Transport Protocol Optimization For Energy Efficient Wireless Embedded Systems

Davide Bertozzi†, Anand Raghunathan‡, Luca Benini† and Srivaths Ravi‡
† University of Bologna, DEIS, Viale Risorgimento,2 Bologna
‡C & C Research Labs, NEC USA, Princeton, NJ 08540
‡Alphion Corp., Eatontown, NJ 07724
{dbertozzi@deis.unibo.it, anand@nec-lab.com,
lbenini@deis.unibo.it, sravi@nec-lab.com}

Abstract— **For wireless embedded systems, the power consumption in the network interface (radio) plays a dominant role in determining battery life. In this paper, we explore transport protocol optimizations for reducing the energy consumption of wireless LAN interfaces. Our work is based on the observation that, the transport protocol, which implements flow control to regulate the network traffic, plays a significant role in determining the workload of the network interface. Hence, by monitoring run-time parameters in the transport protocol, coarse-granularity idle periods, which present the best opportunities for network interface power reduction, can be accurately identified. We further show that, by tuning parameters in the protocol software implementation, we can shape the activity profile of the network interface, making it more energy efficient while remaining compliant to the TCP standard. We have performed extensive current measurements using an experimental testbed that consists of a Compaq iPAQ PDA with a Cisco Aironet wireless network adapter, to validate the proposed techniques. Our measurements indicate energy savings ranging from 28% to 69% compared to the use of state-of-the-art MAC layer power reduction techniques, with little or no impact on performance.**

I. Introduction

The widespread use of battery-driven embedded systems with wireless communication capabilities (*e.g.,* PDAs, palmtop computers, cell phones, networked sensors, *etc.*) has generated significant research interest in energy efficient design techniques for such systems. Researchers have shown that, in most wireless embedded systems, the communication subsystem (the network interface card or radio), accounts for a large portion of the system's total energy consumption [1], [2]. Hence, energy efficiency needs to be addressed in the network protocols, in addition to the design of the hardware and software that implement them. In this work, we focus on the optimization of the transport protocol for energy efficiency, by adapting it to judiciously regulate the operation of the network interface.

The significance of network interface power consumption has led to the development of network interface cards with the ability to work in different power modes, *e.g.,* receive mode, transmit mode, idle mode and sleep mode, exploiting the principles of power management that are well-known in the hardware and system design domains [3]. The challenge in developing an effective power management strategy is to identify the times during which the sub-system can be placed in a particular power mode, since switching between power modes incurs significant latencies and power overheads [1]. Network protocols, therefore, must efficiently exploit the available power modes, by regulating the operation of the communication sub-system based on the application requirements, channel characteristics, and network conditions.

[1]In the context of communication sub-systems, it is worth mentioning that these overheads are even higher than in computation sub-systems, since the wireless client may need to re-associate with the network for continued service. For example, the wake-up time of a Cisco Aironet wireless LAN card is almost twice the wake-up time for an Intel StrongARM SA-1110 processor.

Developing an effective energy reduction strategy for network interfaces requires us to examine their workload, which is determined by the network protocols that regulate the overall traffic flow. Knowledge that is commonly available at each layer of the protocol stack can be used to identify periods in which the network interface can be placed in each power mode, resulting in significantly higher energy savings than conventional approaches. Here, we note that a trade-off exists between awareness of the application-generated workload, and knowledge of channel conditions. The higher layers of the protocol stack have full control of the workload of the network adapter, but as we move down the protocol stack, the availability of more refined information about the channel status (received signal strength, number of corrupted packets, *etc.*) comes at the cost of greater uncertainty about how the channel is going to be used in the long term. Most work on low power protocols has been at the medium access control (MAC) sub-layer. While the MAC protocol is directly exposed to channel variations, it has relatively little application visibility. For example, according to the IEEE 802.11 standard protocol for wireless local area networks (LANs), the network interface card can transition into a low power sleep mode. However, the card has to be periodically awakened to check whether outstanding packets for the client are buffered at the access point [4], independent of whether such packets are actually present, potentially leading to unnecessary and significant energy overheads.

In this paper, we consider transport protocol optimizations to increase energy efficiency, specifically focusing on the popular Internet protocol TCP. Power saving mechanisms at the transport layer have the advantage of being application-independent, and as most network applications run on top of TCP, the optimization of its energy efficiency allows a large range of applications to make an energy-aware use of the network interface. Note that, modifying the applications themselves to be power-efficient is another alternative, albeit much less applicable, since such an approach would require modifications of widely used legacy applications (telnet and FTP clients, web browsers, streaming media players, *etc.*).

Based on an extensive analysis of TCP buffering and flow control mechanisms, we have observed that energy saving measures targeting a specific traffic profile can be triggered easily by automatically monitoring various statistics related to the TCP send and receive buffers. Further, we show that tuning knobs provided in the protocol's software implementation allows us to regulate the activity profile of the network interface, making it more suitable for energy reduction. We exploit in-built signaling mechanisms of TCP, such as window advertisement and silly window syndrome (SWS) avoidance [5], to identify scenarios where low power modes can be triggered with minimal overheads. We study the impact of varying relevant TCP parameters on energy efficiency and performance. These tradeoffs are exploited to increase the energy savings obtained, with reduced performance penalty, while remaining compliant to the TCP standard.

