RC23350 (W0409-162) September 27, 2004
Computer Science

IBM Research Report

The Power-Aware Streaming Proxy Architecture

Marcel C. Rosu, C. Michael Olsen, Lu Luo*, Chandrasekhar Narayanaswami
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

*School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

The Power-Aware Streaming Proxy Architecture

Marcel C. Roşu[§] C. Michael Olsen[§] Lu Luo[*] Chandrasekhar Narayanaswami[§]
[§] IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA
{rosu, cmolsen, chandras}@us.ibm.com

[*] School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA
luluo@cs.cmu.edu

Abstract

To improve energy efficiency, major components of mobile devices implement two operational modes: active and sleep mode. When devices are idle, their components are in sleep mode and power consumption can drop by an order of magnitude or more. When devices are in use, their components sleep for many short time intervals. To conserve energy, components have to transition less frequently between modes, and sleep for longer intervals.

*Our approach to reduce energy consumption uses HTTP proxies to shape incoming WLAN traffic. Our proxies use techniques specific to the HTTP payload to compensate for any negative impact that shaping may have. This paper describes the architecture of a Power-Aware Streaming Proxy (*PASP*), built on the Power-Aware x-Proxy (*PAxP*) framework.* PASP *uses RTSP/RTP semantics to effectively schedule media streams across the WLAN. We compare* PASP *with* PAWP*, which is a* PAxP *extension for web traffic, and identify problem areas.*

1. Introduction

Major components of mobile devices implement two operational modes: active and sleep mode. In each mode, components transition between states with different performance characteristics and power consumption levels. When mobile devices are idle, their components are in sleep mode and their energy consumption can drop by an order of magnitude or more. When in use, mobile devices are typically lightly loaded and their components may sleep for a large number of short time intervals. To conserve energy, mobile devices have to be configured such that their components transition less frequently between active and sleep modes, and sleep for longer time intervals. The sleep intervals of various components are often correlated. For instance, a packet receipt generates an interrupt which activates the processor and the memory.

As processor, memory, display, and (micro)disk become more energy efficient, the operation of the wireless LAN (WLAN) interface accounts for an increasing fraction of the total power consumption. Requirements for higher data rates in noisy environments set a lower bound on the energy consumed by the WLAN interface in active mode. As a result, keeping the interface in sleep mode for longer periods and reducing its sleep-mode power consumption represent the most promising approaches for improving its energy efficiency.

When mobile devices are lightly loaded, their operation is mostly driven by outside events, such as packet arrivals or user inputs. Due to the inherent correlations between component operations, keeping the WLAN interface in sleep mode for longer intervals reduces the number of transitions between power modes of the other device components, such as processor, memory and disk, and keeps them in sleep mode for longer intervals. In addition to hardware technologies, the energy consumed by the mobile device is determined by the client applications and their usage patterns. In particular, web browsing and multimedia streaming are characterized by high energy consumption.

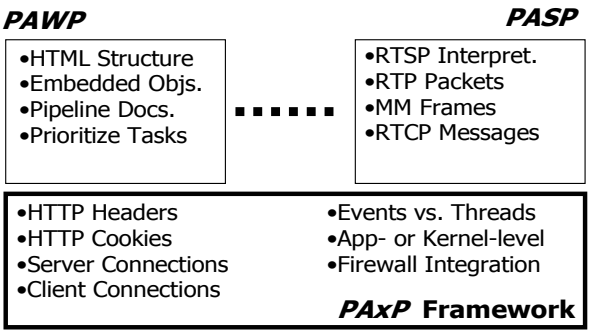

Figure 1. *PAxP* Extensible Architecture

Our approach to reducing the energy consumed by networked applications uses HTTP proxies to shape the WLAN traffic directed to the mobile client. Traffic shaping delays downloaded content to create packet bursts and extend idle intervals. In addition, media streams are transformed to reduce their energy and bandwidth requirements. The HTTP proxies are implemented as extensions to a Proxy-Aware x-Proxy (*PAxP*) framework (see Figure 1), which implements the core HTTP processing tasks. The framework extensions use

techniques specific to the HTTP payload, such as web documents or multimedia streams, to control the traffic shaper and to compensate for any negative impact that traffic shaping may have.

Our approach assumes that the cost of bandwidth across the wide area network (WAN) is low and therefore it is cost-effective to trade increased WAN usage for improved energy efficiency of the mobile device. In addition, it is assumed that the proxy has access to several configuration parameters of the mobile device and WLAN access point.

