Energy-Efficient Network Memory for Ubiquitous Devices

ENERGY AND DELAY TRADEOFFS OCCUR WHEN A DESIGN MOVES SOME OR ALL LOCAL STORAGE OUT OF THE EMBEDDED DEVICE AND INTO A REMOTE SERVER. USING THE NETWORK TO ACCESS REMOTE STORAGE IN LIEU OF LOCAL MEMORY CAN RESULT IN SIGNIFICANT POWER SAVINGS.

Joshua B. Fryman
Chad M. Huneycutt
Hsien-Hsin (Sean) Lee
Kenneth M. Mackenzie
David E. Schimmel
Georgia Institute of Technology

Mobile applications constantly demand additional memory, and traditional designs increase dynamic RAM (DRAM) to address the problem. Modern devices also incorporate low-power network links to support connected ubiquitous environments. Engineers attempt to minimize network use because of its perceived high consumption of power. This perception is misleading. For 1-Kbyte application pages, network memory is more power efficient than one 2-Mbyte DRAM when the mean time between page transfers exceeds 690 ms. During each page transfer, the application delay to the user is only 16 ms.

Embedded systems for consumers add more features while shrinking the physical-device size. Current 2.5 or 3G (third generation) cell phones incorporate 144 Kbps or better network links, offering customers not only phone services but also e-mail, Internet access, digital camera features, and video on demand. With feature expansion demanding additional storage and memory in all computing devices, DRAM and flash memory densities are increasing in an attempt to keep pace. This continuous storage expansion translates into growing power dissipation, increased temperature, and battery drain.

To reduce energy drain and increase battery life, designers use the smallest parts and fewest possible components. This minimalist approach has the added benefit of keeping manufacturing costs down, but works against application feature expansion and device flexibility for dynamic upgrades.

In an attempt to address some of these problems, companies such as NTT Japan are investing time and research in solutions that allow for mobile computing—dynamically migrating application code between the remote device and other network-connected systems.[1]

One avenue for power savings has not been fully considered, however. Many embedded devices, and all mobile devices, have a network link (based on GSM, Global System for Mobile Communications; Bluetooth; Ethernet; and so on) in a larger distributed environment. After designers incorporate sufficient power to support a network link, they attempt to minimize the link's use

Published by the IEEE Computer Society

because of its excessive energy needs when active. Products therefore incorporate all the needed local storage in the device, buffering as much as possible to avoid retransmission. This ignores the fact that the remote server has a much less restricted power budget and is easily made more powerful to quickly handle requests.

For ubiquitous always-on devices such as 3G cell phones, there is the potential to use the network link as a means for remotely accessing applications. This remote access could reduce local storage space, thereby reducing energy demands on the mobile platform. Remote memory could reside in a remote server or within the network infrastructure.

Using the network link to access remote memory can provide a more energy-efficient solution than traditional local memory. Traditional designs assume that the additional cost of using the network link for moving code and data will far outweigh any benefit of removing or reducing local storage. The common misconception assumes that the network is in use constantly, and therefore is much more power consuming than local storage.

This situation is not always the case, as we will demonstrate. The best low-power mobile DRAM available today is 10 to 100 times less expensive to access in terms of energy-per-bit than a very low-power Bluetooth network. However, for these same parts, the sleep-mode current of the Bluetooth network module is 10 to 100 times less expensive than the DRAM part. Therefore, if sufficient time elapses between accesses, the network link is more power efficient than local DRAM.

Device models

To investigate the possible performance effect of using the network as a mechanism for accessing remote storage, we must consider different device models and characteristics. We examined three fundamental models of embedded computing devices: legacy, pull, and push. One can characterize each model by its type of network link and communication characteristics. We assume that applications exhibit sufficient locality such that there are well-defined *working sets* that change infrequently.[2]

We consider each model independently. Although it's possible to make general comparisons across models, each has different design-time characteristics, which make direct comparison difficult. The underlying hardware design behind each model is the same, however, as shown in Figure 1a. In this device, the program and data values are copied from flash memory to local DRAM for performance reasons. This copying requires sufficient DRAM to hold all or part of the flash memory contents.

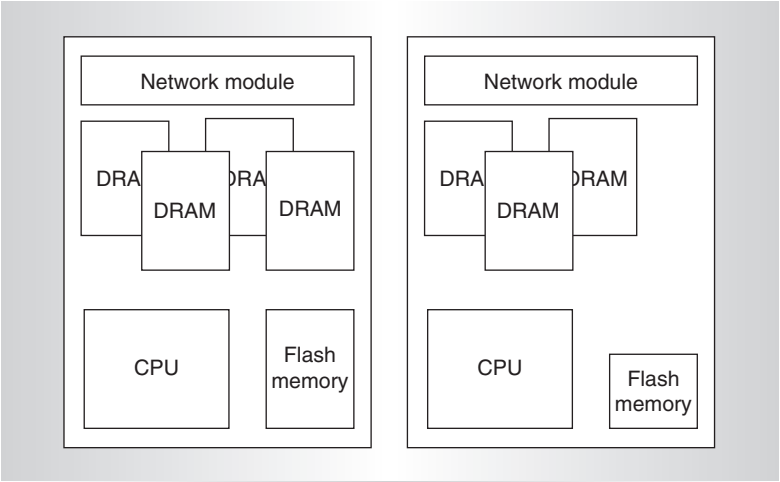

Figure 1. Basic 3G cell phone or other ubiquitous networked device: A typical mobile embedded device (a) and the small reduction proposed here (b).

