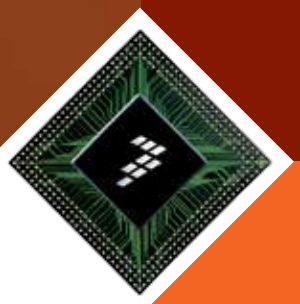


White Paper

Low-Power Network Standby in the Home and Office

Ben Eckermann
Digital Networking
Freescale Semiconductor

freescale.com

Abstract

Offices have long been networked, but the home is becoming an increasingly electronic and networked place. As the public demands (and governments mandate) an energy-efficient home, and everything from TVs to printers and refrigerators becomes networked, the energy consumption of these devices becomes critical. Most of these networked edge devices spend the majority of their time idle, but still need to remain present on the network. This creates the need for a new energy conservation technique—minimizing power while network-connected and idle—but the devices still need to be ready to respond at a moment's notice.

This paper analyzes the nature of embedded computing systems, reviewing system-level power optimizations to minimize network standby power. Then, in the context of low-power embedded systems in the home and office, we investigate and propose three ways to handle the network traffic at the heart of a low-power network standby system—packet classification, packet accumulation and autorespond proxies.

1. Introduction

Growing energy demands from embedded electronics and increasing evidence of dramatic global climate change are generating greater environmental and cultural pressure for energy-efficient solutions in embedded computing applications. This, coupled with continued expectations for higher performance embedded computing with each new product generation, despite environmental concerns, is impacting the future usage models of embedded processing systems.

Although traditional offline powered equipment, such as appliances, HVAC and lighting systems, dominate electric equipment energy consumption, embedded electronics and online equipment, such as printers, storage, networking infrastructure and data centers, are increasingly consuming a larger share of our energy resources. Furthermore, even equipment that was traditionally offline, such as TVs, refrigerators and HVAC controls, are now going online, while containing even more embedded processing.

To balance the performance required for powerful new electronic applications with rising concerns over energy consumption, environmentally aware "green" movements and government regulations and programs are driving manufacturers to develop intelligent strategies for optimizing performance within specific energy budgets.

Traditional embedded computing platforms have been designed for maximum work load with little regard to the cyclical work profile across hourly, daily, weekly or extended time intervals. However, new-generation high-performance systems are shifting their design focus from provisioning worst-case maximum power loads to optimizing for energy efficiency across varying workloads. Products such as printers are good examples of a cyclical workload, as they tend to spend much more time in a ready-to-print state or performing low-workload management services than they do for higher energy consumption printing states. Other embedded applications such as home network gateways, industrial processing plants and telecommunications systems can employ similar profiling to reduce energy waste and costs too.

As an example, office printers typically print cumulatively only one hour out of a 168-hour work week. Without system power management techniques, the 167 hours of idle time power cumulatively can exceed the active state power. Lowering power consumption requires advanced energy management schemes from new product development engineers. A simple strategy to design lower power consuming electronics begins to address the energy-efficient embedded computing challenge; however, larger gains will come from creating flexible systems that can pace workload with energy consumption in an intelligent and efficient manner.

Previous work, such as "Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems,"[1] "Long Idle: Making Idle Networks Quiet for Platform Energy-Efficiency"[2] and "Long Idle: Making Idle Networks Quiet for Platform Energy-Efficiency,"[3] has concentrated on analyzing and optimizing energy for PCs. However, networked embedded devices have a different set of requirements with more stringent power limits. Therefore, solutions that require additional hardware components may save significant power in a PC, but the same conclusion cannot necessarily be drawn in an embedded system where the power of the additional hardware components may exceed the power otherwise saved by their use. A potential example of this is the autorespond proxy described later in this paper.

Embedded devices also have other differing characteristics from PCs. Historically, although the gap is diminishing, processing levels are not as high for embedded devices as for PCs. Furthermore, other key features of the workload may impact the optimal low-power system solution—an embedded processor is less likely to have to perform scheduled virus scans or data backups, and an embedded processor may require a faster wakeup time from a sleep mode than a general-purpose PC. As such, this paper concentrates on low-power network standby as it relates to embedded systems in the home and office.

2. Understanding the Cyclical States of Embedded Computing Systems

In most cases, all the work performed in embedded computing applications is done in cycles—a combination of active states, management states and network standby states that are dynamically administered to most effectively optimize energy-efficient performance on demand. This is true for such applications as high-speed printing, home routers and WAN managed systems. The various network functions that the system performs in each state are outlined in figure 1.