Our work focuses on the popular 802.11 WLAN technology. The 802.11 specifications define two power management modes: *active* mode and *power save* mode. In *power save* mode, which is the 802.11 term for sleep mode, the WLAN interface consumes 5 to 50 times less power than in *active* mode. A typical WLAN driver switches the interface to *power save* mode after an idle interval of approximately 100 msecs. In such configurations, the relative power consumed by the WLAN interface varies from 5-10% in high-end laptops to more than 50% in PDAs.

The bursty character of the shaped traffic allows the WLAN interface to safely switch to *power save* mode after a shorter idle time interval, i.e., use a smaller timeout value, than typical, which reduces its energy consumption. Due to the hard-to-predict nature of incoming traffic, it is not possible to save energy by reducing the interface timeout without shaping the incoming traffic. Such a simplistic approach increases the roundtrip times of TCP connections, which degrades the performance of client applications.

This paper presents the architecture of a Power-Aware Streaming Proxy (*PASP*), designed to improve the energy efficiency of playing multimedia streams on mobile devices. *PASP* uses RTSP/RTP semantics to effectively schedule media streams across the WLAN. Besides shaping incoming traffic, *PASP* performs domain-specific stream manipulations, such as dropping select video frames and packets, while preserving the consistency of the forwarded stream. The *PASP* proxy is similar to the Power-Aware Web Proxy (*PAWP*), which performs domain specific operations on downloaded web pages to improve the energy efficiency of web browsing on mobile devices. Both proxies are implemented as extensions of the *PAxP* framework (see Figure 1).

The paper is organized as follows: Section 2 provides an overview of the power-management features available in 802.11 WLANs. Section 3 describes the *PAWP* proxy and some of our experiences with the current prototype. Section 4 describes techniques for streaming media over HTTP. Section 5 describes the *PASP* proxy and compares it with *PAWP*. Section 6 is a brief survey of the related work. The last section summarizes our approach and describes future extensions of this work.

2. Power management in 802.11 WLANs

This section provides a brief overview of the power-management features of an 802.11 client interface, or *station,* in an infrastructure network; see [9] for an extensive description.

The power management *mode* of a *station* can be either *active* or *power save*. The power *state* of a *station* can be either *Awake*, when the *station* is fully powered, or *Doze*, when the *station* consumes very little power but it is not able to receive or transmit frames. In *active* mode, the *station* is in the *Awake* state. In *power save* mode, the *station* is typically in *Doze* state but it transitions to *Awake* state to listen for select beacons, which are broadcasted every 102.4 ms by the wireless *access point* (see Figure 2). The *station* selects how often it wakes up to listen to beacons when it associates with the *access point*. The transition between modes is always initiated by the *station* and requires a successful frame exchange with the *access point*.

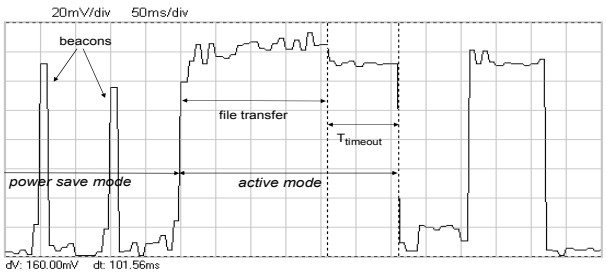

Figure 2. WLAN – Dynamic Power Consumption

The *access point* buffers all pending traffic for the *stations* known to be in *power save* mode and alerts these *station*s in the appropriate beacons. When a *station* detects that frames are pending in the *access point*, it sends a poll message to the *access point*. The *station* remains in the *Awake* state until it receives the response to its poll.

The *access point*'s response to the poll is either the next pending frame or an ACK frame, which signals that the *access point* delays the transmission of the pending frame and assumes the responsibility for initiating its delivery. The *station* must ACK every received frame. If the *More Data* field of the frame indicates additional pending frames, the *station* may send another poll frame. Otherwise, the *station* returns to *Doze* power state.

The *power mode* of the client *station* is controlled by the WLAN device driver. The *station* may switch from *power save* mode to *active* mode at any point in time, e.g., after receiving the first data frame from the *access point*, or after sending a data frame to the *access point*. An example of transitioning from *power save* mode to *active* mode and back is shown in Figure 2. The station will switch back to *power save* mode after no data frames are received or transmitted for a predetermined interval, shown as $T_{timeout}$. Switching from *active* mode to *power save* mode delays receiving any frames until after the next beacon is received.

