Energy Conservation in 802.11 WLAN for Mobile Video Calls

Haiyang Ma
Graduate School for
Integrative Sciences and Engineering
National University of Singapore, Singapore
haiyang@comp.nus.edu.sg

Roger Zimmermann
School of Computing
Interactive & Digital Media Institute
National University of Singapore, Singapore
rogerz@comp.nus.edu.sg

***Abstract*—Battery powered mobile devices suffer from significant power consumption of the WiFi network interface during video calls. By utilizing the dynamic Power Save Mode (PSM), our study proposes an adaptive RTP packet transmission scheme for multimedia traffic. By merging the outbound packet delivery timing with inbound packet reception and estimating each delay component along the packet processing and transmission path, each client manages to meet the stringent end-to-end latency for packets while creating longer sleep intervals. As a benefit it involves no cross-layer communication overhead as the interface state transitions are completely transparent to the application. The experimental results show that 28.53% energy savings on the WiFi interface can be achieved while maintaining satisfactory application performance.**

***Keywords*-Energy use optimization, WLAN, Power Save Mode (PSM), 802.11, mobile video calls**

I. Introduction

Video calling has acquired great popularity on mobile platforms in recent years with the advances in video codec efficiency and transmission bandwidth increase, especially the widespread deployment of 3G/4G networks and WiFi Wireless LANs. However, a video call involves many power-consuming hardware components in simultaneous execution and mobile devices are usually powered by capacity constrained batteries. On a modern smartphone, the power consumption of the WiFi interface can reach up to 7 times that of CPU and RAM during data transmission [1], and the WiFi interface has a very high maintenance energy in idle state compared to the cellular radio interface [2]. The fast draining of battery results in mobile video calls of limited duration, generating a hugh gap against user expectation.

To reduce the power consumption of the WiFi interface, Power Save Mode (PSM) was proposed [3], which enables the device to transit to a low-power sleeping state during idle network intervals. However, as PSM will incur additional delays, it is primarily intended for delay-tolerant applications such as web browsing and file download, etc., and was regarded as not applicable to time-critical applications.

In this paper, by utilizing dynamic PSM widely available in current WiFi deployment, we design a packet transmission scheme for delay-sensitive multimedia traffic and show that considerable energy on the WiFi interface can be saved by aggregating the available queuing time for each packet, with negligible impact on the communication quality.

In summary, our main contributions are two-fold: First, we design an adaptive RTP packet transmission scheme for simultaneous audio and video traffic, which require media synchronization and incur various coding delays. Unlike previous cross-layer approaches that force the WiFi cards into state transitions from applications according to pre-calculated sleep intervals, which can harm the underlying communication behavior between client and access point, we only schedule the traffic from the application layer. Second, we implement the adaptive transmission into a real video call system and evaluate its performance. The effects under different WiFi configurations are compared and possible contributing reasons are analyzed.

The rest of the paper is organized as follows. Section II presents a brief overview of related work. Section III introduces the background knowledge of power saving features in 802.11. Section IV shows a detailed analysis of the power consumption and sleeping behavior in video calling and Section V presents the design of the adaptive transmission scheme and how various parameters are obtained. The experimental implementation and comparison of results are illustrated in Section VI. Finally, Section VII concludes the paper and proposes several directions for future work.

II. Related Work

Extensive work has been carried out exploring the trade-offs between energy consumption on WiFi interface. We classify the techniques into several categories.

A. Idling Detection

Since network I/O is largely driven by user interactions, Crk *et al.* [4] predicted network traffic by monitoring users' interaction with applications through the capture and classification of mouse events. Then the NIC was adaptively switched according to traffic modeling and prediction. Dogar *et al.* [5] exploited the bandwidth discrepancy between wired and wireless connections and proposed Catnap, which defers the packet transmission by combining small data blocks into big chucks and creating some idle periods to enable the mobile clients to sleep during data transfer.

B. MAC and Hardware Level Manipulation

Due to the adoption of carrier sense multiple access/collision avoidance (CSMA/CA) as the medium access control (MAC) mechanism, in 802.11 WLANs large amounts of frame overheads are generated, especially for small packets. Based on the current packet loss rate and the target voice quality, Tsao *et al.* [6] dynamically disabled the MAC-layer acknowledgement for each transmitted packet to reduce the consumed time and energy. Rozner *et al.* [7] proposed NAPman, which implements an energy-aware fair scheduling algorithm for both CAM (Constantly Awake Mode) and PSM clients to minimize WiFi radio wake up time and eliminate unnecessary retransmissions. Manweiler *et al.* [8] executed a TDMA-like scheme by dynamically rescheduling an AP's beaconing time in a circle to minimize beacon overlap with nearby APs. These approaches need heavy modifications to the hardware or MAC layer driver code, either for APs or clients. In our transmission scheme we have made no modifications to underlying driver and hardware behaviors.

