EE-AOC: Energy Efficient Always-On-Connectivity Architecture

Sameh Gobriel, Christian Maciocco, Tsung-Yuan Charlie Tai, and Alexander W. Min
Circuits and Systems Research Lab
Intel Labs, Intel Corporation
{*sameh.gobriel, christian.maciocco, charlie.tai, alexander.w.min*}*@intel.com*

***Abstract*—Mobile platforms, such as laptops, tablets or Ultrabooks, must feature *always-on* network connectivity to make mobile applications (e.g., email, instant messaging, etc.) visible and accessible to/from the Internet. Always-on connectivity for mobile platforms can be achieved conventionally by periodically exchanging keep-alive messages with their counterparts (i.e., servers). However, frequent message exchange wastes energy because it requires the platform to operate in (or transition to) the active state (e.g., S0) instead of remaining in a long, uninterrupted low-power standby mode. To address this problem, we present a novel and secure, yet simple, architecture, called *Energy Efficient Always-On-Connectivity* (*EE-AOC*), that allows mobile devices/applications to be continuously visible and reachable from the network. *EE-AOC* offloads the keep-alive message exchange process to the network device from the host, thus, it does not need to wake up the whole platform. Our experimental results implementing the prototype show that *EE-AOC* achieves always-on connectivity and application reachability at about 85% lower power (i.e. 6.67X gain) than that needed to provide the same functionalities in today's platforms.**

***Keywords*-Always-on-always-connected; connected standby; energy efficient communications; wireless networks; sleep mode.**

I. Introduction

With the skyrocketing use of mobile services and applications in everyday lives, mobile devices, such as tablets, laptops or Ultrabooks, are expected to provide always-on network connectivity anytime anywhere. Unfortunately, always-on network connectivity may incur significant power consumption on mobile platforms because it requires the devices to stay in (or transition to) the active operating state (e.g., S0) [1]—that would require orders of magnitude more energy than low-power standby state (e.g., S3)—to receive and process keep-alive messages, even when the platform is idle. The transition between the standby and active states is very power intensive, and it can drastically shorten battery life, which cannot be tolerated given today's power-hungry mobile devices. Therefore, there is a clear and urgent need for architecture that achieves always-on connectivity for mobile platforms with high energy efficiency.

A typical example of an always-on application is *push email* [2] delivery, made popular by the RIM/Blackberry service, and is now offered by several web-based email services, including Google's Gmail. Users automatically receive email messages as soon as they arrive at the server rather than explicitly having to periodically poll the server to check for new messages. Mobile devices, in this example smart phones, stay in active state (although the display may be switched off) and connected to the network in order to be able to received pushed emails.

While power management features that leverage low-power platform states—i.e., wake up a platform from low-power states only when necessary—have been studied extensively, existing solutions suffer from practical limitations and may not be suitable for mobile platforms. For example, Wake-on-LAN [3] and its wireless equivalent Wake-on-WLAN (WoWLAN) [4] have been proposed as energy efficient means to wake up a platform from the standby S3 state to the active S0 state. However, they are not widely deployed or suitable for mobile devices because they can operate only on a local area network. Moreover, they require modifications to the infrastructure (e.g., access points) to enable wider, Internet-related usages, making them insufficient to fully realize always-on connectivity for mobile platforms.

In this paper, we present a novel architecture, called Energy Efficient Always-On-Connectivity (*EE-AOC*), which enables the low-power operation of always-on and always-connected usage models for mobile platforms. *EE-AOC* allows mobile platforms to enter a low-power sleep state, e.g., S3, without the risk of losing network connectivity, thereby making them continuously reachable to/from the application servers on the Internet. *EE-AOC* offloads the processing of the protocols required for preserving network connectivity to the wireless network communication device (W-NIC) instead of processing them in the host. *EE-AOC* has the following salient features:

- W-NIC in *EE-AOC* handles protocols necessary for maintaining network connectivity, including link layer key refresh, address resolution protocol (ARP) [5], and the presence protocols for various application servers, in a highly energy efficient manner, consuming slightly more energy than the S3 standby state energy consumption. However, *EE-AOC* does not require the W-NIC to have full network stack processing capabilities, hence reducing device cost and ensuring continuous availability.
- W-NIC in *EE-AOC* has the ability to wake-up the platform to the active state upon reception of a generic

network packet from any Internet device. This extends the capability of WoWLAN, which only supports wake patterns based on a magic packets (e.g., a broadcast frame containing a specific number of repetitions of the target device 48-bit MAC address) transmitted on a local area network [4].

- *EE-AOC* allows a platform in sleep state to be woken up securely by targeted Internet services, being highly robust against Denial-of-Service (DoS) attacks, such as an energy drain attacks caused by constant malicious wakes.