In an embedded networked application, the system spends much of the time in a low-power network standby mode and wakes up in response to an external event. If the system takes too long to wake up, the window for acting on the event that caused the wakeup may have closed.

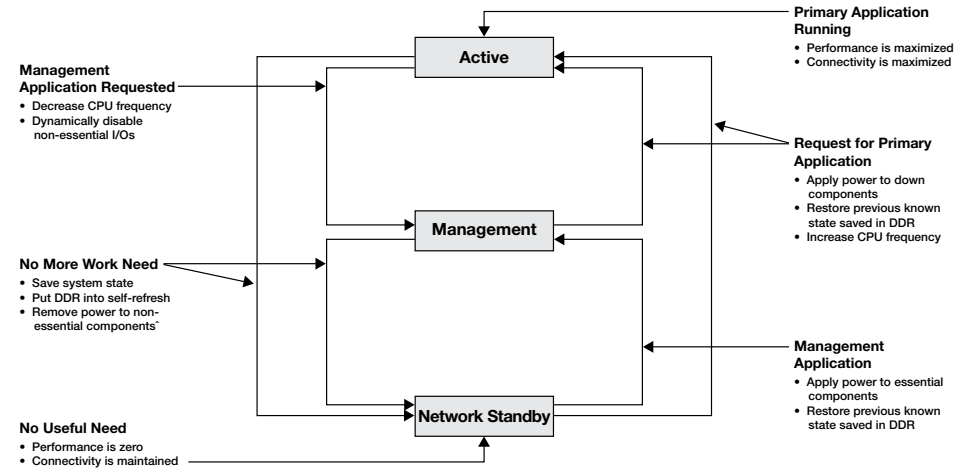
Further power reductions can potentially be achieved in non-active states. In wired networks, 10/100 MB Ethernet interfaces rather than Gigabit may be sufficient, saving significant Ethernet PHY power. Even bigger savings have been available since 2010, when the IEEE® ratified the Energy Efficient Ethernet (also known as 802.3az) standard[4]. This standard reduces power during periods of low use, and Ethernet PHYs that support it can reduce power by up to 70 percent when the PHY is not receiving or transmitting packets.

3. Packet Classification

Lossless packet operation in a networked environment is a method for ensuring that critical packets initiate the wakeup sequence and that no targeted packets are lost. A common example is wake-on-ARP (address resolution protocol[6]). An ARP packet can find a host's hardware address when given its network layer address. When the system receives an ARP packet that is destined for it alone, it triggers a wakeup to respond per protocol specifications.

Computer networks in the home and office typically have traffic 24 hours a day, even when no one is in the home or office, and even when the building is not occupied. This "idle" network traffic is analyzed in

Figure 1: System States



papers such as "Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems,"[1] "Long Idle: Making Idle Networks Quiet for Platform Energy-Efficiency"[2]. Note that even though such papers analyze traffic in the context of PCs, this is still relevant for embedded networked devices in the home or office, which generally coexist on networks shared with PCs and thus are subject to the same classes of incoming traffic.

"Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems,"[1] shows that in the systems they measured, the primary source of broadcast traffic in both the office and home is in fact ARP, and that while office networks may have significant multi-cast router traffic, such as the Hot Standby Router Protocol (HSRP) or Protocol Independent Multicast (PIM), these multicast packets in general have the potential to be safely ignored.

Further optimization is possible with the realization that many packets that arrive on the network are destined for other devices on the network, not the embedded processor in question. They can therefore be safely ignored, as can router multicast traffic. If the controlling processor wakes up on every packet, it will be constantly awake and have no time to enter a low power mode. Nevertheless, there are times when the processor needs to wake up to service and process certain packets. This is similar to the concept of "proxy_1" in "Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems."[1]

Embedded processors need to be able to wake on any targeted network event. For example, in a networked system with what is hereby termed packet classification, an embedded processor can enter the network standby mode where the system is dormant, yet its network controller still operates and ignores no packets. At the same time, if desired, DDR can be in self-refresh mode, still accessible if needed to buffer packets, while potentially also storing the system state for fast wakeup.