C. VoIP Traffic Reshaping

Pyles *et al.* [9] proposed SiFi, which leverages statistical analysis on history and builds an empirical distribution function to predict the future silence periods during which a network card will be put into PSM mode. Choi *et al.* [10] utilized a two-way Brady model to classify the talking session into talk-spurt and mutual silence periods, and applied two kinds of PSM for each. Namboodiri *et al.* [11] forcibly put the WiFi interface into sleep based on the calculated playout deadline for each voice packet. Energy saving through silence suppression will not work effectively in video calling as video frames still need to be sent periodically even if delivery of some audio frames can be suppressed. Moreover, we only reshape the inbound and outbound traffic from the application layer and do not force the state transition of network card. This eliminates cross-layer communication overhead and guarantees that the communication mechanism between client and AP remains intact as the AP and clients keep exchanging state information through WiFi management and control frames.

III. Background

A. Power Save Mode

Modern WiFi network interfaces have at least two power modes: Constantly Awake Mode (CAM) where the device stays awake all the time, and Power Save Mode (PSM) where the device periodically wakes up and goes back to sleep in the idling state to save energy [3]. In PSM, the devices informs the AP (Access Point) of their state transition by sending a Null Frame with power management bit either set to 1 (going to sleep) or 0 (going to wake up or remain in the current awake state). The AP will buffer all inbound packets destined for the device in sleep state, and broadcasts beacon frames to all associated devices at a scheduled time interval. Upon reception of a beacon frame, the device can check the Traffic Indication Map (TIM) element to see if there is any buffered data destined for itself. Based on how the device will react to receive the buffered data, PSM can be further classified into static (legacy) PSM [3] and dynamic PSM [12]. In static PSM, the client sends a Power Save Poll (PS-Poll) frame to poll each frame from the AP and goes back to sleep immediately if there is no more buffered data. Since the reception of every buffered frame will initiate a PS-Poll and an acknowledgement frame, the delay incurred by static PSM is generally regarded as unacceptable for even web browsing[12]. In dynamic PSM, however, the client switches to CAM directly in case of any buffered data, and it goes back to PSM if no network activity is observed for a predefined timeout period after data reception. Compared with static PSM, dynamic PSM effectively shortens the transmission delay at the price of less sleeping opportunities.

B. IEEE 802.11e

Quality of Service (QoS) guarantee was proposed in the 802.11e amendment [13], Inside which Unscheduled Automatic Power Save Delivery(UAPSD) is supposed to better support real-time two-way traffic like VoIP [14][15], as an outbound packet from the client can trigger all buffered frames at an AP to be released, negating the beacon waiting and frame polling delays. However, due to the introduction of a more complicated scheduling at the MAC layer such as Service Period and Transmit Opportunity, UAPSD requires clients and APs to be upgraded from the current legacy deployment and is therefore still in limited availability at present. We checked the WiFi Alliance website[1] and found that of all the certified network adapters, only 7.69% support the WMM (WiFi Multimedia) Power Save feature currently (WMM is a subset of 802.11e and uses UAPSD to provide enhanced power saving).

Because of the low transmission efficiency of static PSM and the limited availability of UAPSD, in this paper we use dynamic PSM to design our adaptive transmission scheme for video calling and show that it works effectively in current WiFi deployments.

IV. Transmission Analysis

A. State Transitions under Dynamic PSM

In dynamic PSM, a sleeping client can wake up either to send a frame, or to receive frames if there is any buffered data at the AP. Figure 1 shows the state transitions in three consecutive beacon intervals. For better illustration purpose the acknowledgement frames for every Data and Null Frame are not drawn. We use T_{bc} to represent the beacon interval, T_{out} for the timeout interval and δ for the transmission time

[1] http://certifications.wi-fi.org/search_products.php

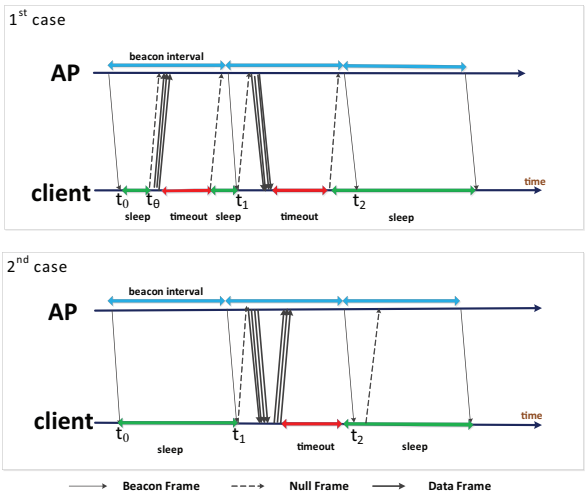

Figure 1. WiFi interface state transitions in adaptive PSM.

of one frame between a client and its associated AP. Suppose there is a beacon frame arriving at the client side at t_0, t_1 and t_2, respectively, and the beacon at t_1 indicates that there are N_{in} buffered frames at the AP. At the reception of each beacon the client uses T_{tim} for beacon checking. Here we assume that a client checks every beacon frame (the WiFi standard defines a integer DTIM value, which enables the configured client to check only once for every DTIM number of beacons. A DTIM value larger than 1 will incur at least 200 ms latency (100 ms for typical T_{bc}), and is thus impractical for video calls).