We would like to note that although we focus on TCP-based [6] presence protocols, *EE-AOC* is not restricted to TCP and can be applied to any network protocols.

The main contributions of this paper can be summarized as follows:

- We design an energy efficient architecture for mobile platforms, called *EE-AOC*, that maintains device connectivity and application presence to the Internet by delegating the keep-alive message protocol processing to the W-NIC, while the platform remains in a low-power standby state.
- We prototype *EE-AOC* by modifying the firmware of an off-the-shelf Intel WiFi NIC, to demonstrate its efficiency for AOAC usages, e.g., IMAP [7] based pushed emails, and compute continuum [8] usage models. Our in-depth evaluation results show that *EE-AOC* saves about 85% of the overall platform energy.

The rest of the paper is organized as follows. In the next section, we present a literature review related work. Section III highlights the platform energy consumption to maintain network connectivity. Section IV presents our *EE-AOC* framework and describes its architecture and algorithms for offloading the network connectivity and presence maintenance to the W-NIC. Section V evaluates the proposed *EE-AOC* technology and presents the implementation results. We conclude the paper in Section VI.

II. Related Work

Several mechanisms have been proposed to reduce the energy consumption of networked platforms, and various regulatory organizations worldwide are now mandating improvement in the energy consumption of platforms in standby state [9]. Prior proposals can generally be grouped into three categories: (i) those that reduce the active power consumption of systems [10]–[12], (ii) those that reduce the power consumption of the network infrastructure, routers and switches [13]–[15], and (iii) those that opportunistically put the devices to sleep [16]–[18].

EE-AOC falls into the third category and advocates for localized energy-efficient optimization within the platform to extend the sleeping state while maintaining network connectivity, presence and reachability. *EE-AOC* uses the platform W-NIC device for both wake-up and presence, rather than using a network-wide implementation of a proxy service [19]. As mentioned previously, *EE-AOC* is not restricted to a magic packet as defined in WoWLAN, and it supports any wake patterns securely negotiated between the client and the server. In [16], network presence, reachability and application support are realized through the use of an offload processor and application program modifications. In this paper, we show that *EE-AOC* achieves this goal at a very low energy level, without requiring additional hardware. *EE-AOC* saves significant platform energy because it interrupts the host platform only when application support is required.

III. Problem Statement

In this section, we analyze and quantify the potential platform power implications when the platform low-power state is continuously interrupted to exchange keep-alive messages.

A. Mobile Platform Battery Lifetime

Current platform power management policies guide the platform to enter a low-power sleep state (i.e., Sx) when the platform becomes idle for a pre-defined period of time. In general, the longer the system stays in sleep state uninterrupted, the higher the energy gain is, because more peripheral devices and system components can enter a deeper low-power state for a longer period of time.

Recent work has shown that smart timing approaches for Operating Systems (OS) can be used to increase the quiet (idle) time for the OS by skipping timer tick interrupts when the platform is idle or by adaptively changing the rate of timer interrupts (e.g., [20]). This forms the basis for *"tickless OS"*, in which the OS is moving away from scheduling periodic clock interrupts every few milliseconds to a more event-driven approach [21], [22], in which the OS is woken-up to process an event being posted by the applications or by the hardware on demand.

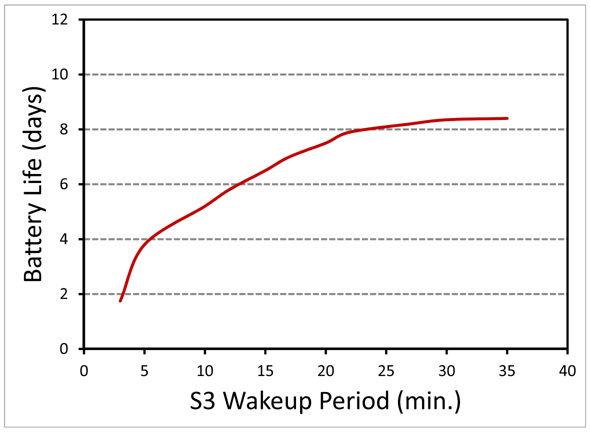

Figure 1. Platform Battery Life vs. S3 Time

Figure 1 illustrates the relationship between typical platform battery life (in days) and sleep time, i.e., the period of time the platform resides in the sleep state (i.e., S3) before transitioning to the active state (i.e., S0). The concavity of the curve indicates that the battery life is sensitive to the sleep time (or the wakeup frequency). It shows that the battery depletes much faster with increasing wakeup frequency and becomes less sensitive to wakeup frequency at longer sleep times (e.g., $> 25\,\text{min}$). The concavity of the curve can indeed be attributed to the considerable energy overhead caused by platform transitions from the sleep state to the active state.