Switching a client from *power save* mode to *active* mode to receive frames is very advantageous from a performance standpoint, because in the *active* mode the *access point* will forward data frames to the client as soon as they come in, while in the *power save* mode it must queue them up and wait for the *client* to wake up. Unfortunately, in order to absorb variations in packet delivery, the client must remain in *active* mode while waiting for more data, which wastes power. Thus, from an energy standpoint, it is *never* advantageous to transition into the *active* mode except if it is known, or highly expected, that data will be coming in at a very high rate.

Client-side only solutions are restricted by the limitations in predicting the next frame arrival time and by the limitations imposed by the 802.11 specifications. Our approach overcomes these limitations by using an application-level proxy to schedule incoming WLAN traffic in a manner that accounts for client-side configuration.

3. Power-Aware Web Proxy

The Power-Aware Web Proxy (*PAWP)* is the first extension of the *PAxP* framework. *PAWP* [15] is designed to capture the web traffic directed to the mobile client and to shape it into alternating intervals of high and no communication. In contrast to web caching proxies, *PAWP* discards the forwarded objects immediately. *PAWP* does not require any client modifications.

The current *PAWP* implementation does not modify the forwarded content; future implementations may include transcoding functionality. For improved performance, *PAWP* should be integrated with the caching proxy, if present, or with the firewall (see Figure 3).

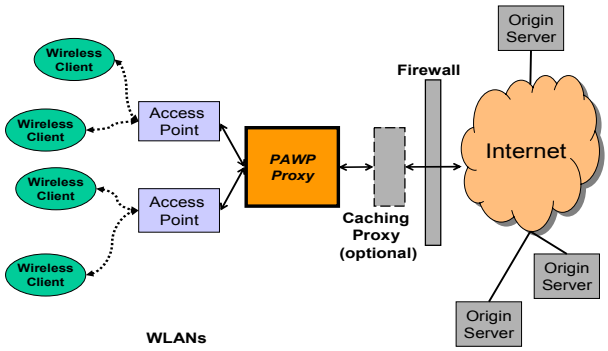

Figure 3. Usage Setting for *PAWP*

To create longer idle intervals, the proxy buffers the downloaded content until one of the data release rules is triggered or until no additional data is expected. Once a transfer is initiated, all the buffered data is forwarded at the maximum WLAN link speed. The proxy aggressively prefetches embedded objects to compensate for the induced delays.

PAWP has four major components, which are shown in Figure 4: the client-side module, the server-side module, the decision module, and the global state module (the blackboard). The client-side module processes HTTP requests from the WLAN clients. If the requested object was already prefetched with the correct cookie, then the client-side module builds the response immediately and it requests permission to send the object back to the client. Otherwise, the request is added to the global data structures in the blackboard module.

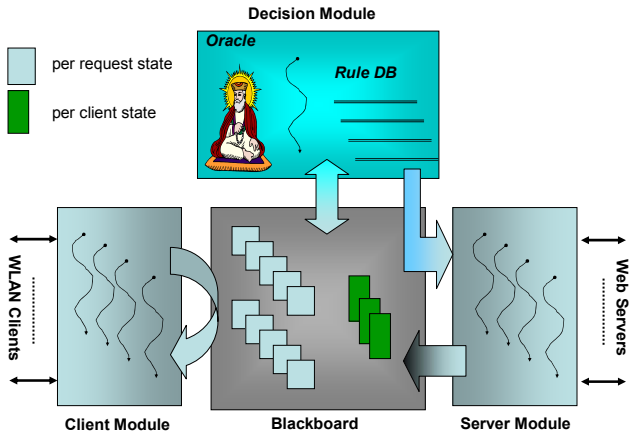

Figure 4. *PAWP* Architecture

The server side-module handles origin servers: establishes and manages the TCP connections, constructs HTTP requests, and adds responses to the blackboard. In addition, this module parses uncompressed text/html responses, generates prefetch requests for the embedded objects, and adds them to the blackboard. The client- and server-side modules use the HTTP processing functionality in the underlying *PAxP* framework. One example of such *PAxP* support is pipelining responses from the server- to client-side module: as fragments of a web object are received, they are stored on the blackboard and the oracle is notified after a configurable threshold is reached. The other two *PAWP* modules make little use of the framework.

The decision module is activated every time the client- or server-side modules change the state of the blackboard. The decision module determines when a client request is forwarded to the server, when a response can be returned to a client, when to reuse a TCP connection, etc. This module acts as the proxy's oracle and its behavior is controlled by an extensible set of rules.

One example of the data release rules is shown in Figure 5. Namely, *PAWP* releases data to a client immediately, if the WLAN interface is known to be active. The *oracle* uses the time of the last client request and $T_{timeout}$ to determine the power mode of an interface. Next, data is never delayed for more than a maximum amount of time, called *MaxDelay*. Whenever more than *MinObjects* are ready to be sent, they are forwarded to the

client; *MinObjects* is set to be smaller than the maximum number of outstanding requests from the client device. Finally, content is forwarded when enough data (say 8K bytes) is available to justify interrupting the idle interval.