The energy savings provided by our technique range from 28% to 69% compared to the use of state-of-the-art MAC layer power management policies, with very little performance degradation. All results come from actual current measurements performed on an experimental testbed that consists of a Compaq iPAQ H3800 PDA running the Linux OS [6], connected to a wireless LAN through a Cisco Aironet 350 adapter.

The rest of this paper is structured as follows. Section II discusses previous work in the area of low power protocols. Section III discusses the proposed transport protocol energy optimization strategy. Section IV details the experiments performed to validate the proposed techniques, and discusses the results and tradeoffs obtained.

II. RELATED WORK

Protocol optimizations for low power embedded systems have been proposed for different layers of the communication protocol stack [7]. At the physical layer, energy efficiency can be achieved by choosing appropriate modulation, coding, and transmission power control schemes, based on the channel characteristics [8], [9]. Several options have also been presented for energy efficient media access control (MAC) protocol design [10]. At the network layer, attempts have been made to develop low power, scalable, and fault-tolerant routing algorithms, and to enable longer network survivability [11], [12].

As we move to the upper (transport and higher) layers of the protocol stack, innovative protocols can be designed to exploit application-specific information and reduce power usage [13]. Using application-specific information, it becomes easier to identify or predict periods of zero workload (idle periods) with respect to the channel utilization in order to shut down the network interface. These idle periods can be forced by traffic shaping techniques [14] or application-level buffering policies [15], or predicted [16].

Our work targets the transport layer of the protocol stack, and deals with the energy efficiency of TCP, the reliable end-to-end communication protocol. Most of the optimizations reported in the literature attempt to enhance TCP for performance [17], [18]. In particular, TCP exhibits very poor performance in a wireless scenario, because it implicitly assumes that all losses are due to congestion and reduces the sender window size accordingly. Since such a policy ignores the effects of the error-prone wireless channel, it may inefficiently reduce throughput in the presence of non-congestion-related errors [19]. Techniques such as Indirect-TCP [20] have been proposed to deal with this problem [17].

The use of TCP on battery-driven systems has led researchers to investigate further optimizations for energy efficiency. An early study on this topic [21] showed that the energy characteristics of TCP significantly depend on the error correlations and, unlike throughput, on the particular TCP implementation (*e.g.*, Tahoe, Reno, NewReno, Vegas). Since then, a number of suggestions have been documented to improve the energy efficiency of TCP [14], which include:

- The power-constrained side of a TCP connection (the wireless terminal) can be made simpler, by transferring functionality such as timers and state to the other side [22]. This optimization targets the processor energy expended in executing the TCP implementation.
- TCP can be made aware of non-congestion-related losses, improving both performance and energy. Techniques used for this purpose are local re-transmissions, split connections, and additional forward error correction. In this work, we assume that such well-known techniques are already implemented, and investigate further optimizations for energy reduction.
- TCP can also be used to make the sender transmit in a predictable manner [23]. By making the sender transmit data in bursts with sufficient separation to one another, the receiver is provided the opportunity to sleep in the idle periods.

Tuning TCP's parameters is the most straightforward way to improve its efficiency. The default parameters of TCP have been consciously designed to sacrifice throughput, in exchange for fair sharing of bandwidth on congested networks. One of the most critical parameters is certainly the TCP buffer size, and techniques for determining the optimal size for performance are reported in [24]. The TCP buffer size can also be dynamically adjusted to the connection and server characteristics [25], [26].

The goal of this work is to show that the TCP buffering mechanisms can be exploited to significantly increase energy efficiency of the transport layer with minimum performance overheads. The energy savings obtained are over and above those possible using MAC layer techniques, and the proposed techniques can result in beneficial energy *vs.* performance trade-offs.

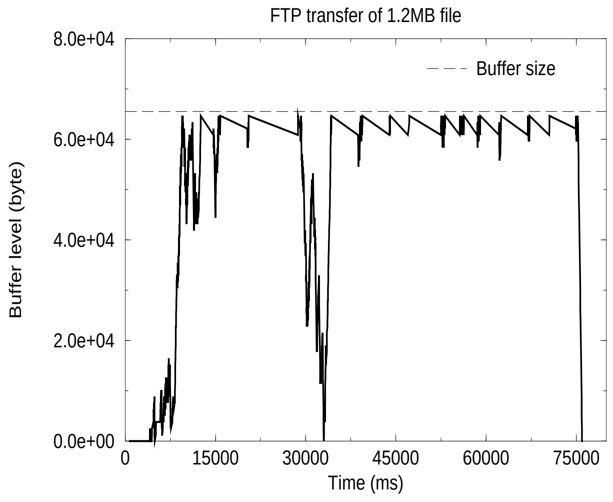

Fig. 1. Buffer occupancy for an FTP download to FLASH memory

III. LOW-POWER TCP BUFFERING

We target the receiver side of a TCP connection, and make the assumption of a single application scenario for the ease of explanation. By monitoring the TCP receive buffer occupancy or utilization, we want to identify periods of inactivity so that the network interface card can be sent into a low power mode. Unlike lower-layer (*e.g.*, MAC layer) techniques, we are looking for coarse-grained opportunities, where the network interface does not need to periodically wake up to synchronize with the access point or base station.