For true packet classification, the network controller needs to have some additional capability—in particular a receive filer to inspect and classify incoming packets. The receive filer can be configured to drop packets that don't need to be processed. Packets that need processing, such as ARP packets destined for the correct address, are written to DDR, and the network controller wakes the system from deep sleep for processing. For flexibility, the receive filer should be programmed to accept and wake on whatever packets are interesting for a particular system usage configuration. An example of a product performing packet classification is the Freescale MPC8536E communications processor[7], which implements packet classification in its deep sleep mode through the enhanced triple-speed Ethernet controller (eTSEC).

The goal is to achieve the best of both worlds—operating at ultra low power the vast majority of the time, yet with no penalty of reduced functionality from protocol timeouts due to dropped packets—because the system can wake and respond as needed.

4. Packet Accumulation

Packet classification, as described earlier in this paper, ensures that the system only processes what it needs to process, and therefore goes a long way to minimizing power consumption in networked devices. However, further optimization is not only possible but desirable in real-life systems.

Real-world applications can contain an extensive software footprint, and the time to boot is non-trivial. In systems where clock cycles are measured in nanoseconds, the time for software to boot may be measured in seconds.

"Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage"[3] lists a 10-second boot as the shortest achievable today for a PC, although per Microsoft itself[8], Microsoft® Vista requires resumption from its S3 sleep state in two seconds. Even lower boot times are achievable in embedded applications; for example, Lineo[9] lists an optimized boot of 1.07 seconds for X-Windows on the Armadillo 500-FX platform, and 1.9 seconds for Android on the same platform.

Even with packet classification, the arrival rate of packets that require further processing may approach the time it takes for the system to wake to process a packet. In such a scenario, the system will be continually waking and sleeping, without ever being able to spend any significant time in the lowest power dormant state.

This situation can be improved through packet accumulation. With the benefits of packet accumulation, multiple packets can be buffered until such time as the system is ready to wake to process them. This minimizes the overhead of waking and sleeping, allowing the system to efficiently process a group of packets in bulk.

There are several caveats to be aware of when using packet accumulation:

- The system must be able to help guarantee that its packet accumulation buffer does not overflow, regardless of whether that buffer is in dedicated on-chip SRAM in an SoC or in external DRAM. This implies that while performing packet accumulation, the system maintains a count of the number or size of received packets to help ensure that it does not exceed the available buffer, and that the system wakes before this occurs.
- The system must be able to respond to packets within a defined maximum time, regardless of network traffic. This means that as the system starts to accumulate packets, a timer must be started. When the timer expires, the system needs to wake up, regardless of how many packets have been accumulated. This prevents network protocols from timing out if a packet is received and accumulated, but subsequently the network has relatively little relevant traffic to force the packet accumulation buffer to fill.
- There may be certain types of packets for which it is desirable to wake immediately and process, rather than accumulate multiple packets to process. For example, for a networked printer it may be desirable to respond and accumulate multiple ARP requests before wake, but if a packet that looks like the beginning of a print job arrives, then the system should wake immediately in order to print as soon as possible. "Long Idle: Making Idle Networks Quiet for Platform Energy-Efficiency"[2] uses heuristics to differentiate between packets that are "idle" (bufferable) and packets that are "active" (desire fast response), but relatively simple deep packet inspection is also a workable solution.

An example of a product performing both packet classification and packet accumulation with deep packet inspection is the Freescale QorIQ P1022 communications processor[10]. It can both classify packets with its eTSEC controller, and also accumulate packets as needed, storing them in external DRAM, while maintaining counters in its eTSEC and timers in its interrupt controller to guarantee packet response within predefined maximum times.

5. Autorespond Proxy

The shortcoming of packet classification and accumulation on larger networks, such as that found in corporations, is the amount of time spent servicing protocols such as ARP and SNMP, which are required to maintain network connectivity. As an example, if it takes a system 500 ms to go through a cycle of wakeup, message processing and return-to-sleep, then even modest message frequency (<500 ms) could force a system to stay permanently in a high-power state.

For this reason, techniques recently introduced into network standards have added an intelligent proxy to the network interface to maintain network connectivity. The ECMA-393 standard[11] defines the concept of "full network connectivity" as the ability of the computer to maintain network presence while in sleep and intelligently wake when further processing is required. Microsoft's network driver interface specifications (NDIS)[12] and "Network Power Management for Windows 7"[13] provide a framework for protocol offload. In particular, these standardize the way that systems running Microsoft Windows 7 can allow IPv4 address resolution (ARP) and IPv6 network solicitation (NS) to be offloaded to an external network interface controller (NIC), rather than the primary Windows host.