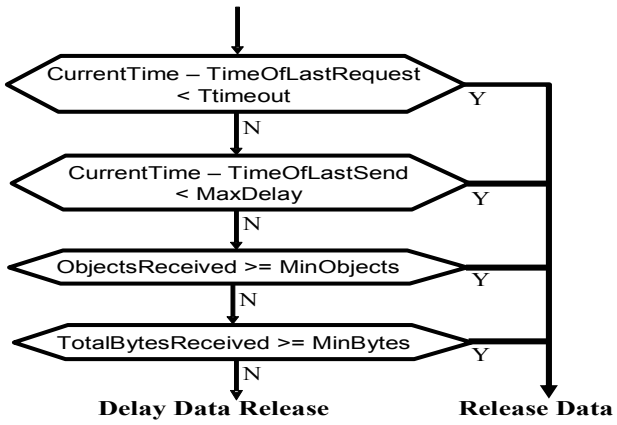

Figure 5. Rules for Releasing Data to the Client

In contrast to no-proxy configurations, where TCP packets arrive at the client in an unpredictable pattern due to the WAN transmission delays, *PAWP*-to-client traffic is characterized by a certain degree of predictability. The proxy effectively indicates to the WLAN client that a short time interval, such as ten milliseconds, of no incoming traffic signals a long enough interval of network inactivity to justify switching back to *power save* mode. Note that higher traffic predictability is achieved at the expense of an increase in the download latency of some objects.

To avoid an increase in the user-perceived latency of downloading a web page, *PAWP* aggressively prefetches all the embedded objects in the page. As a result of prefetching, many of the subsequent client requests are served immediately, without incurring the delay of accessing the origin server and without being delayed by the proxy, as the WLAN interface is known to be active. Typically, the positive effects of object prefetching compensate the negative effects of delaying packets and page download latency is shorter. In addition to object prefetching, this proxy architecture benefits from splitting TCP connections between the WLAN client and origin servers.

PAWP has to address several challenges. First, the proxy must correctly handle HTTP cookies. All client cookies are forwarded and Set-Cookie operations are recorded locally, for later use. Objects prefetched without the proper cookie information are discarded. To lower the likelihood of incorrect prefetches, the architecture includes a mechanism for downloading client cookies into the proxy and for sharing cookies between related proxy installations.

Second, the proxy has to be efficient in prefetching the embedded objects. For instance, *PAWP* attempts to prefetch objects in the order they are expected to be requested by client. In addition, when the client device caches web objects, a large fraction of the client requests are "conditional GETs" and the proxy uses prefetched objects to handle these requests correctly. Because of client caches, some prefetched objects are never requested and they are discarded after a several tens of seconds.

Incorrect and useless prefetches increase WAN bandwidth usage and, most importantly, delay subsequent client requests or useful prefetches sent on the same TCP connection. Due to the heavy usage of web cookies on popular web sites, perfect prefetching is rarely achievable. Our experiments with a large number of web sites show that incorrect prefetches do not impact download latencies if they represent less than 5% of the downloaded objects.

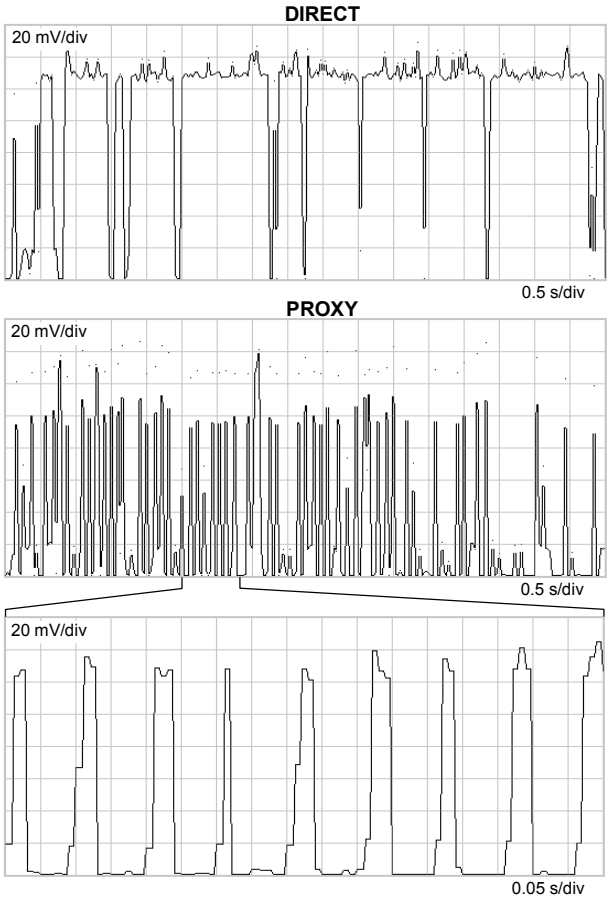

Figure 6. Dynamic Power Traces (eBay.com)