The idea to exploit TCP-generated idle periods for network interface shutdown is already present in [14]. However, the authors claim they get idle periods of the order of the round trip time (RTT), but in real scenarios this idle period is often too small to make the network adapter switch to sleep mode. For example, with a Cisco Aironet 350 card, we noticed a wake-up time of about 300 ms. This is mainly due to re-association with the LAN, which is quite time consuming. Such an overhead prevents us from shutting down the card when the available sleep time is less than 300 ms (even if we neglect the time overhead to switch off the card), and the RTT is very often not that high.

Our approach is complementary, since we identify or somehow generate TCP idle periods with a much higher granularity than the RTT. In particular, there are two relevant cases wherein TCP experiences coarse-grained inactivity: when the buffer is *full* and when the buffer is *empty*. It bears mentioning that the TCP optimizations that target these two conditions are fairly orthogonal, and complement each other in the sense that they are typically useful for applications with different characteristics (bulk download oriented *vs.* interactive).

A. TCP receive buffer full

To demonstrate the TCP buffer full condition, we report in Fig. 1 the TCP receive buffer occupancy at the client, during an FTP download to the FLASH memory of an iPAQ handheld device. The iPAQ runs the Linux (Familiar Distribution) operating system, which contains an implementation of TCP that we subsequently refer to as linuxTCP.

Writing a FLASH memory is a slow operation, mainly because of the cell erasure overhead: the Linux garbage collector thread, that is in charge of this task, on average takes 70% of the FTP transfer execution time. This explains why the average buffer occupancy is very high: the application is very slow in reading data out of the TCP receive buffer with respect to the available network bandwidth, and this causes the buffer to be frequently full. The high average buffer occupancy during the socket lifetime is typical of download oriented applications such as FTP to a FLASH file system, or TCP-based streaming multimedia [27].

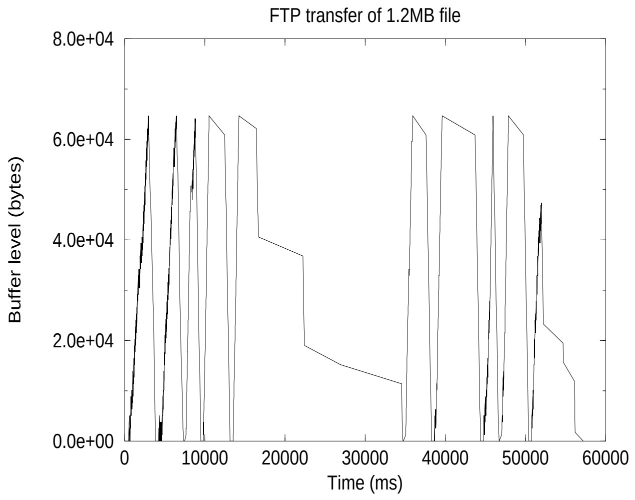

Fig. 2. Buffer occupancy profile for FTP with our low-power TCP implementation

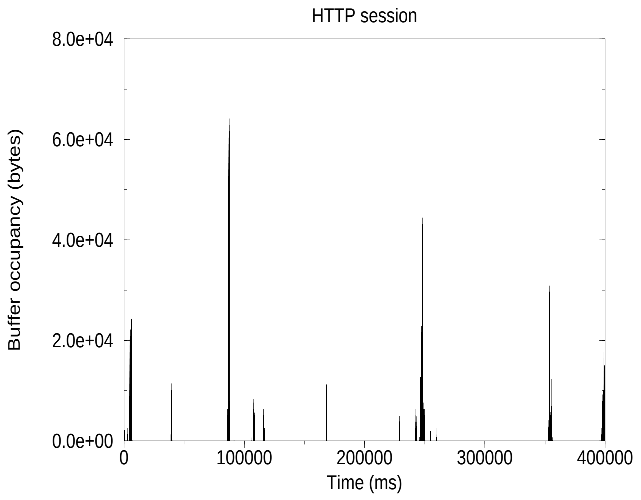

Fig. 3. Buffer occupancy profile for an HTTP session

It is important to note that TCP provides a mechanism that prevents the receive buffer from overflowing. The receiver advertises a *receive window* by piggy-backing this information onto acknowledgment (ACK) packets it transmits to the sender. The sender also estimates the amount of data that can "live" in the network. The number of bytes transmitted by the sender against receipt of an ACK is computed as

$$txbyte = min(rxwnd, cwnd);$$

where $rxwnd$ is the receive window advertised by the receiver (indicating the buffer space availability), and $cwnd$ is the contention window (the sender's estimation of the amount of data that the network can sustain without causing congestion).