In the 1st (upper) case, the sleeping client transits to CAM at t_θ ($t_0 < t_\theta < t_1$) to send N_{out} data frames to the network. Since a Null frame is needed to notify the AP of every state transition and all Null and Data frames require acknowledge frames, it takes a total of $2\delta(N_{out}+1)$ to complete the transmission. Then it waits for the timeout value T_{tout}, and hence uses another 2δ for Null frame before going back to sleep. It wakes up again to receive N_{in} buffered data at t_2 and this takes a duration of $2\delta(N_{in}+1)$. We can see that the sleeping time T_{sleep} for the client during the three beacon interval is:

$$T_{sleep} = 3T_{bc} - (3T_{check} + 2T_{tout} + 2\delta(N_{out} + Nin + 4)) \quad (1)$$

In the 2nd (lower) case, the client defers the transmission of the N_{out} outbound packets to the time when it receives the N_{in} inbound packets. Since the AP knows that the client is in CAM state, a Null frame is not necessary for outbound transmission. The sleeping time for the client is:

$$T_{sleep} = 3T_{bc} - (3T_{check} + T_{tout} + 2\delta(N_{out} + Nin + 2)) \quad (2)$$

Compared to the 1st case, the client sleeps $T_{tout} + 4\delta$ longer. Since a timeout period is needed for every transition from CAM to PSM and a state transition incurs overhead (Null frame), in the 2nd case it wakes up only once to process both inbound and outbound traffic. This reminds us that we can reduce state transitions and create more sleep opportunities by merging inbound and outbound traffic timings. We can either defer the reception to the next transmission timing, or defer the transmission to the next reception timing. However, in PSM the device cannot defer the reception of buffered packets that have been broadcast by beacon frames, otherwise they will be dropped at the AP. This means that we must schedule the timing of the outbound packets to be as close as possible to the reception of inbound packets.

B. Delay In Video Calling

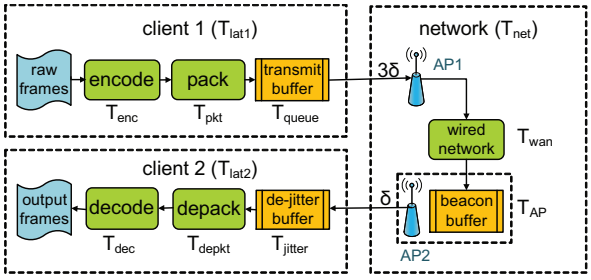

Figure 2. Delay components of video call frames under dynamic PSM.

As Figure 2 shows, in a typical video call, two clients communicate with each other through a direct network connection. As a network inactivity period of T_{tout} is required for entering sleep state, both participating clients have to send packets in batch to create longer networking inactivity, which will inevitably incur additional latencies. The total end-to-end latency for a packet, which is the delay from packet generation time at one client to playout time at another client, should be upper bounded to enable smooth playout, otherwise noticeable jitter will greatly degrade user experiences. We set this upper bound as T_{max}. Figure 2 illustrates the delay components of a multimedia packet. The two participants of a video call, termed client 1 and client 2, are associated with two different APs and the two APs are connected through a wired network (we omit the case where the two clients are associated to the same AP, which is a simplified version of our assumption). We use T_{lat1}, T_{lat2} and T_{net} to denote the processing time at client 1, client 2 and the network transmission latency, respectively. Then the following condition must be met:

$$T_{lat1} + T_{lat2} + Tnet \leq T_{max} \quad (3)$$

Suppose at client 1 a video or audio packet is generated at an interval of T_{gen} ms. After T_{enc} ms encoding and T_{pkt} packetization delay, it waits in a queuing buffer for T_{queue} before it is sent out. Then T_{lat1} can be expressed as

$$T_{lat1} = T_{enc} + T_{pkt} + T_{queue} \quad (4)$$

When the packet is received by client 2 from AP_2, it is not directly sent to the decoder. Instead, it is usually queued into

a de-jitter buffer first to minimize delay variations and sort packets arriving out of order [16]. This queuing will incur a de-jitter delay as T_{jitter}. Then T_{lat2} can be expressed as:

$$T_{lat2} = T_{jitter} + T_{depkt} + T_{dec} \quad (5)$$

where T_{depkt} and T_{dec} represent the de-packetization and decoding latency.