The "proxzzzy for sleeping hosts"[14] is not tied to the Windows 7 operating system and is therefore more suitable to a wide range of embedded applications. Similar to Microsoft's NDIS and power management[12, 13], it also has the requirement for IPv4 ARP and IPv6 NS proxying. It goes further to provide options of further proxying of other protocols such as IGMP, DHCP, IPv4 SIP, IPv6 Teredo tunneling, SNMP, mDNS and LLMNR.

Fundamentally, however, the concept of all such proxying is similar—to maintain Energy Star specifications'[11] "full network connectivity" by using some sort of hardware that is distinct from the primary processor in a system that would otherwise maintain network connectivity. The intent here is for the proxying hardware to be much lower power than the primary processor, thereby allowing the primary processor to be in a low-power state, or potentially even off, for extended periods of time.

Nedevschi, et al.,[1] also implement several types of autorespond proxy in their "proxy_2" through "proxy_4" definitions, although there is no concept of packet accumulation. Their paper analyzes in detail the types of incoming packets and provides information as to which may be the best "low-hanging fruit" protocols to proxy. This data shows that the best protocol to proxy is ARP, as it is the highest percentage of incoming broadcast traffic, and packets destined for the host cannot be ignored. However, other packets which the paper classifies as both "don't wake" (because they may occur frequently) and "mechanical response" (for which a proxy autoresponse may be possible) includes the protocols SSDP, IGMP, ICMP and NBDGM, as well as ARP. All of these and potentially other protocols may be considered as candidates for an autorespond proxy, although a definitive list is highly network dependent and an optimized proxy should be tuned to specific use cases. Where the classification and accumulation of packets can be handled in more cases by an intelligent Ethernet controller with scratch pad memory, an autoreponding proxy typically requires an additional processing element capable of running a networking stack. In a quiet network, the power saving between classification and accumulation methods and autorespond proxy are similar. However, for real networks the cumulative power saving can be an order of magnitude greater if the packet classification and accumulation techniques fail to keep the system in an idle state most of the time.

Another key consideration when implementing an autorespond proxy is that the system must behave similarly on a network as a fully powered version of itself. In the printer example, the PC sending a print job must not see noticeable differences in communication interactions with the printer, or else it could break driver compatibility across diverse OSs and platforms. The printer with an autorespond proxy can't make assumptions that the devices it communicates with have ability to deal with its special "low power modes." Many devices such as network attached storage (NAS) and set-top boxes—which have similar workload profiles of being online but having its main purpose idle most of the time—would see similar power reduction benefits by implementing an autorespond proxy.

An autorespond proxy can be implemented either externally in a "smart" NIC or embedded in an equivalent integrated function on an SoC. The Freescale QorIQ T1042 communications processor[20] is an example of a device which implements an integrated autorespond proxy function. Firmware running on its frame manager can handle or terminate ECMA-393 protocols, while the majority of the device is in a low-power idle state. This capability enables systems to achieve less than ½ W while maintaining full network connectivity, as measured from the AC wall plug.

6. Saving Energy with an Implementation of Network Standby

To demonstrate how low-power network standby and advanced power management features can reduce overall system energy consumption, consider the following real-world example. In this example, only the power of the embedded processor including I/O is considered, rather than overall system power. The power when in network standby is of interest—not the power when performing other primary embedded functionality, such as a printer printing, a DVR recording a TV program, or so on, which is likely to be significantly higher.

The system under consideration has the following properties, based on using an instance of a product, such as the Freescale P1013/P1022[10]:

- 1.8 seconds to wake to full operation, including voltage ramp-up and software boot. This assumption is a mid-range combination of the 1.07 s X-Windows and 1.9 s Android wake times per Lineo[9] and the Windows 8 wake from S3 state of 2.0 seconds per Microsoft[8].
- 1 ms per packet to process a packet (based on the assumption of a host with a maximum processing rate of up to 1 million packets per second).
- 300 mW SoC power dissipation when in network standby mode, at room temperature (25° C junction temperature).
- 5 W SoC power dissipation when active and running typical code, 65° C junction temperature.
- In an otherwise idle network, packets, regardless of whether they require response or not, are assumed to arrive every 80 ms. This is equal to 12.5 packets per second, per Gobriel, et al[2].
- Packets that require response arrive on average every 3 seconds, but only require response every 60 seconds (these values are protocol and network dependent, but represent realistic assumptions).
- Packets that cannot be responded to by an autorespond proxy arrive relatively rarely, less often than once every 60 seconds, and in the context of an autorespond proxy, such distinction can be ignored from an average system power consumption perspective.