When the receive buffer is full, the receiver side of a TCP connection advertises a *zero window*, thus preventing further transmissions at the sender side. As the application reads data out of the receiver buffer, the window becomes non-zero, but it is not immediately advertised to avoid the "silly window syndrome" (SWS) [5]. In fact, the receiver has to wait until its window has considerably increased, to prevent the inefficient exchange of small amounts of data across the connection (instead of full-sized segments). The normal algorithm is for the receiver to advertise a larger window only when the window can be increased by either one full-sized segment or by one-half the receiver's buffer space, whichever is smaller [5].

We propose to exploit this built-in client-driven flow control mechanism for energy reduction: the network interface card can be *completely* switched off between the zero and non-zero window advertisements, as TCP itself ensures that the server will not transmit any packets during that time. Further, we view the buffer occupancy threshold when a non-zero window is advertised (immediately following a zero window) as a *tunable parameter*, and consider the default value adopted by linuxTCP as an upper bound. The lower bound is zero buffer occupancy, *i.e.*, the case wherein the card is recovered from sleep state only when the buffer is completely drained by the application.

By using the aforementioned aggressive approach, we get the new profile for buffer occupancy that is shown in Fig. 2. We clearly see the effect of shutting down the network adapter when the buffer is full: the buffer is progressively emptied at a rate which is affected by the application speed (*e.g.*, processing overhead) and operating system delays (*e.g.*, scheduling). On the other hand, buffer fill-up speed depends on the relative ratio between network bandwidth and application speed. On an average, the sleeping period between the buffer full and buffer empty condition is of the order of seconds, which explains the significant energy savings that will be documented later in this paper. Note that, since we exploit higher-layer protocol knowledge to avoid periodically waking up to contact the access point, this strategy achieves energy savings that are higher than conventional MAC layer techniques.

B. TCP receive buffer empty

Another idleness condition that can be exploited at the transport layer occurs when the buffer is empty. Generally speaking, this condition might be due to complete network inactivity (no open sockets), or due to temporarily un-utilized sockets.

Power management actions have to be taken very carefully when the TCP buffer is empty, because, even though no packets are presently stored, they might be in transit over the network. Therefore, care has to be taken to ensure that incoming packets are not dropped. This can be ensured in two ways:

- The traffic can be regulated by using TCP control flow mechanisms (*e.g.*, by artificially advertising a zero window). This is a high-impact approach, that can be employed whenever the vanilla protocol does not exhibit good characteristics for energy optimization (*e.g.*, long idle periods).
- The traffic flow can be predicted considering the status of the outstanding TCP sockets. This approach is likely to have less impact on the original TCP traffic, and can be used whenever the traffic profile generated by an application can be directly exploited for energy minimization with minimum performance overhead. Since TCP has built-in mechanisms for re-transmission, any dropped packets will be automatically recovered, as long as they are re-transmitted after the receiver's network interface is woken up.

For the buffer empty condition, we propose to take the second approach, based on traffic prediction. In fact, many applications display a traffic profile consisting of bursts of network transfers followed by long idle periods. This is often the case in interactive applications, such as chat and web browsing. For example, in web browsing, since users spend time reading a web page before clicking on a link to generate a new HTTP request, web-pages are likely to be downloaded onto the client terminal largely inter-spaced in time (see for example Fig. 3).

The idleness we suggest to exploit to shut down the network adapter is not the one generated by network congestion or wireless channel conditions, since they are often fine-grained and relatively difficult to predict. Rather, we consider the more promising scenario that occurs when the buffer is empty because no socket is actively using the network. In other words, no packets are expected at the receiver side. This situation is potentially detectable by TCP, and can be exploited to switch off the card, provided we can restore it to seamlessly allow subsequent network operations. The duration of time between one network "transaction" and the next one determines the amount of energy reduction. A timeout mechanism can be efficiently employed in this context. Therefore, the linuxTCP implementation was changed to support a new low power feature related to the buffer empty condition, as described next.

B.1 Implementation

We activate a timer whenever the TCP receive buffer becomes empty. This condition is checked after each transport layer read instruction is completed, and when an ACK is received in response to a prior transmission (both of these operations free up buffer space, and can potentially result in an empty buffer). Subsequent read or write operations can reset and re-activate the timer if they occur before it expires. Otherwise, a timer handler is called that switches off the network interface card, because we predict that the card is not likely to be used for a long time.

In further detail, we exploit the linuxTCP $tcp\_recvmsg$ function, that transfers data from the TCP buffer to the user buffer, and is called by the application-level read instruction on a certain socket. The length of data to be read is passed to this transport layer routine as a parameter. For non-blocking sockets, the execution flow does not get out of $tcp\_recvmsg$ until all data has been read. When this happens, received packets are acknowledged and deleted from the receive buffer. At this point, we carry out a buffer occupancy check, and activate the card shutdown timer if the buffer is empty. The same check is performed after an ACK is received [2].

The card is re-activated when a new TCP packet transmission or read operation from the TCP buffer has to be carried out. The timeout granularity should be much higher than the granularity of time periods between network transactions so to efficiently detect exploitable idle periods. Should the timeout expire between two successive buffer read operations, for example, the second one would be delayed by the card's wake-up time.