In experiments using popular web sites, such as *cnn.com*, *nytimes.com*, *msn.com*, *amazon.com*, *eBay.com*, *americanexpress.com*, using the proxy reduces the energy consumed by the WLAN interface by 51% to 73%, while increasing the download latency by up to 4%. Figure 6 shows two dynamic power traces collected while downloading eBay.com in 'Direct' and 'Proxy' experiments, respectively. The traces represent the entire download process which lasted 8.0 secs and 6.6 secs in the 'Direct' and 'Proxy' case, respectively. For these experiments, the client device is an IBM ThinkPad T20

running Windows 2000. The WLAN interface is an Intersil PRISM3 PCMCIA card, which consumes 848 mW in the *Awake* state and 25 mW in the *Doze* state.

4. Multimedia streaming over HTTP

Using HTTP for tunneling RTSP/RTP streams has become a popular option for many multimedia players. Moreover, in network configurations where for security reasons HTTP proxies must be used to access the Internet, this is the only option available. Compared to using one TCP connection, HTTP does not provide any additional functionality. In fact, to accommodate HTTP semantics, clients open two TCP connections to the multimedia server.

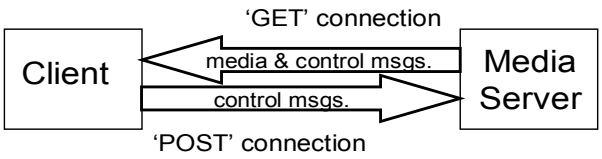

Figure 7. HTTP Connections used for Tunneling

One connection is used to transport the server-to-client data, which includes media stream messages and server control messages. The client opens the first connection using an HTTP GET request (see Figure 7). The initial reply from the server contains only the HTTP header, which includes the 'Connection:close' field; therefore, this connection stays open until closed by the server.

The second connection is used to transport the client-to-server data, which consists of control messages only. The client opens the second connection using an HTTP POST request. The server never replies to this request.

The client has to use HTTP 1.0 for both requests. In addition, the two client headers must include an 'x-sessioncookie' directive with the same value in both requests. The value is expected to be a globally unique identifier and it is used by the server to unambiguously bind the two connections. An optional directive 'x-server-ip-address' is used when the origin site consists of a server cluster using round robin DNS for load balancing. Adding the value returned in the header of the GET response to the header of the POST request insures that both connections are handled by the same server.

The client control messages sent on the POST connection represent RTSP [12] commands, such as DESCRIBE, SETUP, PLAY, PAUSE, and TEARDOWN, and RTCP messages, such as receiver report (RR) and application specific (APP) messages. Using HTTP does not impose any restrictions on RTSP usage; however, RTSP commands are encoded using the base64 method to prevent the proxy from classifying them as malformed HTTP requests. Also, when using HTTP tunnels, existing players rarely send RTCP messages. The value of the 'Content-length' field in the POST request header is set to an arbitrary large value to impose no restrictions on the size and number of RTSP commands or RTCP messages that can be sent to the server.

Most of the traffic on the GET connection consists of RTP [14] messages carrying the media streams. The RTP messages are embedded in the RTSP stream (see 10.12 in [12]), with each media stream using a different channel. The server control messages include the responses to the client RTSP commands and RTCP messages embedded in the RTSP stream, such as sender report (SR) messages. RTCP messages use a dedicated channel. Media servers rarely send RTSP commands to clients. Any server commands would be sent on the GET connection and responses received on the POST connection.

The request and reply headers of the two HTTP requests include directives that instruct HTTP proxies to not cache any elements from the two transactions. A client may close the POST connection at any given time and open another one later, when it needs to communicate with the origin server. HTTP proxies are expected to handle these cases correctly.

5. Power-Aware Streaming Proxy

The Power-Aware Streaming Proxy (*PASP*) is an extension of the *PAxP* framework designed to handle RTSP/RTP multimedia streams tunneled over HTTP. In contrast to *PAWP*, the streaming proxy is designed to alter the forwarded content. Although presented as separate proxies, *PASP*, *PAWP,* and future *PAxP* extensions are expected to be part of the same executable.