T_{net} includes the transmission time from client 1 to AP_1, AP_1 to AP_2, and AP_2 to client 2. As we have analyzed in Section IV-A, if client 1 stays in CAM before sending, it takes δ to send this frame, otherwise 3δ is needed. For simplicity we assume 3δ is required for the delivery of every outbound packet. The hardware transitional latency for the interface card from PSM to CAM is not accounted for as we will show in the experimental section that such time proves trivial and can be safely neglected. We define the transmission time in a wired network from AP_1 to AP_2 as T_{wan}. Suppose AP_2 receives the packet at t_r, it either transmits it directly to client 2 if client 2 happens to be awake, which takes δ, otherwise the packet must wait in the AP's transmission buffer first. If client 2 remains asleep until the delivery time of the next beacon at t_{nb} ($t_{nb} > t_r$), the packet will be delivered after this beacon's broadcasting. However, client 2 may wake up to send frames at some time between t_r and t_{nb}, which enables the AP to deliver inbound packets immediately after reception of outbound packets from client 2 as the AP knows that the client has transitioned to CAM state. To summarize, T_{net} can be expressed as

$$T_{net} = 3\delta + T_{wan} + T_{AP} + \delta \quad (6)$$

T_{AP} refers to the buffering time at the AP, ranging from 0 to T_{bc}, the beacon interval. This shows that T_{net} is composed of a relatively stable part, $4\delta + T_{wan}$, and a highly volatile part, T_{wan}. From Inequation 3 and Equation 4 we can get that the queuing time for outbound packets should satisfy:

$$T_{queue} \leq T_{max} - T_{enc} - T_{pkt} - T_{lat2} - T_{net} \quad (7)$$

For each client, its own processing time ($T_{enc}+T_{pkt}$) can be obtained during execution. To get T_{lat2} and T_{net}, however, the client has to receive feedback from the other client (peer client). If both clients are working properly, T_{lat1} and T_{lat2} should be stable as well. Combining Equation 6, we rewrite the above inequality as

$$T_{queue}+T_{AP} \leq T_{max}-T_{enc}-T_{pkt}-T_{lat2}-4\delta-T_{wan} \quad (8)$$

We refer to the left part of the inequality as Quota Time T_{qt}. This means that if we know the processing latencies at both clients and the network transmission latency (excluding buffering time at AP), we can obtain a maximal sum of the buffering time at its own transmission queue and at the receiver's AP before delivery. During program executions we have to decide how to allocate the correct portions for each part so that the playout deadline for each packet can be met and the interface can have a long duration of sleep.

V. Transmission Schedule Design

We use SIP (Session Initiation Protocol) to establish and manage the calling session and RTP (Real-time Transport Protocol) to transmit media packets. RTCP (RTP Control Protocol) is utilized to provide execution feedbacks to each client. The generated audio and video packets are queued into the same transmit buffer.

A. Session Establishment

The two participants establish a video call connection by SIP. Apart from the ordinary parameters exchanged in this phase by SIP such as supported codecs and transmission bitrates, we modify the protocol so that the following parameters are exchanged as well: beacon interval (T_{bc}), timeout value (T_{tout}) and initial de-jitter buffer size (T_{jitter}). As at this time T_{depkt} and T_{dec} are not available (actual processing has not started yet), we use the packet generation interval T_{gen} as a conservative estimate, as in most cases T_{depkt}+T_{dec} is much smaller than T_{gen}. With these parameters each client can obtain an initial estimation of T_{lat2} from each other.

B. Exchange of Execution Condition

We make use of RTCP to enable the two clients to exchange instantaneous information on their execution conditions. As in a video call each client works both as a sender and a receiver, they will transmit and receive Sender Report (SR) and Receiver Report (RR) on a regular interval T_{rtcp}. According to the RTCP specification [17], each client can acquire its peer's packet loss rate PR_l since the last report. We extend RTCP so that the packet miss rate PR_m, averaged decoding time T_{dec} and de-packetization time T_{depkt} since the last report are exchanged as well. While T_{dec} and T_{depkt} help the client to calculate a correct quota time Tqt, PR_p and PR_m are utilized in the decision making procedure, as shown in Section V-D.

C. Estimation of Network Latency

According to RTCP, Round Trip Time (RTT) can be derived as:

$$RTT = NTP_{recv} - Delay_{sr} - NTP_{send} \quad (9)$$

This is shown in the upper diagram of Figure 3. NTP_{send} is the timestamp when an initiating client sends a SR to peer client and $Delay_{sr}$ is the delay between when the peer receives this SR and sends a RR as a response. NTP_{recv} is the timestamp this RR is received by the initiating client. Half of the RTT is utilized to estimate network transmission latency. However, calculation of RTT according to Equation 9 is only applicable to the CAM state in a WiFi network. As the lower diagram of Figure 3 indicates, since this RTT calculation involves two transmitted RTCP packets, under WiFi PSM it is actually a sum of the transmission latency in the wired network, latency between the AP and

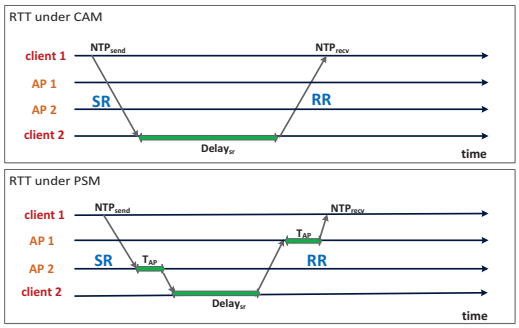

Figure 3. RTT calculations in WiFi CAM (upper) and PSM (lower) states.