As shown later, the policy described above can provide very high energy savings for applications that generate bursty traffic. The introduced overhead, *i.e.*, the latency associated with card switching, must not seriously affect user-perceived performance. Another network-related overhead may occur when, after a card shutdown, new packets try to reach the receive buffer before the TCP read operation is scheduled: in this case, they cannot be received because the interface is off, and would be re-transmitted by the server at the expiration of the retransmission timeout. However, this would only affect the first few packets in each burst. Overall, these factors lead to very minimal overheads.

IV. EXPERIMENTAL RESULTS

The experimental testbed used to validate the proposed protocol optimization policies consists of a Compaq iPAQ H3800 PDA, provided with network connectivity by means of a Cisco Aironet 350 wireless LAN adapter. A PC card extender from Sycard Inc. is used to measure the current drawn by the network interface: a 1 ohm resistor is placed in series with the card's power supply, and a data acquisition board samples the current flowing through the resistor, allowing software processing by means of Labview from National Instruments Inc. The Linux Familiar Distribution (Linux 2.4) [6] runs on the iPAQ.

We compare the energy and performance of the proposed optimized TCP (lpTCP) against the TCP implementation in the Linux distribution (linuxTCP). To compare the utility of the proposed techniques in the presence of MAC layer power reduction, the experiments have been performed using the three in-built MAC layer power management policies of the Cisco Aironet network interface card:

- CAM (Constantly awake Mode): The client adapter is kept continuously powered up, so as to have little lag in the message response time. This approach consumes the most power but offers the highest throughput.
- PSP (Power Save Polling): The access point buffers incoming messages for the client adapter, which wakes up periodically to retrieve them. The energy consumption is minimized at the cost of low throughput. This approach is recommended for battery-constrained devices.
- PSPCAM: The card switches between PSP and CAM modes, depending on the network traffic. This mode switches to CAM when retrieving a large number of packets, and switches back to PSP after the packets have been retrieved. This is an intermediate approach between CAM and PSP, from the energy-performance trade-off viewpoint.

[2]We cannot trigger the timer immediately after a packet transmission, because TCP is waiting for the corresponding ACK.

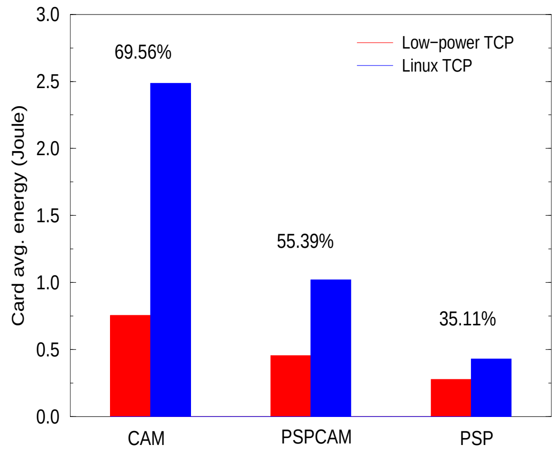

Fig. 4. Energy savings of the proposed lpTCP compared to Linux TCP

A. Card shutdown at buffer full

We implemented the dynamic card shutdown policy, triggered when the TCP receive buffer is full. The silly window syndrome avoidance mechanism in linuxTCP was changed, so that a non-zero window is advertised when the buffer has been emptied up to a certain (wake-up) threshold. This threshold can be considered a tunable parameter of the lpTCP implementation. Between the zero and non-zero window advertisements, when the sender is not supposed to transmit packets, the network adapter is completely shut down.

We execute a FTP transfer of a 600 kByte file from a wired server on the Ethernet LAN onto the wireless client, which is connected to the network through a wireless access point. We repeated the download 25 times (to eliminate spurious network and OS related variations), and present the average energy and execution time measurements to evaluate the performance of our technique.

A.1 Energy savings from buffer full: lpTCP vs. linuxTCP

Fig. 4 shows a comparison between the energy consumed by the proposed lpTCP implementation (with the wake-up threshold set to one fourth of the buffer size), with respect to the original linuxTCP implementation. As expected, in CAM mode, lpTCP achieves the highest energy savings (almost 70%), because we put the card to sleep for periods of the order of seconds instead of keeping it in idle mode. The large granularity idle periods are obtained by tuning the card wake-up threshold, and hence by regulating the TCP traffic, to best exploit the low power features of the card.

In PSPCAM mode, our technique still provides energy savings as large as 55%. During periods of inactivity, PSPCAM switches the card to sleep mode, but wakes it up periodically to have buffered packets at the access point forwarded. Our technique relies on the fact that this synchronization overhead is not necessary given the client-controlled nature of the SWS mechanism: while the TCP receive buffer is being emptied, no packets are expected to be buffered at the access point.