To help the mobile client save energy, *PASP* acts similarly to *PAWP* and schedules incoming WLAN traffic in intervals of high and no communication. To enable future energy savings, *PASP* modifies the server-to-client media stream tunneled on the GET connection. In addition, *PASP* modifies the media stream to adapt it to the current WLAN link bandwidth. To hide the effects of traffic shaping or stream transformation from the origin media server, *PASP* intercepts and alters the client-to-server stream, tunneled on the POST connection.

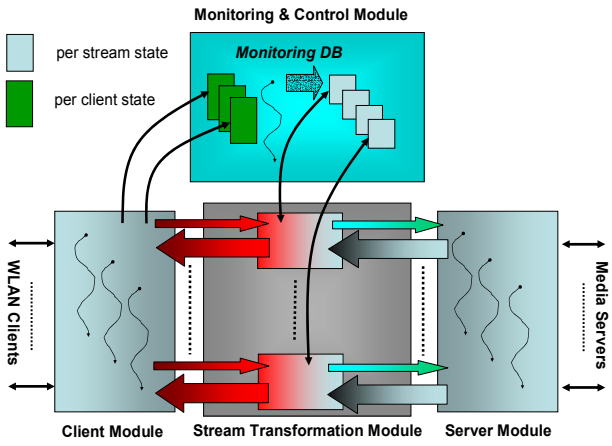

Figure 8. *PASP* Architecture

PASP has four major components, which are shown in Figure 8: the client-side module, the server-side module, the stream transformation module, and the monitoring & control module. The client- and server-side modules are similar to the corresponding *PAWP* modules. In addition, the client-side module collects information about the client link bandwidth and energy level, if available. The stream transformation module handles both server-to-client and client-to-server streams. The monitoring & control module selects the stream transformations using all the available information.

5.1. Scheduling media streams

Scheduling incoming media streams is significantly easier than scheduling web responses due to the more relaxed latency restrictions. More specifically, player applications buffer tens of seconds of content before they start playing a stream. Therefore, incoming streams may be delayed until the player application is close to running out of content to play. In contrast, any delay in the download of an embedded web object translates into an equal increase in the user-perceived delay.

Besides the buffered amount, there are other factors that determine for how long media content can be delayed. For instance, delaying content for one or more seconds may result in the client connection being marked as idle. This may reduce the rate at which the proxy can forward content in the future, as the congestion window of idle connections is reset to the minimum value.

Note that traffic scheduling increases the jitter in the arrival of packets. As long as there is content to play, inter-arrival jitter does not affect user experience but high jitter may determine the player to send (more) Receiver Report (RR) RTCP messages. To prevent the origin media server from reacting unnecessarily and possibly disadvantageously to high jitter reports, the stream transformation module intercepts and updates client RR messages accordingly.

Similar to *PAWP*, when scheduling media streams, *PASP* does not change forwarded content and it only needs to know the configuration of the client WLAN interface. However, *PASP* needs more information to effectively transform the server-to-client streams. This information includes client capabilities, such as screen resolution, computing resources and battery left, and characteristics of the WLAN client association, such as link bandwidth.

5.2. Transforming media streams

A streaming proxy aware of the MPEG-4 stream structure [13] has ample opportunities for transforming a stream to reduce client energy consumption, or to adapt it to the existing WLAN link bandwidth, which is known to vary significantly and unexpectedly in noisy WLAN environments. The adaptation of the stream to the link bandwidth reduces client energy consumption as well, although indirectly: without adaptation, the player application periodically stops for buffering content, which increases total play time and energy consumption.

There is already a significant body of work on transcoding proxies and more recently, on energy-aware transcoding techniques [8, 19]. The proposed transformations selectively drop video frames or alter their resolution. MPEG-4 defines a hierarchical structure for the coded video data, which enables a wide range of stream transformations. For instance, MPEG-4 allows for both natural and synthetic video to be coded and provides access to the individual objects in a scene. Video objects are encoded in one or more video object layers, each one with a different temporal or spatial resolution. Finally, video object planes, which are the representation of a video object in a particular video layer, are segmented into video packets in a way which minimizes the impact of packet loss on user experience.

PASP performs several simple transformations on the media stream. First, it forwards only the video layer that is most appropriate for the client rendering capabilities and its current battery level. Second, it selectively drops the B- and P-video object planes while it forwards as many I-planes as possible. In MPEG-4, the meaning of I-, B-, and P-planes is very similar to the one used in previous MPEG specifications for encoded video frames. Third, for streams consisting of multiple video objects planes, such as synthetic video, it may apply different transformations to different objects. For instance, objects that are too small to be displayed on the mobile device screen may be deleted from the forwarded stream. Finally, dropping individual video packets is the simplest transformation that PASP can apply to the data stream. However, the proxy never drops RTP packets that carry stream configuration information.

