69	9.76
500	5.76	97.32	9.17
200	5.84	97.24	9.24
100	5.82	97.23	9.25
50	5.62	96.79	9.66

In order to measure the energy saving, each experiment produced the power saving index, *I_ps*, as the ratio between the energy consumed by the WNIC during the streaming session when the *RT_PS* layer worked and the energy that it would have consumed if the *RT_PS* layer had not worked [6], [7]:

$$I\_ps = \frac{E_{RT\_PS}}{E_{\overline{RT\_PS}}} \ , \tag{9}$$

where:

$$E_{RT\_PS} = (T_{ON} \cdot W_{ON}) + (T_{SLEEP} \cdot W_{SLEEP}) \ , \tag{10}$$

$$E_{\overline{RT\_PS}} = W_{ON} \cdot (T_{SLEEP} + T_{ON}) \ , \tag{11}$$

T_{ON} giving the sum of all the on-periods, T_{SLEEP} representing the sum of all the off-periods, and finally considering $W_{ON} = 750\ mW$ and $W_{SLEEP} = 50\ mW$ [17].

As shown in Table 2, in absence of any interfering traffic and when the data rate is 11 Mbps, the achievable *I_ps* ranges from 9.17% to 9.76% depending on the client buffer size. Hence, the best power saving is 90.83% of the energy globally consumed when no power aware policy is provided.

When decreasing the data rate the *I_ps* grows up and thus the energy saving decreases. When the data rate is 5.5 Mbps, the *I_ps* ranges from 10.05% to

10.92% (up to 89.95% energy saving); whereas, when the data rate is 2 Mbps it ranges from 14.54% to 16.26% (up to 85.46% energy saving).

Table3. I_PS vs. Client Buffer Size.
Worst Case: 2Mbps Data Rate, 2Mbps Background Traffic.

Client Buffer Size (Kbytes)	Average Transmission Rate (Mbps)	$\frac{T_{SLEEP}}{T_{SLEEP}+T_{ON}}$ (%)	*I_ps* (%)
1000	1.017	84.84	20.82
500	1.063	85.80	19.92
200	1.037	86.38	19.38
100	.995	85.13	20.55
50	.854	81.71	23.74

The *I_ps* also goes up when some competing traffic is added because, in order to prevent the playback process starvation, the WNIC is kept *ON* and working for much more time and this results in a lower energy saving. The greater the throughput of the background traffic, the higher the *I_ps*. In Table 3 the worst performance of our solution is presented. It is obtained by combining the minimum data rate (2 Mbps) and the maximum transmission rate of the background traffic (2 Mbps). As can be seen the *I_ps* equals 23.74% in the worst case thus leading to 76.26% energy saving.

In both Table 2 and 3 *I_ps* variations are evident when the client buffer size changes. These variations are not easy to foresee because they are connected to the fluctuations of the ratio:

$$\frac{T_{SLEEP}}{T_{SLEEP}+T_{ON}}, \qquad (12)$$

where both T_{ON} and T_{SLEEP} always change together and with the same sign. When T_{ON} grows up due to a decrease in the transmission rate, also T_{SLEEP} grows up because of the increase of the initial delay. On the other hand, when T_{SLEEP} goes down due to a transmission rate increase, also T_{ON} goes down because less time is needed in order to transfer the same number of bytes. Experimental results show that when the client buffer size decreases, the ratio (12) increases until it reaches a maximum value, then starts decreasing. The *I_ps* ratio, consequently, decreases first, then increases. The best *I_ps* value (the minimum) corresponds to a client buffer size which ranges from 500 Kbytes to 100 Kbytes. However, in most experiments it corresponds to 200 Kbytes.