In a legacy system not implementing either packet classification or packet accumulation, the system would never be able to spend any meaningful time in a deep sleep mode. This is because the time to wake (1.8 seconds) is significantly longer than the time between arrival of packets (80 ms). On average, each time entering deep sleep, it would only be in deep sleep for 40 ms (half the average packet arrival rate), and it would be awake for at least 1.8 seconds. Therefore the average power required to maintain network standby would, for all intents and purposes, be equal to the max power, namely 5 W.

In a system implementing packet classification but not packet accumulation (such as the MPC8536E PowerQUICC IIITM integrated processor[7]), the system would wake on every interesting packet (on average every 3 seconds), and be awake for the time to boot, plus the time to process one packet. This is a period of (1.8 s + 1 ms) every 3 seconds.

Therefore, the average power required to maintain network standby with packet classification would be:

$P_{ave} = t_{active}/t_{total} * P_{active} + (t_{total} - t_{active})/t_{total} * P_{ns}$ (1)

Where:

P_{ave} = average power in a given time period t_{total}

P_{active} = power consumption when in active mode

P_{ns} = power consumption when in network standby mode

t_{active} = time during total when in active mode (not in network standby), or in the process of waking into this mode

t_{total} = total length of the time period under consideration

Using (1), the power dissipated in a system with packet classification but not packet accumulation is:

(1.8 s + 1 ms)/3 s * 5 W + (3 s – 1.8 s – 1 ms)/3 s * 300 mW
= 60% * 5 W + 40% * 300 mW
= 3.12 W

This is summarized in table 1, below.

In a system implementing both packet classification and packet accumulation (such as the Freescale P1013/P1022[10]), the system would wake every 60 seconds in order to respond to packets before the worst-case 60-second timeout. On average, 20 packets requiring response would have arrived in that time. In this system, t_{active} then reduces to 1.8 s + 20 * 1 ms, which is the time to wake up, plus the time when in active.

Therefore, using (1) once again, the average power required to maintain network standby with both packet classification and packet accumulation would be:

(1.8 s + 20 * 1 ms)/60 s * 5 W + (60 s–1.8 s–20 * 1 ms)/60 s * 300 mW
= 3% * 5 W + 97% * 300 mW
= 0.44 W

In a system that utilized an autorespond proxy, for the workload stated here, the system can remain in the network standby state the entire time. In other words, the system need never enter the active state for any typical network packet processing. For the QorIQ AMP series T1042 communications processors[20], the power consumed with the autorespond proxy active when in the network standby state is typically 150 mW. Thus, this is also the average power consumption of a system with an autorespond proxy.

7. When to Use Each Technique

The system described above was only an example system, and it is not necessarily the case that the same conclusion will be reached in all systems regarding the relative merits of packet classification, packet accumulation and autorespond proxies. As such, there are some generic recommendations that can be made as to when to use each technique.

First of all, given that there is no mode in which the power of a legacy system is less than the power of a system with packet classification, it follows that it can be generally recommended to always perform packet classification. Waking only on interesting packets rather than all incoming packets is a relatively small amount of additional hardware complexity, and thus is recommended.

Similarly, the incremental hardware and software complexity to support packet accumulation as well as packet classification can likewise also be recommended in general. Separate studies may be done in the future on the optimal number of packets or sizes of buffers to support for packet accumulation, but in its simplest and most minimal form, packet accumulation utilizes memory that exists in a system with packet classification for other purposes. This memory may exist, for example, to store the system context for waking from the network standby mode, or it may simply be the size of memories required to receive a maximum-sized Ethernet jumbo frame of the order of 9000 bytes. As such, the incremental power to support packet accumulation rather than simply packet classification may be negligible, and thus, packet classification with packet accumulation can be broadly recommended.

For applications on small networks where the network connectivity messages are expected to be infrequent, the simpler approach of packet classification combined with packet accumulation may be optimal as it can lead to slightly lower system cost and power. However, in most cases the power and cost issues are minimally impacted when implementing autorespond proxy capabilities. Even in the cases where packet classification and accumulation are acceptable, over time most commercially available NICs and SoCs will likely adopt the autorespond proxy due to its more general-purpose nature. Hence, an autorespond proxy is the best general-purpose solution, particularly when the power saving over the life of the product is considered.