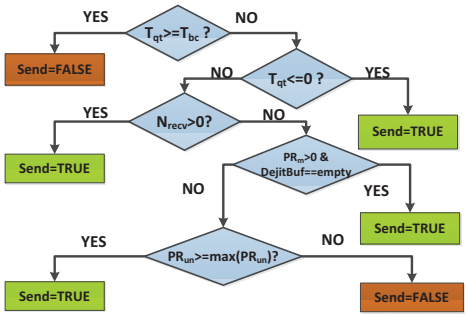

Figure 4. Flow chart of transmission decisions.

the clients, and the possible buffering time at the APs in both directions before being delivered to clients. To differentiate those elements we make the following modifications. After the peer client receives SR, it checks the difference T_{diff} between the receiving time and the last network activity (receiving or sending) time. If T_{diff} is smaller than T_{tout}, it means that the network interface remains active since the last network activity and when this RTCP packet was delivered to itself immediately without buffering at the AP, otherwise it may have been buffered at the AP for a duration upper bounded by T_{diff}-T_{tout}. This maximal buffer time, denoted as T^l_{AP}, is transmitted back in RR. When the initiating client receives this RR, it also calculates a corresponding T^s_{AP} as the maximum buffering time at its associated AP. Then RTT is calculated as:

$$RTT = NTP_{recv} - Delay_{sr} - NTP_{send} - \frac{(T^s_{AP} + T^l_{AP})}{2} \quad (10)$$

That is, we take half of the upper bound as an estimation of the actual buffering time at both APs. If both T^l_{AP} and T^s_{AP} are zero then the calculation is the same as Equation 9. If the derived RTT is not a positive value, this indicates that the actual buffering latency at the two APs are much smaller than $T^s_{AP}+T^l_{AP}$. In this case we disregard the short buffering time and recalculate RTT according to Equation 9. Combined with Equation 6, we can obtain

$$T_{wan} = \frac{RTT}{2} - 4\delta \quad (11)$$

Each time we receive an RTCP packet and derive a new T_{wan} value, the current estimation T_{wan} is updated by an infinite impulse response (IIR) filter:

$$T_{wan} = \alpha T_{wan} + (1-\alpha)RTT; \quad (12)$$

where α is a smoothing coefficient utilized to prevents occasional spike values.

D. Making Transmission Decision

Figure 4 demonstrates the flow chart of transmission decisions. Each outbound packet is denoted as P^n_k where k represents the batch number and n the queuing sequence into this batch. We divide the total program running time into many small Decision Periods T_d. In each T_d, we try to fetch inbound packets from the network interface and then make a sending decision based on the queued packets in the transmission buffer. Although these packets are generated at different times, they will be delivered together in a batch. As such we only have to care about the "oldest" packet (the earliest queued packet) in the buffer currently, i.e, P^1_k, and guarantee that it meets the end-to-end latency requirement. When P^1_k arrives in the transmit buffer, its maximal quota time is calculated as T_{qt} according to Equation 8. If T_{qt} is larger than the beacon interval T_{bc} used by the receiver side, we decide not to send in this decision period. This is because P^1_k will still meet its deadline even if it is buffered at the receiver's AP for a maximum of T_{bc} duration. Then T_{qt} is reduced by T_d since only T_d later we will make the next decision and this also means that the queuing time in the transmit buffer has increased by T_d. On the other hand, if T_{qt} is less than or equal to zero, we must send the packets immediately as P^1_k can only meet its playout deadline if it happens not to be buffered at all at the receiver side's AP. In cases other than the above two, the sending decision is based on the following conditions.

1) Number of Received Packets: If in this T_d we successfully receive inbound packets, we know that the interface must have transitioned to the CAM state. As we have analyzed in Section IV-A, we will send all packets in the transmission buffer at once to reduce network state transitions.

2) Packet Miss Ratio: If in this T_d packet miss happens for play out and there are no more inbound packets queued in the de-jitter buffer, this indicates that packets from the other client failed to arrive in time. We know that either the inbound packets have not arrived at the AP, or currently they are buffered at the AP and waiting to be delivered at next beacon. To prevent further play misses, we send packets immediately as this will enable possibly buffered packets at the AP to be delivered right now, instead of being deferred to the next beacon time.