Finally, our approach exhibits 35% energy savings when PSP is activated. This case leads to the least energy savings for our techniques, since the effective network throughput is reduced, and the buffer full condition occurs fewer times than in the case of CAM or PSPCAM. Moreover, PSP itself spends a lot of time in sleep state, resulting in an already low energy dissipation. However, even in PSP mode, lpTCP demonstrates a sizeable energy improvement, due to the better exploitation of coarse-grained idle periods when there is no need to poll the access point for buffered packets (since the server has been prevented from transmitting).

A.2 Impact of wake-up threshold

Fig. 5 plots the average energy consumed by the network adapter as a function of the card wake-up threshold (under the default buffer size of

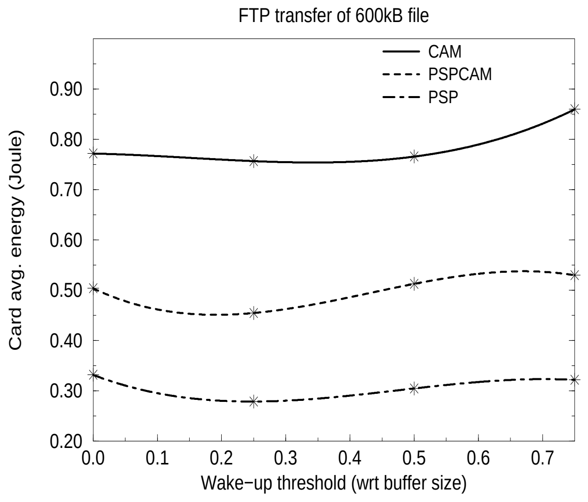

Fig. 5. Energy consumed by the network adapter for a FTP transfer as a function of the wake-up threshold

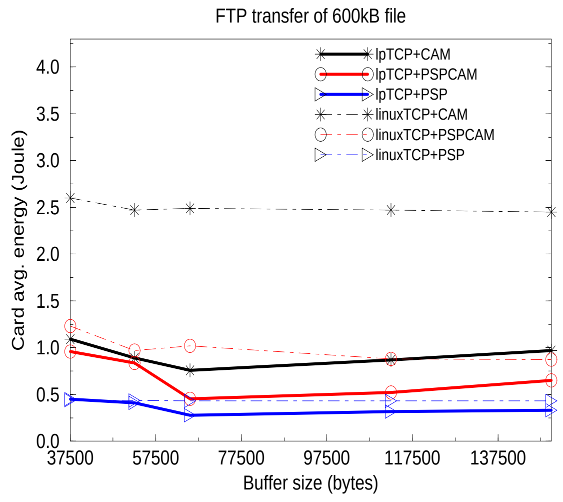

Fig. 6. Dependence of energy on TCP receive buffer size

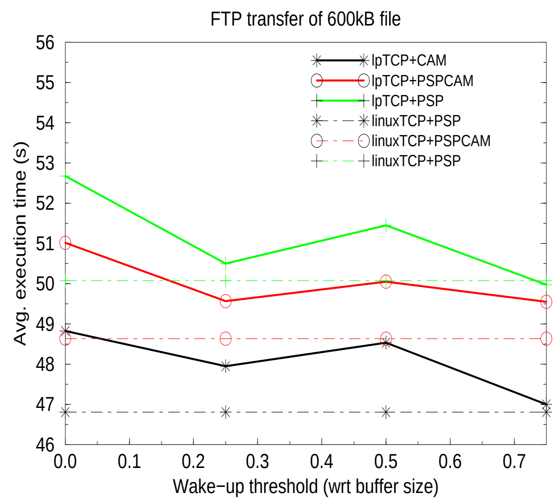

Fig. 7. Impact of our technique on performance, as a function of the card wake-up threshold

64 kBytes). From Fig. 5 we observe that a zero wake-up threshold is not the most efficient solution, because in this case, the network interface is restored only when the buffer is drained. At that time, the application expects to read more data from the buffer, but the reading process has to be suspended by the OS while the card is restored, an ACK with a non zero-window is sent to the server, and finally new packets are received and delivered to the application. For most of this period, the card is active, and this partially depletes the energy savings obtained by switching off the card.

On the other hand, if the threshold is set very high, a large number of power mode transitions take place at the network adapter, and the available sleep periods are very small (tens of milliseconds). This explains the increase in energy dissipation. The optimal threshold for the considered application lies in between, where some buffer-empty overheads can be tolerated in order to have sleeping periods long enough to shut down the card. This minimum energy point shifts to the left as we move from CAM to more aggressive MAC layer power reduction. This is due to the fact that PSP and PSPCAM reduce the throughput, partially hiding the application-network speed gap that causes the buffer full condition. This means that the buffer full condition occurs fewer times, and the time for which the card can sleep decreases as well.

A.3 Impact of TCP buffer size

We investigated the energy trade-off as a function of the TCP receive buffer size. This parameter, whose default value is 64 kBytes, was statically varied from 37,500 Bytes to 150,000 Bytes.

Fig. 6 shows the energy consumed by the same 600 kByte FTP transfer as a function of the buffer size, again comparing our implementation (lpTCP) against the original Linux TCP (linuxTCP). The wake-up threshold for lpTCP was fixed at one-fourth of the buffer size. Energy consumed by linuxTCP is almost constant except for the left-most part of the plot, where the application process is frequently suspended by the OS on a buffer empty condition, resulting in a slight energy increase.