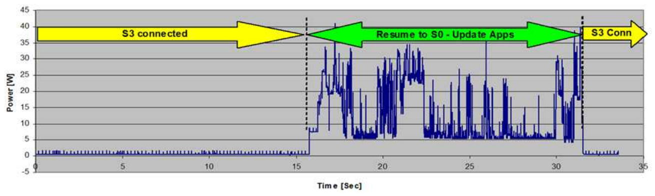

Figure 2. Power Waveform for S3-S0 Transition

Figure 2 shows the power measurement of a laptop when it wakes up from S3 to S0. It indicates that the energy consumed in waking up (i.e, the area under the curve from 16s to 32s) is much higher than the energy consumed when the platform is in sleep state (which is the flat line near 0). The time duration in which the platform stays in the active state before going back to the low-power state depends on various factors, such as the OS scheduling policy, the number of applications running, network connection time, the amount of data exchanged, and network traffic characteristics. Therefore, it is important to minimize the energy overhead due to state transitions to achieve energy-efficient always-on network connectivity.

B. AOAC and Wakeup Frequency

As we mentioned earlier, Always-On-Always-Connected (AOAC) capabilities are crucial for mobile platforms to receive "pushed" data (e.g., an IM message or a tweet update, etc.) from an Internet server in real-time. AOAC is also important for several usage models, in which remote devices need to be discovered and paired together, and thus must be reachable across the Internet (e.g., when a user wants to use his smartphone at a hotel to watch a movie he downloaded on his laptop left at home.)

As a result, connectivity between client platforms and Internet servers must be maintained, which is typically accomplished by exchanging "keep-alive" messages at fixed time intervals. When a connection is established between an AOAC application client (e.g., IM client) and AOAC application servers (e.g., IM server) the connection is kept alive until it expires after a timeout of "T" seconds. If no data is exchanged over this connection for longer than T seconds, the connection is dropped, and the client becomes unreachable by the server (i.e., the server marks the client as offline). Once the connection is dropped, new data (e.g., new IM message sent to the client) will no longer be pushed to the client. However, exchanging keep-alive packets is an energy consuming task because it either requires the client to be in the active state or to transition from a low-power state to the active state.

A key parameter that determines the lifetime of an AOAC device is the frequency of the keep-alive messages. The maximum value of T should be the minimum of T_S and T_N, where T_S is the timeout of the application server and T_N is the minimum timeout of any communication equipment (e.g., Network Address Translation "NAT" box [23]) along the route from the client to the server. These timeouts are in place mainly because each connection has a state (e.g., a state may include source and destination addresses and port numbers) that must be maintained. Due to limited resources and/or improved responsiveness (e.g., faster offline indication) the storage time of the connection state cannot be indefinite and will be dropped after a given timeout. Therefore, AOAC devices must exchange keep-alive messages in a timely manner in order to maintain network connectivity.

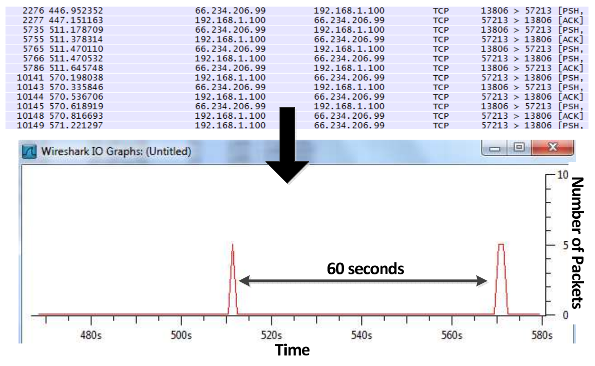
```
 2276 446.952352     66.234.206.99    192.168.1.100    TCP    13806 > 57213 [PSH,
 2277 447.151163     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [ACK]
 5735 511.178709     66.234.206.99    192.168.1.100    TCP    13806 > 57213 [PSH,
 5755 511.378314     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [ACK]
 5765 511.470110     66.234.206.99    192.168.1.100    TCP    13806 > 57213 [PSH,
 5766 511.470532     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [PSH,
 5786 511.645748     66.234.206.99    192.168.1.100    TCP    13806 > 57213 [ACK]
10141 570.198038     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [PSH,
10143 570.335846     66.234.206.99    192.168.1.100    TCP    13806 > 57213 [PSH,
10144 570.536706     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [ACK]
10145 570.618919     66.234.206.99    192.168.1.100    TCP    13806 > 57213 [PSH,
10148 570.816693     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [ACK]
10149 571.221297     192.168.1.100    66.234.206.99    TCP    57213 > 13806 [PSH,
```

Figure 3. Skype IM Application Keep-Alive Message Period

However, it is practically infeasible to accurately predict connection timeout values (i.e., T) at the design stage because of their inherent variability and the unpredictability of the network path between client and server. For example, from the application server side, a typical TCP default session timeout is 2 hours. On the other hand, from the client side, an IM offline indication is usually in the range of 3-5 minutes. Figure 3 shows the network packet trace of a Skype application, in which the client exchanges keep-alive messages with the server approximately once every 60 seconds. Moreover, the network timeout has a wide range of values. Default factory settings of an NAT device timeout can typically range from as short as 15 seconds to as long as one hour. Therefore, AOAC devices need to use a shorter timeout (in the range of tens of seconds) for keep-alive period, i.e., T, which may have a significant power impact as shown in Figure 1. In the next section, we elaborate on how *EE-AOC* overcomes these practical challenges.