3) State Feedback from RTCP: Each client checks the packet loss rate PR_{pl} and miss rate PR_{pm} of the peer client from the most recently received RTCP packet. We know that packet misses can be reduced if all packets are sent out early and received in time for playing, while packet loss happens due to problems through network transmission such as congestion and queuing at routers, etc. Apart from the transmission path, bursty transmission of UDP packets through WiFi can also increase packet loss and have a negative impact on effective network throughput. The poor performance of WiFi in dealing with bursty traffic is due to its Distributed Coordination Function (DCF) for access control at the MAC layer, in which case a large portion of time is spent on sensing for free medium and considerable framing overheads are generated [18]. This indicates that a large packet loss rate, as well as packet miss rate from the peer client can both be incurred if we accumulate too many packets before each outbound delivery. As such we use the Packet Unavailable Rate PR_{un} to denote the sum of PR_{pl} and PR_{pm}, where we set an upper bound as PR_{un}^{max}, as a high PR_{un} will inevitably undermine user experiences. If we check that the current PR_{un} exceeds the upper bound, the current packets will be released immediately, otherwise the remaining quota time will be reduced by a value proportional to PR_{un} to accelerate its delivery process:

$$T_{qt} = T_{qt}(1 - PR_{un}/PR_{un}^{max}) \tag{13}$$

To conclude, once we derive the sum of buffering time at the transmit queue and at the peer client's AP, we allocate a dynamic portion to each and make the transmission decision, according to play deadline, packet reception time, packet misses as well as the execution conditions of the peer client.

VI. Experimental Evaluation

A. Experimental Setup

We established two independent 802.11g wireless networks as follows. Two desktop computers, each equipped with a TP-LINK WN951N PCI adapter, operate as two APs with hostapd, a user space daemon for AP management that provides more control options compared with commercial wireless routers. To reduce radio interference, the two APs are placed 5 meters apart and configured to be operating in two frequency channels farthest from each other in the 2.4GHz band (Channel 1 and 11, respectively). Two laptop computers, each equipped with a D-Link DWA-652 Cardbus adapter, are associated with each of these two APs, working as video call clients. We use laptops instead of mobile phones and tablets due to their easy modification and compilation of kernel modules. DWA-652 uses the Atheros AR5008 chipset inside, which is operated by the mac80211 driver under Linux. The specifications for the AR5008 are listed in Table III. We measure the transition delays by instrumenting a time function around the state transition code. The power profiles are taken from the Atheros AR5213 chipset [19]. We modified mac80211 and enables it to write a timestamped record to a log file each time a state transition occurs to calculate the total sleep duration.

We implemented our framework into the open source Linphone as the software client for video calls. To conduct repeatable experiments, we cannot use live multimedia streams from the camera and sound card. Instead, we recorded a two-minute long video clip that features a guided tour around our campus[2], extracted its audio and video track and converted them into raw WAV and YUV formats, which are read frame by frame from disk to simulate a real-time media stream. The parameters adopted for the transmission scheme and coding are listed in Table I, Table II and Table IV. T_{max} is set according to recommendations of ITU-T [20]. Note that δ is obtained by averaging the halved ping results between AP and clients.[3] The beacon checking time for the network interface is automatically deducted in the record file.

Table I
Params for Transmission

Params	Value	Params	Value
T_{bc}	100 ms	T_{max}	300 ms
T_{jitter}	60 ms	T_{rtcp}	2 s
T_d	5 ms	PR_{un}^{max}	0.05
δ	0.78 ms	α	0.8

Table II
Params for Audio

Params	Audio
Codec	Speex
Samp. interval	20 ms
Bitrate	28 kbps
Samp.frequency	32k Hz

Table III
Specs of AR5008

Params	Value
Transit Time (Awake to Sleep)	4-5 μs
Transit Time (Sleep to Awake)	56-58 μs
Power (Awake)	219.6 mW
Power (Sleep)	10.8 mW

Table IV
Params for Video

Params	Video
Codec	H.264
fps	25
Bitrate	192 kbps
Resolution	480x360

B. Experimental Results

Each experiment is run 5 times and the results are averaged. Due to space constraints the results of only one client are demonstrated. As the two clients play a completely equal role in the communication session, the results from each client are consistent and similar with the each other.

1) Default Timeout Interval: In this section we evaluate the performance of our transmission scheme under the default timeout value of 100 ms, as adopted by mac80211. The results are demonstrated in Table V, in comparison with normal transmission, where packets are generated and sent directly without buffering. In the table, A stands for Audio and V stand for Video. $Jitt_{play}$, P_{sp}, D_{sp} and E stand for play jitter, percentage of sleep time, average sleep duration and energy use, respectively. We can see from Table V

[2]Campus video has more dynamic and complex scenes than a conference video, introducing large coding delay variations.

[3]Actually this is not the exact transmission time for a frame between an AP and its clients, which takes only 2-5 μs according to our packet sniffing records from Wireshark, but rather a statistical averaged mean time that takes into account the much larger medium contention overhead.

Table V
PERFORMANCE COMPARISON BETWEEN ADAPTIVE AND NORMAL TRANSMISSION WITH A 100 MS TIMEOUT.