The behavior of lpTCP is similar to Fig. 5, because increasing the buffer size causes the absolute value of the wake-up threshold to be proportionally increased, and since a larger buffer implies that the probability of the buffer full condition decreases. Again, for small buffer sizes, we notice an increase in energy due to frequent power mode switches of the interface card and smaller sleeping periods. Note that the PSPCAM curve becomes closer to CAM, because in these conditions, the communication takes place through a large number of small transfers, and this effectively prevents the card from transitioning to sleep state in the PSPCAM case.

If we observe the differences between linuxTCP and lpTCP for any given MAC layer policy, we note that our technique is more effective for medium sized buffers. At very large buffer sizes, we can expect lpTCP to behave similarly to linuxTCP, since the buffer full condition will never be activated. At very small buffer sizes, in theory, it is possible that lpTCP will actually incur energy overheads due to frequent power mode switching, very small sleep periods, and the overhead of application wait during buffer empty. In practice, however, we did not observe this for reasonable values of the buffer size. In Fig. 6, at the minimum buffer size of 37,500 Bytes, lpTCP with PSP breaks even with linuxTCP with PSP.

A.4 Performance Impact

The impact of the proposed lpTCP implementation on performance is reported in Fig. 7 [3]. For each MAC layer power management mode, the average execution time for the same FTP transfer is reported, for both linuxTCP and lpTCP. For high wake-up thresholds, the execution times tend to converge, because the card activation overhead is hidden by the high value of the TCP buffer occupancy when the threshold is crossed. While the card is coming up, the application can still rely on a large amount of buffered data to read. On the contrary, as the wake-up threshold becomes smaller, the remaining buffered data might not be sufficient (depending on the application consumption rate and the network bandwidth at card wake-up), and the application might end up waiting for new packet arrivals. In all cases, the delay overhead

[3] Note that the y-axis range has been zoomed to make the small performance variations clearly visible.

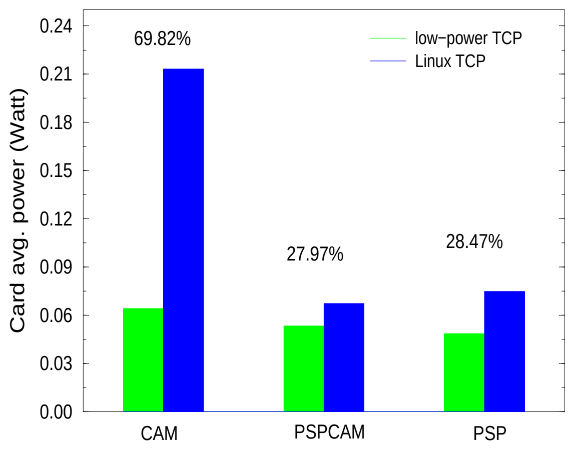

Fig. 8. Power consumption of lpTCP *vs.* linuxTCP for a HTTP session.

introduced by our energy-aware implementation is at most 5.3%, which occurs when lpTCP is used with PSP, and with zero wake-up threshold.

B. Card shutdown at buffer empty

We validated the energy optimization technique that exploits the buffer empty condition for a web browser session on the iPAQ. An open-source web browser, called Dillo [28], was used for our experiments.

We traced a web browsing session of 25 minutes, and repeated the access trace with lpTCP and linuxTCP as the underlying transport protocols to measure energy consumption. The sequence of web pages was chosen to contain a mix of text oriented as well as image oriented pages. lpTCP is expected to give maximum energy savings with text dominated content, where the card enters sleep state immediately after a quick download, as opposed to image dominated content, where the download operation is time-consuming and the user reads the text while the image transfer is being completed, leaving the card awake and active for longer periods. In the extreme case when the user starts a different transfer while the previous one is still in progress, the card does not sleep at all, resulting in identical behavior for lpTCP and linuxTCP.

The power consumption results for the network interface are reported in Fig. 8. The proposed lpTCP outperforms linuxTCP by almost 70% from an energy viewpoint in CAM mode. Fig. 8 also shows that with aggressive MAC layer power management enabled, the energy savings still amounts to 27-28%. Furthermore, there is not much difference between PSPCAM and PSP. This also holds for linuxTCP, and is a consequence of the bursty traffic, which keeps the card inactive for most of the time.

For these experiments, the timeout was fixed to 4s, but it could easily be tailored to the user or application, by exposing it as a tunable parameter, or by employing a dynamic learning technique within lpTCP itself. Finally, we noticed that the performance degradation represented by the card switching overhead is hardly perceivable by the user, similar to the performance results presented earlier.