IV. EE-AOC Technology

EE-AOC achieves the functionality of exchanging keep-alives with the Internet application server, and hence, maintains the connectivity, presence and reachability of an AOAC device to the network at very low power. This is achieved by offloading a series of "keep-alive" packets to the wireless network interface card (W-NIC), which impersonates the platform in a low-power state to other hosts and servers on the network.

Furthermore, *EE-AOC* allows the NIC to maintain a TCP keep-alive session to the server, modifying only the NIC firmware, without changing the NIC hardware. More importantly, by using TCP at the server side, the network infrastructure pipes (e.g., network switches, network wireless access points, etc.) do not require changes and are not even aware that the platform is not running in the active mode.

A. *Always-On-Always-Connected System Architecture*

In an AOAC system, clients maintain connectivity with the Internet server and keep the network pipe open with keep-alive messages to the server. For example, when there is application data to be pushed to the client, e.g., an IM message or an email, the server will use the established client-server session. Like most Internet traffic, the established session is based on TCP protocol. Consequently, in order for the client to receive live updates from a set of AOAC applications, it has to maintain an alive session with the corresponding server for each supported AOAC application. Obviously, the overhead of connection maintenance increases with the number of established connections.

When the client enters a low-power state between the transmission of keep-alive messages, then from an energy-efficient system design standpoint, it is crucial to maximize the time the client spends in a low power state. Having multiple ongoing AOAC connections, each with its own periodicity, leads to unaligned timing in sending the keep-alive messages, and as a result, the client is forced to exit the low-power state more often.

To reduce unaligned keep-alive message periods, an AOAC system uses a push server, as shown in Figure 4. A typical example of such architecture is Apple Push Notification Service (APNS) [24]. In this case, the client maintains the connectivity and an alive session with only one Internet push server. The server aggregates and proxies data updates for other application servers. As shown in Figure 4, when an update is available for the user, the application server sends a notification to the push server, which then pushes this notification to the client. Upon reception of the notification, the client either (i) exits the low-power state to an active state, so that it can receive the data update from the push server or (ii) can establish a new connection to the application server to retrieve the available update.

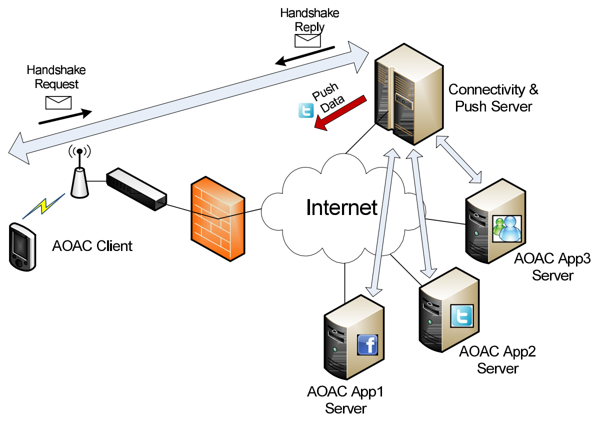

Figure 4. AOAC System Architecture

B. *EE-AOC and Network Infrastructure*

The client is connected to the Internet through a network infrastructure (i.e., routers NAT boxes, proxy servers, etc.), and each component of this infrastructure will keep a state for the client as long as the connection is maintained. Soon after the connection terminates, or if it is not maintained, the network infrastructure will drop the client state to save resources. Then, the client will need to re-initiate its state in the network for new connections.

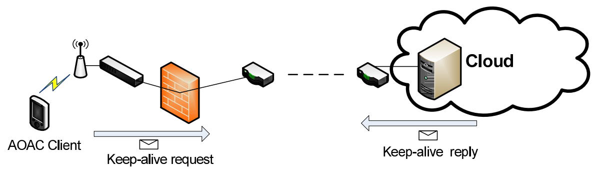

Figure 5. Keep-Alives Handshake through Network Infrastructure

An example of such a network state is firewall flow state that does not block traffic as long as it is part of an existing flow. As shown in Figure 5, when the client initiates the connection, the firewall will check the connection state to decide whether or not it is permissible or not. If the connection is legitimate, then the firewall will keep the traffic belonging to this flow going through and will not block the traffic. When the firewall detects that the traffic has stopped, e.g., with no TCP FIN (i.e., flow end message) being detected, it will drop the state of this flow after a short timeout, unless new packets belonging to this flow are identified. In this case, the client must reset the cycle and reinitiate the request to connect to the server. On the other hand, most firewalls will block the traffic if the flow state is dropped and new data or a new connection request is sent form the Internet cloud server to the client because most firewalls do not allow connections to be initiated from outside.