The structure of the MPEG-4 video stream and the specifications of the MPEG-4 RTP payload support the identification of the video elements mentioned above. As *PASP* has a better understanding of the client device and WLAN conditions than the origin media server, it is expected that *PASP* can perform a more effective stream transformation than the server. The challenge is to hide as many of these transformations as possible from the origin media server. Otherwise, the origin server may attempt to apply its own transformations, which would increase the complexity of *PASP*'s task.

To hide its stream transformations from the server and to preserve the consistency of the stream, *PASP* intercepts and modifies the client and server control messages. The player application sends updates to the origin server on the characteristics of the stream using RR RTCP messages. Similarly, the server sends updates to clients using SR RTCP packets. In addition, the player may use APP RTCP packets to communicate with the server. *PASP* modifies the RR packet fields that refer to lost packets and inter-arrival jitter. Similarly, *PASP* modifies the SR packet fields that refer to packet and octet count.

These packet changes are necessary because both client and server are aware that the connection between them is reliable, i.e., media streams are tunneled over HTTP.

In addition, modified players and *PASP* can communicate status information, such as battery level, using RR and SR *packet extensions*. These exchanges may require *PASP* to generate new SR packets and to remove or modify RR packets. In addition, *PASP* interprets APP packets with known formats from select media players and acts upon, if necessary.

PASP performs two important tasks. First, it shapes the media stream to reduce the client energy consumption or to adapt it to changing WLAN conditions. Second, *PASP* intercepts and modifies the stream control messages, which are RTCP messages tunneled over HTTP. It modifies control messages to hide *PASP* stream transformations from the origin media server and from the client. As a result, the data and the control messages of a media stream remain consistent, which is expected when reliable protocols, such as TCP or HTTP are used for media transport.

6. Related work

We believe that our work is the first to take advantage at the proxy server of the semantics of the HTTP payload to reduce the energy consumption of the mobile client. HTTP proxy servers have been developed for many other purposes. Most commonly, proxies are used for web caching and as firewall components.

The idea of prefetching web pages to reduce page download latency was previously explored. The authors of [7] found that local proxy prefetching could significantly reduce web latency and that prefetch lead-time is an important factor in the performance of prefetching. A survey of 14 related studies on web prefetching can be found in [4]. The *PAWP* architecture focuses on reducing the energy consumption of mobile clients by prefetching embedded objects.

Proxies are also used for transcoding content to better suit the capabilities and, more recently, the energy consumption characteristics of mobile client devices [8, 19]. Our approach uses a different model for the WLAN interface that is closer to typical implementations and more appropriate for the mix of web and media workloads that mobile devices are expected to handle. In addition, the *PASP* architecture focuses on the challenges of using some of the existing stream transformations with the protocols used for tunneling media over HTTP.

Chandra et al. [2, 3] investigate using a proxy for an application-specific protocol designed to reduce the WLAN interface power consumption when streaming media. Their approach is limited to streaming media applications and requires proxies at both ends of the WLAN. Our approach can be applied to any application that uses HTTP for transport and it does not require installing a proxy on the client device.

Many techniques that reduce the energy consumed by the WLAN interface can be found in literature. The power saving mode of IEEE 802.11b is based on the work of Stemm and Katz, which shows that leaving the WLAN card in sleep mode whenever possible can dramatically reduce the power consumption of the device [18]. At the transport level, the "Bounded Slowdown protocol" [6], introduces a power saving mode that dynamically adapts to network activity and guarantees that a connection's round trip time (RTT) does not increase by more than a preset factor. At the MAC level, Qiao et al. propose to combine Transmit Power Control and PHY rate adaptation to pre-compute an optimal rate-power combination table for a wireless station [11]. Gundlach et al. describe a transport-level scheduling policy designed to burst packets to clients [5]. The implementation uses a transparent proxy placed between the server and the wireless access point. This approach is similar to ours to the extent that both enable periodical release of data. However, as our approach employs HTTP payload information, it is able to better optimize the data delivery to the client. Our approach is capable of handling more complex situations, such as web pages with a lot of embedded objects while theirs cannot.