Table 1: Average Power Consumption for Various Network Standby Techniques

System	Average Power Consumption
Legacy System	5.00 W
Packet Classification but Not Packet Accumulation	3.12 W
Packet Classification and Packet Accumulation	0.44 W
Autorespond Proxy	0.15 W

8. Conclusion

Embedded and networked computing applications are all around us, everywhere we go, but particularly in the home and office. Designers are severely challenged to continue feeding the industry with increased product performance while adhering to constantly shrinking energy budgets.

Within the cyclical states of the embedded networked application, there are three techniques for low-power network standby that were evaluated. Packet classification saves power by allowing systems to parse and drop packets that do not need to be responded to. Packet accumulation extends the time the system can remain in the network standby state by buffering packets for response at a later point in time. An autorespond proxy provides separate hardware to respond to common packet types without waking the system.

Going forward, the increasingly stringent government regulations and public desires will require network standby to ultimately approach 0 W. The challenge will be to further innovate and reduce network standby power.

9. References

[1] S. Nedevschi, J. Chandrashekar, J. Liu, B. Nordman, S. Ratnasamy, and N. Taft, “Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems,” USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2009.

[2] S Gobriel, C Maciocco, T. C. Tai, “Long Idle: Making Idle Networks Quiet for Platform Energy-Efficiency,” International Conference on Systems and Networks Communications (ICSNC), 2010.

[3] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta, “Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage,” USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2009.

[4] IEEE, “IEEE Standard 802 Part 3, Amendment 5: Media Access Control Parameters, Physical Layers, and Management Parameters for Energy-Efficient Ethernet,” IEEE Standard 802.3az, 2010.

[5] Frequency dependence on Leakage: K. Roy et al., “Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits,” Proc. IEEE, vol. 91, no. 2, pp. 305–327, February 2003.

[6] D. Plummer, “An Ethernet Address Resolution Protocol,” IETF RFC-826, November 1982.

[7] “MPC8536E PowerQUICC III™ Integrated Processor Reference Manual,” Freescale Semiconductor, MPC8536ERM Rev1, 2009.

[8] Windows Hardware Certification Requirements for Client and Server Systems. **msdn.microsoft.com/en-US/library/windows/hardware/jj128256**

[9] Lineo Warp Website, “Products and Services – Warp,” **lineo.co.jp/modules/products/warp2.html**.

[10] “P1013/P1022 Fact Sheet,” Freescale, QP1022FS Rev1, 2011.

[11] ENERGY STAR specification, “ENERGY STAR Program Requirements for Computers,” Version 5.0, **energystar.gov/ia/partners/prod_development/revisions/downloads/computer/Version5.0_Computer_Spec.pdf**.

[12] Microsoft, “Network Driver Interface Specifications (NDIS),” Version 6.20.

[13] B. Combs, “Network Power Management for Windows 7,” Microsoft Windows Driver Developer Conference, 2008.

[14] ECMA standard, “Proxzzzy for Sleeping Hosts,” ECMA-393, February 2010, **ecma-international.org/publications/files/ECMA-ST/ECMA-393.pdf**.

[15] K. Sabhanatarajan, A. Gorden-Ross, M. Oden, M. Navada, and A. George, “Smart-NICs: Power Proxying for Reduced Power Consumption in Network Edge Devices,” IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2008.

[16] M. Gupta and S. Singh, “Greening of the Internet,” ACM SIGCOMM, Karlsruhe, Germany. August 2003.

[17] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, “Reducing Network Energy Consumption via Sleeping and Rate Adaptation,” USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2008.

[18] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-Efficient Communication Protocol forWireless Microsensor Networks,” Proc. 33rd Hawaii Int. Conf. System Sciences (HICSS), Maui, HI, Jan. 2000.

[19] J. Shafer and S. Rixner, “A Reconfigurable and Programmable Gigabit Ethernet Network Interface Card,” Technical report TREE0611, Department of Electrical and Computer Engineering, Rice University, December 2006.

[20] QorIQ T Series T1020/22 and T1040/42 Processors Fact Sheet,” Freescale, T1FAMILYFS REV 0.

freescale™

For more information, visit freescale.com/QorIQ

Document Number: LPOENETSTBYHOWP REV 0