Therefore, protocols for keep-alive messages must be able to traverse the network infrastructure with default settings (i.e., no special ports to be opened or no special changes to the security policy). Furthermore, the keep-alive handshake process has to be initiated by the client because a server-initiated handshake will not work in the absence of an alive session. In section IV-D, we highlight how our *EE-AOC* technology achieves the above-mentioned properties.

C. EE-AOC and Secure Wake

AOAC clients are always connected to the network, waiting for data updates to be pushed by the push server. This always-on connectivity, hence longer exposure to the Internet, makes mobile clients vulnerable to attacks. An attacker can launch a Denial-of-Service (DoS) attack on AOAC clients by pushing frequent wake-up messages to deplete their batteries faster (see Figure 1), thereby rendering them unreachable to legitimate users.

Another possible attack scenario is that an attacker can impersonate the client by sending fake keep-alive messages to the server on behalf of the legitimate AOAC client. As a result, the server can be misled to believe that the legitimate client is still available and connected to the network. This may cause a data integrity violation because the client's presence can no longer be trusted. Even worse, it can cause data loss if the server forwards the client's private data to the attacker.

Therefore, the keep-alive message exchange must be designed to provide integrity and authenticity of the data exchanged between the client and server. In the next section, we discuss the secure design and overall operations of *EE-AOC*.

D. EE-AOC Software Architecture

EE-AOC enables end-users' software applications/services to be connected to the network application server, while the platform remains in a low-power standby mode. As we mentioned earlier, these applications/services can maintain connectivity by periodically sending keep-alive messages to the application servers. To achieve this goal in an energy-efficient manner, *EE-AOC* defines an interface and network device functionality and capability as we describe next.

AOAC applications intending to maintain connectivity and presence to the network pre-build a list of keep-alive messages for the next few minutes, using each application's proprietary protocol, appropriate sequence number, and periodicity information (if required). Then they are secured with the application key/tokens, and handed over to the communication device along with the appropriate information about each message (e.g., required periodicity to maintain presence, where should it be sent (to which address), etc.) before the platform transitions into the standby state.

Upon transitioning to the low-power state, the communication device performs the following three operations: (i) it recovers the keep-alive messages, (ii) it orders them chronologically, and (iii) it sends these pre-build keep- alive messages to their destinations network at the appropriate time in order to maintain its presence to the application servers. The outline of the *EE-AOC* architecture and the offload algorithm is highlighted in Figure 6.

It should be noted that the idea of offloading TCP has been proposed before. However, full TCP offload is very complex and consumes a considerable amount of scarce resources (e.g., processing, memory, etc.) from the network interface card, to the extent that can hardly be supported by client network card vendors. Although *EE-AOC* defines an offload framework for the NIC, it is simple and requires no modifications to hardware or additional resources from the network card, as will be evident later.

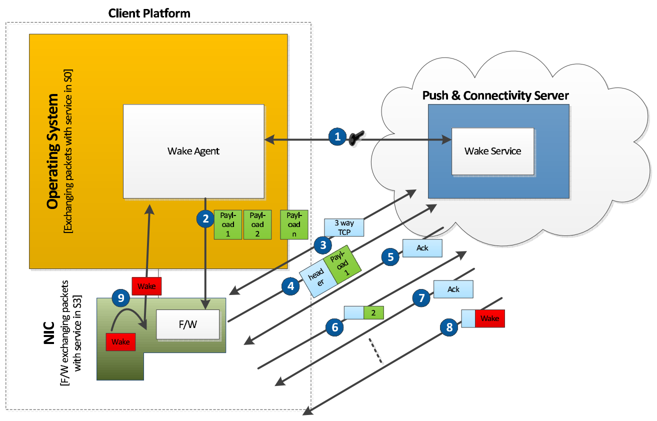

Figure 6. EE-AOC Offload Architecture and Operation

As shown in Figure 6, the offload operations can be divided into two phases. The first phase is to prepare a list of keep-alive messages, and this is executed while the platform is in the active mode; this phase runs inside the wake agent as part of the OS or as an application. The second phase is to exchange the packets with the Internet server in a timely manner to keep the connection alive. This will run inside the NIC firmware when the platform is in standby or a low-power state, while the NIC is in the operating state. We elaborate on the detailed steps of these two phases as follows:

In the platform active state before going to standby:

1) The wake agent registers the platform with the Internet server.
2) The wake agent exchanges keys with the server and establishes a shared private key with the server. [Arrow 1 in Figure 6]
3) The wake agent prepares a train of payloads (multiple payloads) that are encrypted with the shared private key.
4) The wake agent formats the train of packets as http post messages (destination port 80) on top of TCP. Since HTTP/TCP connections are legitimate inside out connections, no firewall will block the connection.