Params		Normal Trans	Adapt Trans
PR_l (%)	A	0.033	0.083
	V	0.143	0.256
PR_m (%)	A	0.0	0.852
	V	0.0	0.0
$Jitt_{play}$ (ms)	A	0.237	1.534
	V	1.375	2.866
P_{sp} (%)	-	**0.0**	**14.445**
D_{sp} (ms)	-	**0.0**	**40.202**
E (J)	-	26.352	22.731

Table VI
SLEEPING CONDITIONS AND ENERGY CONSUMPTIONS BETWEEN DIFFERENT TIMEOUT VALUES.

Params		80 ms	60 ms	40 ms	20 ms
PR_l (%)	A	0.150	1.118	3.453	5.907
	V	0.712	1.282	3.876	6.361
PR_m (%)	A	1.740	2.478	1.747	4.392
	V	0.671	0.643	1.290	2.861
$Jitt_{play}$ (ms)	A	3.460	3.951	5.725	12.52
	V	4.820	5.261	7.754	13.500
P_{sp} (%)	-	**18.564**	**23.019**	**30.754**	**27.132**
D_{sp} (ms)	-	**45.883**	**47.322**	**50.015**	**32.169**
E (J)	-	21.717	20.589	18.835	19.586

that with normal transmission the WiFi interface remains in CAM state and never goes to sleep, as the 100 ms timeout value is much larger than packet generation intervals. While our adaptive transmission scheme has 13.74% savings of energy consumption on the network interface, its performance in terms of packet loss, packet miss and play jitter is still comparable to normal transmissions. Compared with the audio stream, the video stream has a slightly smaller PR_m and larger PR_l, since on average video packets are much larger than audio ones, which makes them more susceptible to being dropped at routers and APs. Moreover, their large play interval (40 ms) makes them more immune to play jitters caused by late arrivals of inbound packets.

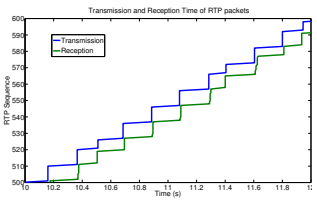

Figure 5. Transmission and reception time of audio RTP packets from 10 s to 12 s.

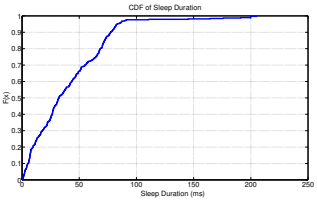

Figure 6. CDF of sleep duration with 100 ms timeout.

Figure 5 shows the transmission and reception time of audio RTP packets in a two-second interval (10s - 12s). The X-axis indicates the running time while the Y-axis indicates the RTP sequence number. It shows that packets are sent and received in batch and most packet transmissions immediately follow the end of receptions, eliminating extra timeouts. Figure 6 shows the CDF (Cumulative Distribution Function) of the sleep duration. We can see that 68.9% of the sleep interval lasts no less than 20 ms.

2) Variable Timeout Interval: It can be observed from the results in Section VI-B1 that the sleep percentage is not significant. This is due to the adoption of a large timeout value (100 ms), as on average only 40 ms can be utilized for actual sleep during a network inactivity of 140 ms. The adoption of a large timeout value in dynamic PSM is to prevent the drastic increase of measured RTT, which has a detrimental effect on TCP throughput with congestion control. However, this is not a necessary concern for UDP traffic. This reminds us that the timeout value can be shortened to enable more sleep opportunities. As such, we choose another four timeout values, i.e, 80 ms, 60 ms, 40 ms and 20 ms, and rerun the experiments. Figure 7 shows

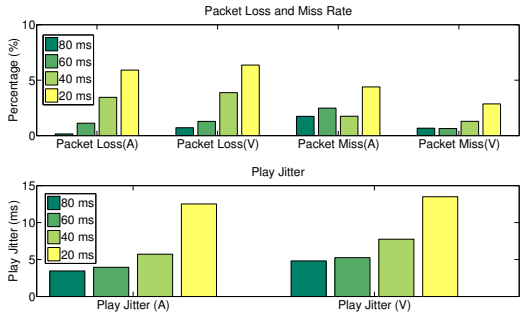

Figure 7. Packet loss rate, miss rate and play jitter with variable timeout.

the packet loss rate, miss rate and play jitter while Table VI demonstrates the sleep conditions and power consumption of the network interface under variable timeout settings. The results of normal transmissions are omitted here as in all cases the interface remains in CAM state for almost all the time and therefore results are nearly identical to the 100 ms settings, except that under a 20 ms timeout setting a negligible 0.73% sleep percentage is observed, due to the small variations in the encoding latency for audio frames.