V. CONCLUSIONS

The realization of energy efficient wireless embedded systems requires optimization of the network protocols and their implementation, in addition to the design of an optimized system architecture. In this work, we have developed standards-compliant techniques for optimizing transport layer protocols, so as to enable efficient energy reduction of the network interface. We have demonstrated the proposed concepts by implementing an energy efficient version of the Internet transport protocol, TCP. Power measurements on a PDA with a wireless LAN interface indicate that the proposed techniques can lead to significant energy savings compared to current TCP implementations. Further, these techniques are complementary to power reduction techniques implemented in lower-layer (*e.g.*, MAC) protocols. We believe that several popular applications (*e.g.*, file download, web browsing, *etc.*) employed on wireless handhelds can benefit from the proposed techniques.

REFERENCES

[1] M. Stemm and R. H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices," *IEICE Transactions on Communications*, vol. E80-B, no. 8, pp. 1125–31, 1997.
[2] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava, "Energy aware wireless microsensor networks," in *IEEE Signal Processing Magazine*, pp. 40–50, Mar. 2002.
[3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, 1997.
[4] *LAN MAN Standards Committee of the IEEE Computer Society*. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification: IEEE standard 802.11, 1999.
[5] *TCP/IP Illustrated Vol. 1* . Addison-Wesley, 1994.
[6] *The Familiar Handhelds Project*. `http://familiar.handhelds.org`.
[7] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J.-C. Chen, "A survey of energy efficient network protocols for wireless networks," *Wireless Networks*, vol. 7, no. 4, pp. 343–358, 2001.
[8] C. Schurgers, O. Aberthorne, and M. B. Srivastava, "Modulation scaling for energy aware communication systems," in *Proc. IEEE Symp. on Low Power Electronics and Design*, pp. 96–99, Aug. 2001.
[9] E. Cianca, M. Ruggieri, and R. Prasad, "Improving TCP/IP performance over CDMA wireless links: A physical layer approach ," in *Proc. IEEE Symp. on Personal, Indoor and Mobile Radio Comm.*, pp. 83–87, Sept. 2001.
[10] H. Woesner, J.-P. Ebert, M. Schlager, and A. Wolisz, "Power saving mechanisms in emerging standards for wireless LANs: The MAC level perspecitve," *IEEE Personal Communications*, vol. 5, pp. 40–48, June 1998.
[11] J.-H. Chang and L. Tassiulas, "Energy conserving routing in wireless adhoc networks," in *Proc. IEEE Infocom*, pp. 22–31, June 2000.
[12] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proc. Mobicom*, pp. 70–84, Sept. 2001.
[13] R. Kravets and P. Krishnan, "Application-driven power management for mobile communication," *Wireless Networks*, vol. 6, no. 4, pp. 263–277, 2000.
[14] S. A. Akella, R. K. Balan, and N. Bansal, "Protocols for Low-Power," tech. rep., Carnegie Mellon University, 2001.
[15] D. Bertozzi, L. Benini, and B. Ricco, "Power aware network interface management for streaming multimedia," in *Proc. IEEE Wireless Communications and Networking Conf.*, Mar. 2002.
[16] T. Simunic, L. Benini, P. W. Glynn, and G. D. Micheli, "Dynamic power management for portable systems," in *Proc. ACM Int. Conf. Mobile Computing and Networking*, pp. 11–19, 2000.
[17] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 6, pp. 756–769, 1997.
[18] T. V. Lakshman, U. Madhow, and B. Suter, "TCP/IP performance with random loss and bidirectional congestion," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, pp. 541–555, Oct. 2000.
[19] S. Avancha, V. Korolev, A. Joshi, and T. Finin, "Transport protocols in wireless networks," in *Proc. IEEE Intl. Conf. on Computer, Communication andNetworking*, Oct. 2001.
[20] A. V. Bakre and B. R.Badrinath, "Implementation and performance evaluation of Indirect TCP," *IEEE Transactions on Computers*, vol. 46, pp. 260–278, Mar. 1997.
[21] M. Zorzi and R. Rao, "Is TCP Energy Efficient?," in *Proc. IEEE Intl. Wkshp. on Mobile Multimedia Communications*, Nov. 1999.
[22] Z. J. Haas and P. Agrawal, "Mobile-TCP: An asymmetric transport protocol design for mobile systems," in *Proc. Intl. Conf. on Communications*, pp. 1054–1058, 1997.
[23] S. Chandra and A. Vahdat, "Application-specific network management for energy-aware streaming of popular multimedia formats," in *Proc. USENIX Annual Technical Conference*, June 2002.
[24] "TCP Tuning Guide for Distributed Applications on Wide Area Networks," in *USENIX and SAGE Login*, `http://www-didc.lbl.gov/tcp-wan.html`, February 2001.
[25] E. Weigle and W. Feng, "Dynamic right-sizing: A simulation study," in *Proc. IEEE Intl. Conf. on Computer, Communication and Networking*, Oct. 2001.
[26] A. Cohen and R. Cohen, "A dynamic approach for efficient TCP buffer allocation," *IEEE Transactions on Computers*, vol. 51, pp. 303–312, Mar. 2002.
[27] J. S. Chase, A. J. Gallatin, and K. G. Yocum, "End system optimizations for high-speed TCP," in *IEEE Communications Magazine*, pp. 68–74, Apr. 2001.
[28] *The Dillo Web Browser Project*. `http://dillo.cipsga.org.br`.