5) The wake agent adds TCP-SYN packet to the head of the packet train, followed by an ACK packet. (needed for the 3-way TCP session initiation handshake)
6) The wake agent transfers the train of packets to the NIC driver. [Arrow 2 in Figure 6]
7) The wake agent downloads the wake filter to the NIC. This is a pattern of a general packet that once the NIC receives and matches, it should wake the platform up.
8) When the driver detects the OS event that indicates that the platform is transitioning to standby it loads the NIC firmware image with the train of packets.

In the standby/low power platform state:

1) The NIC blindly transmits the first packet it got from the train of packets. As mentioned, the first packet is TCP-SYN
2) The server replies to the TCP-SYN with a SYN-ACK message that has its own sequence number that the NIC firmware will extract from the packet and send the Ack message (2nd packet in the train of packets) accordingly. [Arrow 3 in Figure 6]
3) Once the TCP flow is initiated with the server, the NIC sends the http-formated keep-alive packets downloaded from the wake agent.[Arrow 4 in Figure 6]
4) The server decrypts the keep-alive packet, and if the decrypted packet is correct, then an ack message is sent to the client. [Arrow 5 in Figure 6]
5) The NIC keeps a retransmission window of 1 packet and waits for an ack of the packet from the server. If the ack is not received, then the packet is retransmitted for a maximum number of retries. Keeping a window of 1 packet eliminates the need for a complete TCP stack offload because the NIC firmware does not need to handle the sizing of the TCP congestion window or any optional flags in the TCP header.
6) If the maximum number of retries (or any other exception, e.g., platform is out of network coverage) the NIC wakes up the platform to S0 to handle this case.
7) The NIC goes on the train of packet payloads it has in its firmware, sending them one by one to the server. This is done at a pre-determined frequency, typically once every minute. [Arrow 6 in Figure 6]
8) The NIC keeps sending the keep-alive packets until the whole list of packets has been exhausted, and in this case, the NIC wakes up the platform to active state to restart, where the wake agent re-exchanges keys and prepare a new list of packets.
9) If while in standby there is new data to be sent from the server to the client (outside in data), the server simply sends the encrypted wake message as the payload to a TCP message, formatted as an http reply. [Arrow 8 in Figure 6]
10) The NIC decrypts the received packet, and if the wake packet content matches the pre-defined wake filter, then the NIC wakes the platform up to active state.
11) In active state, the application connects to the server to download the new data received, and when this is finished, the platform goes back to standby and the keep alive cycle restarts.

V. Performance Evaluation

We implemented the system described in Figure 4. Specifically, we used the Intel WiFi 5350 [25] NIC firmware to offload the secure keep-alive messages. We had a wake service running on the Internet server to aggregate and push updates to the client when it was in a low-power state. We used two exemplary applications, i.e., Facebook and email, to demonstrate the benefits of *EE-AOC*. We implemented the wake server to periodically monitor the Facebook account of the client and push updates posted on his Facebook wall towards the client. Similarly for an email application, we used push Internet Message Access Protocol (p-IMAP) [26] to push new emails sent to the client as soon as they arrived at the mail server. In our evaluation, we analyze the power consumption and battery life of the platform when using *EE-AOC*.

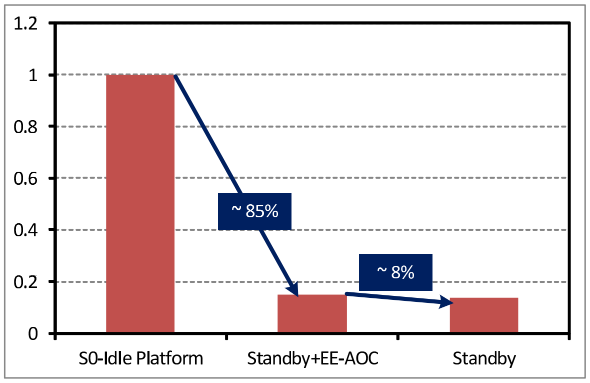

Figure 7. Normalized Power Consumption Comparison

Figure 7 shows the power consumption of a laptop in three operating states. (1) *Fully-on*, that is when the platform is switched on and stays in the active state (i.e., S0), but no active workload is running. (2) *Standby with* EE-AOC *enabled*, that is when the platform is in standby, but*EE-AOC* is implemented in the NIC, which entails that the communication device is turned on and is exchanging the keep-alive messages with the service to maintain connectivity and presence. (3) *Standby*, that is when the platform is in sleep state with no network connectivity. The power consumption in Figure 7 is normalized to the *fully-on* state. This is because we believe that although the absolute power consumption in each of these states will decrease from one generation to the next, and will be different based on the platform itself, the relative power consumption among these states will remain relatively unchanged.