Table VI shows that energy savings increase with the decrease of the timeout value, while performance for video call slightly decreases but is still within acceptable ranges. With a 40 ms timeout setting the energy savings reach 28.53%. However, the 20 ms case is an exception. It was assumed to be the most energy-saving setting as the smaller the timeout value, the faster it turns into sleep state. This is caused by the high packet loss rate under such settings, which triggers the delivery of outbound packets frequently before they can stay long in the queuing buffer. This is due to the variable latencies experienced by packets transmitted through a network, as packets sent in a batch may not be arriving at a receiver's AP simultaneously. Most APs adopt a normal scheduling algorithm by putting all inbound packets into the tail of a long transmission queue and thus a late-arriving packet for PSM clients would be served much later

[7]. As a result, after a short timeout value the client believes that all inbound packets have been received and goes to sleep, and the late packets are eventually dropped by the AP due to buffer overflow. This also explains why static PSM performs poorly in interactive applications, as it can be viewed as an 'extreme' dynamic PSM with a zero timeout.

Due to the inner working mechanism of WiFi there is a tradeoff between energy savings and application performance, and a timeout value should be carefully chosen with regard to network conditions and scheduling policy, etc.

VII. Conclusions and Future Work

By utilizing the dynamic Power Save Mode (PSM), we propose an adaptive RTP packet transmission scheme for real-time multimedia traffic. It involves no communication overhead between applications and kernel driver. The experimental results show that considerable energy on the WiFi interface can be saved while a comparable application performance can be maintained.

As we have found, the fair scheduling adopted by standard APs incurs undesirable packet losses for clients with an aggressively small timeout value, and this prevents the client from saving more energy by quickly transitioning to sleep state. We plan to investigate this further.

Acknowledgment

This research has been supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

[1] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *the 2010 USENIX Conference*. USENIX Association, 2010, pp. 21–21.

[2] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta, "Wireless wakeups revisited: energy management for VoIP over WiFi smartphones," in *the 5th International Conference on Mobile Systems, Applications and Services*. ACM, 2007, pp. 179–191.

[3] "Wireless medium access control (MAC) and physical layer (PHY) specifications," IEEE 802.11, Jun 1999.

[4] I. Crk, M. Bi, and C. Gniady, "Interaction-aware energy management for wireless network cards," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 371–382, 2008.

[5] F. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, 2010, pp. 107–122.

[6] S. Tsao and C. Huang, "An energy-efficient transmission mechanism for VoIP over IEEE 802.11 WLAN," *Wireless Communications and Mobile Computing*, vol. 9, no. 12, pp. 1629–1644, 2009.

[7] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, "Napman: network-assisted power management for WiFi devices," in *the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, 2010, pp. 91–106.

[8] J. Manweiler and R. Choudhury, "Avoiding the rush hours: WiFi energy management via traffic isolation," in *the 9th International Conference on Mobile Systems, Applications, and Services*, 2011.

[9] A. Pyles, Z. Ren, G. Zhou, and X. Liu, "SiFi: exploiting VoIP silence for WiFi energy savings in smart phones," in *the 13th International Conference on Ubiquitous Computing*. ACM, 2011, pp. 325–334.

[10] H. Choi, J. Lee, and D. Cho, "On the use of a power-saving mode for mobile VoIP devices and its performance evaluation," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1537–1545, 2009.

[11] V. Namboodiri and L. Gao, "Towards energy efficient VoIP over wireless LANs," in *the 9th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2008, pp. 169–178.

[12] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access with bounded slowdown," in *the 8th Annual International Conference on Mobile Computing and Networking*. ACM, 2002, pp. 119–130.

[13] "Wireless medium access control (MAC) and physical layer (PHY) specifications amendment 8: Medium access control (MAC) quality of service enhancement, IEEE 802.11E," Nov 2005.

[14] J. Lee, W. Liao, J. Chen, and H. Lee, "A practical QoS solution to voice over IP in IEEE 802.11 WLANs," *Communications Magazine, IEEE*, vol. 47, no. 4, pp. 111–117, 2009.

[15] X. Pérez-Costa and D. Camps-Mur, "IEEE 802.11e QoS and power saving features overview and analysis of combined performance," *Wireless Communications, IEEE*, vol. 17, no. 4, pp. 88–96, 2010.

[16] T. Yensen, J. Lariviere, I. Lambadaris, and R. Goubran, "HMM delay prediction technique for VoIP," *IEEE Transactions on Multimedia*, vol. 5, no. 3, pp. 444–457, 2003.

[17] "RTP: A transport protocol for real-time applications, Network Working Group," Jul 2003.

[18] P. Verkaik, Y. Agarwal, R. Gupta, and A. Snoeren, "Softspeak: making VoIP play well in existing 802.11 deployments," in *the 6th USENIX Symposium on Networked systems Design and Implementation*. USENIX Association, 2009, pp. 409–422.

[19] X. Zhang and K. Shin, "E-MiLi: energy-minimizing idle listening in wireless networks," in *the 17th Annual International Conference on Mobile Computing and Networking*. ACM, 2011, pp. 205–216.

[20] "ITU-T recommendation G.114. One way transmission time," 2003.