Finally, we have repeated the experimentation by changing the burstiness of the competing traffic flow. In this set of experiments the CBR traffic generator has delivered, instead of one single packet at a time, one burst of packets at a time. We have used bursts of 2, 4, 8, 16, 32 and 264 packets in different experiments and we have found that as the number of packets per burst increases, the ratio (12) also increases and hence the *I_ps* goes down (even though very slowly). This behavior can only be seen when the transmission rate of the background traffic is less than the maximum throughput that it can achieve on the wireless link. In Table 4 the *I_ps* changes are shown when the number of packets sent per single burst of the background traffic also changes. These results have been obtained with a data rate of 5.5 Mbps and background traffic of 1 Mbps.

Table4. I_ps vs. Background Traffic Burstiness:
1Mbps Background Traffic, 5.5 Mbps Data Rate

Packets per burst	Average Transmission Rate (Mbps)	$\frac{T_{SLEEP}}{T_{SLEEP}+T_{ON}}$ (%)	I_ps (%)
2	2.73	94.96	11.37
4	2.75	94.94	11.39
8	2.77	94.96	11.37
16	2.83	95.04	11.30
32	2.90	95.52	10.85
264	3.04	95.88	10.51

During the experimentation, initial playback delays have always been less than 2s. Moreover, the sum of the discarded frames (because expected not to arrive in time for playback) and the lost frames has always been less than 5% of the total number of frames. These losses are negligible since, when transmitting real-time traffic, losses up to 20% of the total number of frames can be tolerated if loss concealment techniques are implemented at the receiver.

8. Acknowledgements

This work has been carried out under the financial support of the Fondazione Cassa di Risparmio di Pisa in the framework of the SMILE project.

9. Conclusion

In this paper we have investigated a wireless scenario for multimedia streaming applications and proposed a solution which provides streaming services between a fixed host and a mobile host while implementing power saving strategies in order to reduce the energy consumption due to the WNIC activity of the mobile device. Specifically, we have focused on audio streaming services for Mp3 files.

Our architecture is based on an indirect-model that splits the connection between the server, on the fixed host, and the client, on the mobile host, in two parts: one between the mobile host and the Access Point and the other between the Access Point and the fixed host.

The power saving strategy is applied to the wireless connection between the Access Point and the mobile host. We have designed an *RT_PS* (*Real Time and Power Saving*) protocol that, by exploiting an adaptive on-off schedule for transmission of the audio frames to the mobile host, periodically provides time intervals where the mobile host can let its WNIC assume a low-power consuming state, i.e., a sleep-mode, and resume it later in order to keep on receiving frames. The RT_PS schedule policy considers that when the available bandwidth on the wireless link is high it is better to transmit until the client buffer becomes full and then stop; when the available bandwidth is low instead it is better to stop first, wait a while and then keep on receiving. Anyway, each schedule decision must consider the risk for the playback starvation (i.e., for the client buffer underflow) and avoid it.

We have implemented a software prototype for both the RT_PS client and server and tested it. Experimental results show that this solution can produce up to 90.83% energy saving when working with 11Mbps data rate, 89.95% with a data rate of 5.5 Mbps, up to 85.46% with a data rate of 2 Mbps. We have also observed that energy savings change with the client buffer size, however high energy savings are experimented also for small client buffers (less than 200 Kbytes). When the streaming session competes with some background traffic another influencing factor arises i.e., the burstiness: the higher the burstiness the higher the saving.

9. References

[1] W. Feng and J. Rexford, "A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video," in *Proc. of the IEEE INFOCOM,* Apr.1997, pp. 58–66.

[2] G. Anastasi, M. Conti, W. Lapenna, "Power Saving Policies for Wireless Access to TCP/IP Networks", in *Proc. of the 8-th IFIP Workshop on Performance Modelling and Evaluation of ATM and IP Networks (IFIP ATM&IP2000),* Ilkley (UK), July 17-19, 2000.

[3] R. Kravets, P. Krishnan, "Power Management Techniques for Mobile Communication", in *Proc. of the 4th Annual ACME/IEEE International Conference on Mobile Computing and Networking (Mobicom '98).*

[4] A. Bakre, B. R. Badrinath, "Implementation and Performance Evaluation of Indirect TCP", *IEEE Transactions on Computers,* Vol. 46, No. 3, March 1997.