As shown in Figure 7, there is a significant difference in power consumption between the *fully-on* state and the *standby* state. In standby, (a.k.a. suspend to RAM), most of the platform components are turned off and the operating system and application states are saved to the RAM which remains powered. The platform in standby consumes only about 10 % of the power consumed in the *fully-on* state, confirming that standby is a very efficient power saving mode for the client. On the other hand, *EE-AOC* requires additional power for the network communication device, which remains connected to the network, sending and receiving data packets. However, The NIC power consumption is typically very small when compared to the whole platform. Thus, the power consumption of *EE-AOC* is slightly higher (∼8 %) than that of standby mode power consumption, yet significantly lower than lighting up the whole platform.

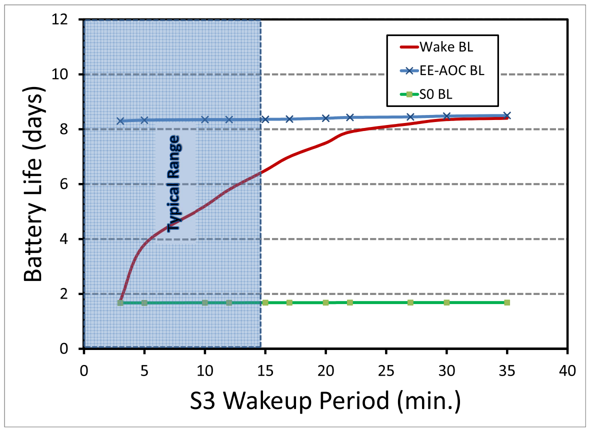

Figure 8. Battery Life vs. Sleep Time with EE-AOC

Figure 8 shows the battery life (in days) of an AOAC client with respect to the frequency of exchanging the keep-alive messages for the following three cases: (1) when the platform is in the fully-on mode and exchanges the keep-alive messages (denoted as S0 BL), (2) when the platform enters the standby state between keep-alive messages (which is the result discussed in Figure 1) and wakes up to S0 when it is time to send a new keep-alive message (denoted as Wake BL), and (3) when the keep-alive messages are offloaded to the NIC with *EE-AOC* (denoted as *EE-AOC* BL).

Figure 8 shows that when the platform stays in the fully-on mode, the battery life of the platform is very short and will only last for less than two days. In this case, the curve is almost constant and the battery life depends less on the periodicity of the keep-alive messages because the energy consumption for transmitting and receiving a packet is negligible when compared the power consumption of the whole platform. However, when the platform enters the standby state between the keep-alive packets, energy consumption is dominated by the overhead of entering into and exiting from the sleep state, leading to the concave-shaped curve of battery life. Unfortunately, without *EE-AOC*, most applications/networks require the keep-alives to be exchanged in the order of every tens of seconds, which leads to a poor platform battery life.

On the other hand, when *EE-AOC* is used, the network communication device exchanges the keep-alive messages on behalf of the platform. Most of the platform components will stay in a low-power mode, while the NIC is turned on to serve the keep-alive messages. The battery in such a case lasts longer than 8 days, which is more than a 4X increase. Moreover, similar to the fully-on case, the curve is almost flat because the energy of sending and receiving packets is small relative to the NIC power. This indicates that the battery life with *EE-AOC* is almost constant, irrespective of the frequency of the packets exchanged.

VI. Conclusion and Future Work

Conventionally, platform power management policies are designed to guide individual platform components or the whole platform into low-power sleep states when the platform becomes “idle”, with no active workloads. Individual power management techniques differ in how deep the sleep state is, the algorithms used to enter and exit the sleep states, and the optimizations to extend these sleep states as long as possible. In this paper, we proposed a novel, yet simple, architecture, called *EE-AOC*, that achieves energy-efficient always-on network connectivity for mobile platforms. *EE-AOC* is very different from existing solutions in a sense that it does not require hardware modifications, while it provides the core functionalities for Always-On-Always-Connected (AOAC) usage models and application visibility to mobile platforms, such as Ultrabooks, laptops and tablets.

As future work we’ll explore how the communication device can help optimize the overall platform power when the platform is in the active state (i.e., S0) either idle with no active applications running or lightly loaded with various tasks. As the platform’s energy in S0 is optimized with the new generation of platforms, any help provided by peripheral devices to optimize its energy consumption will benefit the end-users with longer battery life

References

[1] “ACPI advanced configuration and power interface specifications rev 4.0,” http://www.ecma-international.org/, November 2009, [Online; accessed 08-June-2012].