At the system level, Shih et al. introduce a technique in [17] to reduce the idle power, the power that a WLAN-enabled PDA phone consumes in a "standby" mode. Their approach is to shutdown the device and its wireless card when the device is not being used. A secondary, lower-power wakeup mechanism is used to wakeup the device only when an incoming call is received. Simunic *et al.* describe system-level power management strategies that turn the network interface off completely during idle periods to reduce its power consumption [16]. The STPM algorithm proposed in [1] adaptively manages the power consumption of the WLAN interface using knowledge from application, network interface, and mobile platform.

The work presented in [10] employs an idea similar to ours to manage hard disk power consumption by suggesting the use of aggressive prefetching and the postponement of non-urgent requests in order to increase the average length of disk idle phases.

7. Conclusions

This paper describes a proxy-based approach to reducing the energy consumed by mobile clients when running HTTP-based applications, such as web browsing or media playing. Our approach is to shape the WLAN traffic directed to the mobile client and use techniques specific to the HTTP payload to compensate for any negative impact that shaping may have on application performance. The HTTP proxies are implemented as extensions to the *PAxP* framework. For web payloads, *PAWP* aggressively prefetches embedded objects and pipelines browser and prefetch requests to the origin servers. For media payloads, *PASP* applies encoding-

specific transformations to the media stream to account for the client configuration and its current battery level, and for the current link bandwidth. In addition, *PASP* modifies the associated control messages to ensure their consistency with the modified data stream.

Experiments with several *PAWP* implementations have demonstrated significant reductions in the energy consumed by the WLAN interface and substantial reductions in page download time. We expect *PASP* implementations to enable similar reductions in client energy consumption.

The *PAxP* framework was designed for low latency 802.11-based WLANs. Although some of its elements may provide benefits when used with other wireless technologies, such as CDMA2000, others may not. We plan to experiment with our proxies in networking environments that emulate different wireless technologies.

Acknowledgements. We would like to thank Steve Wood for many useful discussions on MPEG-4.

8. References

1. M. Anand, E. B. Nightingale, and J. Flinn, "Self-Tuning Wireless Network Power Management," In *Proceedings of ACM MOBICOM 2002.*
2. S. Chandra, "Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats," In *Proceedings of MMCN* 2002.
3. S. Chandra and A. Vahdat, "Application-Specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats," In *Proceedings of The 2002 USENIX Annual Technical Conference.*
4. D. Duchamp. "Prefetching Hyperlinks," In *Proceedings USITS 1999*.
5. M. Gundlach, S. Doster, D. K. Lowenthal, S. A. Watterson, and S. Chandra, "Dynamic, Power-Aware Scheduling for Mobile Clients Using a Transparent Proxy," In *Proceedings of ICPP* 2004.
6. R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown," In *Proceedings of ACM MOBICOM 2002*.
7. T. M. Kroeger, D. D. E. Long and J. C. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," In *Proceedings of USITS 1997*.
8. S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Integrated Power Management for Video Streaming to Mobile Handheld Devices," n *Proceedings of ACM Multimedia 2003*.
9. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ANSI/ IEEE Std 802.11, 1999.
10. A. E. Papathanasiou and M. L. Scott, "Energy Efficiency through Burstiness," In *Proceedings of IEEE WMCSA 2003*.
11. D. Qiao, S. Choi, A. Jain, and K. G. Shin, "MiSer: An Optimal Low-Energy Transmission Strategy for IEEE 802.11a/h," In *Proceedings of ACM MOBICOM 2003*.
12. H. Shulzrinne, A. Rao, and R. Lanphier, Real Time Streaming Protocol, RFC 2326, IETF, April 1998.
13. Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, H. Kimata, RTP Payload Format for MPEG-4 Audio/Visual Streams, RFC 3016, IETF, November 2000.
14. H. Shulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, RFC 3550, IETF, July 2003.
15. M.C. Rosu, C.M. Olsen, C. Narayanaswami, and L. Luo, "*PAWP*: A Power-Aware Web Proxy for Wireless LAN Clients", In *Proceedings of IEEE WMCSA 2004*.
16. T. Simunic, L.Benini, P. Glynn, and G. De Micheli, "Dynamic Power Management for Portable Systems," In *Proceedings of ACM MOBICOM 2000*.
17. E. Shih, P. Bahl, and M. J. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," In *Proceedings of ACM MOBICOM 2002*.
18. M. Stemm and R. Katz. Measuring & Reducing Energy Consumption of Network Interfaces in Handheld Devices. In *IEICE Trans. on Fundamentals of Electronics, Communications, and Computer Science*, August 1997.
19. P. Shenoy and P. Radkov, "Proxy-Assisted Power-Friendly Streaming to Mobile Devices," In *Proceedings of the 2003 Multimedia and Networking (MMCN) Conference.*