[5] M. Meyer, J. Sachs, M. Holzke, "Performance Evaluatioin of a TCP Proxy in WCDMA Networks", in *Proc. of the ACM MobiCom 2002,* Sept. 2002.

[6] G. Anastasi, M. Conti, E. Gregori, A. Passarella, "Performance Comparison of Power Saving Strategies for Mobile Web Access", *Performance Evaluation Journal*, Vol. 53, Issue 3-4, August 2003, pp. 273-294.

[7] G. Anastasi, M. Conti, E. Gregori, A. Passarella, "Balancing Energy Saving and QoS in the Mobile Internet: An Application-Independent Approach", in *Proc. of the 36th Hawaii International Conference on System Sciences (HICSS-36),* Hawaii, January 6-9, 2003

[8] M. Stemm, R. H. Katz, "Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices", in *Proc. of the 3rd International Workshop on Mobile Multimedia Communication*, Princeton, NJ, September 1996.

[9] S. Chandra, "Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats", *Multimedia Systems,* Vol. 9, No. 2, August 2003.

[10] S. Chandra, A. Vahdat, "Application-specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats", in *Proc. of the General Track: 2002 USENIX Annual Technical Conference*, 2002.

[11] P. Shenoy, P. Radkov, "Proxy-Assisted Power-Friendly Streaming to Mobile Devices", *Multimedia Computing and Networking*, 2003.

[12] G. Anastasi, M. Conti, A. Passarella, "Power Management in Mobile and Pervasive Computing Systems" in *Algorithms and Protocols for Wireless and Mobile Networks*, Azzedine Boukerche (Editor), CRC_Hall Publisher, 2005.

[13] Web site of IEEE 802.11 WLAN: http://grouper.ieee.org/groups/802/11/main.html.

[14] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Internet-draft, updates RFC 1889, March 2003, http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-new-12.ps.

[15] H. Schulzrinne, "RTP site" , http://www.cs.columbia.edu/~hgs/rtp/, 1999.

[16] Technical report, IIT-CNR, February 2005, http://bruno1.iit.cnr.it/~bruno/techreport.html.

[17] R. Krashinsky, H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown", in Proc. of the *ACM International Conference on Mobile Computing and Networking* (Mobicom 2002).

[18] R. Finlayson, "A More Loss-Tolerant RTP Payload Format for MP3 Audio", Internet-draft, updates RFC 3119, October 2004, http://www.cs.columbia.edu/~hgs/rtp/drafts/draft-ietf-avt-rfc3119bis-03.txt.

[19] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)", Internet-draft, updates RFC 2326, February 2004, http://www.rtsp.org/2004/drafts/draft06/draft-ietf-mmusic-rfc2326bis-06.txt.

[20] M. Handley, S. Floyd, J. Padhye, J. Widmer, "TCP-Friendly Rate Control (TFRC): Protocol Specification", RFC 3448, January 2003, http://www.faqs.org/rfcs/rfc3448.html.

[21] C. Huitema, "Real Time Control Protocol (RTCP): attribute in Session Description Protocol (SDP)", RFC

3605, October 2003, http://www.faqs.org/rfcs/rfc3605.html.

[22] L. Gharai, “RTP Profile for TCP-Friendly Rate Control”, Internet-draft, August 2004, http://macc.east.isi.edu//tfrc-profile.txt.

[23] J. Zhang, “Optimal buffering algorithms for client-server VBR video retrievals”, *PhD thesis*, Rutgers University, 1996.

[24] J. Zhang, J. Y. Hui, “Traffic characteristics and smoothness criteria in VBR video traffic smoothing”, in *Proc. IEEE International Conference on Multimedia Computing and Systems*, June 1997.

[25] L. Huang, F. Hartung, U. Horn, M. Kampmann, “A Proxy-based TCP-friendly streaming over mobile networks”, in *Proc. of the 5th ACM international workshop on Wireless mobile multimedia*, Atlanta, September 2002.