[2] Z. Duan, K. Gopalan, and Y. Dong, “Push vs. pull: Implications of protocol design on controlling unwanted traffic,” in *SRUTI*, 2005, pp. 25–30.

[3] “Wake-On-Lan,” http://en.wikipedia.org/wiki/Wake-on-LAN, [Online; accessed 08-June-2012].

[4] N. Mishra, K. Chebrolu, B. Raman, and A. Pathak, “Wake-on-wlan,” in *international conference on World Wide Web (WWW)*, 2006, pp. 761–769.

[5] D. C. Plummer, "An Ethernet Address Resolution Protocol," http://tools.ietf.org/html/rfc826, November 1982, [Online; accessed 08-June-2012].

[6] "Transmission control protocol, protocol specification," http://www.ietf.org/rfc/rfc793.txt, September 1981, [Online; accessed 08-June-2012].

[7] M. Crispin, "Internet message access protocol," http://tools.ietf.org/html/rfc3501, March 2003, [Online; accessed 08-June-2012].

[8] J. Henrys, "The compute continuum: A wave of connected devices," http://www.internetviz-newsletters.com/eletra/mod_print_view.cfm?this_id=1940237&u=intel&show_issue_date=F&issue_id=000481299&lid=b11&uid=0, [Online; accessed 08-June-2012].

[9] "ENERGY STAR: program requirements for computers," http://www.energystar.gov/ia/partners/prod_development/revisions/downloads/computer/Version5.0_Computer_Spec.pdf, [Online; accessed 08-June-2012].

[10] Y. Agarwal, T. Pering, R. Want, and R. Gupta, "SwitchR: Reducing System Power Consumption in Multi-Clients Multi Radio Environment," in *IEEE International Symposium on Wearable Computers (ISWC)*, 2008, pp. 99–102.

[11] J. Flinn and M. Satyanarayanan, "Managing Battery Lifetime with Energy Aware Adaptation," in *ACM Transactions on Computer Systems*, vol. 22, 2004, pp. 137–179.

[12] X. Li, R. Gupta, S. Adve, and Y. Zhou, "Cross Component Energy Management: Joint Adaptation of Processor and Memory," in *ACM Transactions on Architure and Code Optimization*, vol. 4, no. 14, 2007.

[13] C. Gunaratne, K. Christensen, and B. Nordman, "Managing Energy Consumption Costs in Desktop PCs and LAN Switching with Proxying, Split TCP Connections and Scaling of Link Speed," in *International Journal of Network Management*, 2005, pp. 297–310.

[14] M. Gupta and S. Singh, "Greening of the Internet," in *ACM Sigcomm*, 2003, pp. 19–26.

[15] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing Network Energy Consumption via Sleeping and Rate Adaptation," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008, pp. 323–336.

[16] Y. Agarawal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009, pp. 365–380.

[17] E. Shih, P. Bahl, and M. Sinclair, "Wake on wireless: An event driven energy saving strategy for battery operated devices," in *IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2002, pp. 160–171.

[18] J. Sorber, N. Banerjee, M. Corner, and S. Rollins, "Turducken: Hierarchical Power Management for Mobile Devices," in *IEEE International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2005, pp. 261–274.

[19] "proxZZZy for sleeping hosts," http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-393.pdf, February 2010, [Online; accessed 08-June-2012].

[20] C. Olsen and C. Narayanaswami, "PowerNap: An Efficient Power Management Scheme for Mobile Devices," in *IEEE Transactions on Mobile Computing*, 2006, pp. 816–828.

[21] V. Yodaiken and M. Barabanov, "A Real-Time Linux," in *Linux Journal*, vol. 34, 1997.

[22] T. Gleixner and I. Molnar, "Dynamic Ticks," http://lwn.net/Articles/202319/, October 2006, [Online; accessed 08-June-2012].

[23] K. Egevang, C. Communications, and P. Francis, "The IP Network Address Translator (NAT)," http://www.ietf.org/rfc/rfc1631.txt, [Online; accessed 08-June-2012].

[24] "Apple push notification service," https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html, [Online; accessed 08-June-2012].

[25] "Intel WiFi Link 5100 Series specifications," www.intel.com, 2008.

[26] S. H. Maes, C. Kuang, R. Lima, R. Cromwell, E. Chiu, J. Day, R. Ahad, W.-H. Jeong, and G. Rosell, "Push extensions to the imap protocol (P-IMAP)," http://tools.ietf.org/html/draft-maes-lemonade-p-imap-12, March 2006, [Online; accessed 08-June-2012].