We propose that using the network link to access the equivalent contents of flash memory from a remote server will provide a more energy-efficient model at a lower cost. This becomes possible by reducing the flash memory component to just a boot-block-sized unit and removing some part of DRAM from the local storage. The removed DRAM would normally contain the contents of flash memory copied during boot-up or when an application changes mode. Instead, we propose reserving a space in local DRAM that is large enough to hold the worst-case working set and all local data. Figure 1b shows this reduction concept.

Although we suggest removing DRAM chip(s) and resizing of flash memory storage, these actions are not absolutely necessary. By carefully using V_{DD} gating, we could disable each DRAM and flash memory unit when it is not needed. This V_{DD} gating would result in power tradeoffs similar to those presented here, but would not provide the increased flexibility for future application insertion and

patching. Moreover, the total manufacturing cost of our design decreases, whereas that of V_{DD}-gated units does not decrease (and may even increase).

Next, we introduce each of the three embedded system models and the notation to analyze the energy and delay issues inherent in each; another work presents details of the equations.[3]

Legacy

Typically, designers started to conceive and construct legacy devices without anticipating a need to communication with other systems. So we examine the issues of energy and delay in this model by assuming we must add a network link and reduce local storage. The legacy application remains unchanged, but the code and data now come from network memory.

The original design expected a certain amount of normal energy consumption during computation and sleep or idle times. To see how adding a network link impacts this situation, we model the extra energy incurred by using the network link to fetch new code and data, and the energy consumed by the network link when in a sleep mode. We assume the network link is only used for fetching new code and data, and that the legacy application itself is not attempting to communicate to other devices. We also model the extra time the CPU now spends waiting for network transactions to complete.

To request new code or data, the device must generate and send a message to the remote server. Transmission time T_{Tx} will consume energy as determined by the type of network link, E_{Tx}.

Once the remote server receives the request, there is some interval of time spent processing request T_{Srv}, during which there will be additional energy consumption, E_{Srv}, on the local device monitoring the network. Once processed, the server will reply with the necessary information, which takes time to receive (T_{Rx}), consuming more energy E_{Rx}. The transmission *payload* will consist of bits that consume power in proportion to the rate of network communications.

In comparison, local storage only incurs a very minor time to access (T_{DRAM}), with a correspondingly small energy use, E_{DRAM}. Whether the system transfers data by network or from local storage, the CPU will be idle during these transfers, consuming some amount of energy. However, the CPU could work on other tasks during this time, thus creating a different energy signature, an energy savings we discuss later.

Regardless of the method used—network or local storage—after transferring the payload, the CPU spends time T_{busy} in computation before generating the next request. During this time, the CPU will consume a different amount of energy, E_{busy}, and the backing store can enter into a power-down or sleep mode. Thus, during the work period, the network link and local storage will consume their respective sleep power.

The total energy consumed by the network link (E_N) in the legacy model is $E_N = E_{Tx} + E_{Srv} + E_{Rx} + E_{busy}$, and the total energy in the local storage (E_L) is $E_L = E_{DRAM} + E_{busy}$. In terms of energy, the network model is equivalent to the local storage model when $E_N = E_L$, but to consider the delay impact on application performance, we construct the energy-delay product

$$E_N \times (T_{Tx} + T_{Srv} + T_{Rx} + E_{busy}) = E_L \times (T_{DRAM} + T_{busy})$$

Solving this equation for T_{busy} provides the energy-delay equilibrium point where using a network store is equivalent to using local DRAM. When T_{busy} is greater than this equilibrium value, the network link is more energy-efficient from a total system perspective. That is, so long as the next application page fetched from the network occurs on or after computation time T_{busy}, the network memory model is more efficient.

Pull

Unlike the isolated legacy model, the pull model assumes that the embedded devices already incorporate a network link. The characterization *pull* comes from how the device uses a network: The local device, on its own initiative, pulls information from the network. External network devices cannot arbitrarily send information to a device operating in pull mode.

Using the same notation as the legacy model, there are only minor differences in the energy analysis. In the pull model, the original design already budgeted power for a network link. They expected the link to be in a

Table 1. Best-case energy consumption of typical DRAMs for mobile applications.

Vendor	Model	Size (Mbytes)	Width (bits)	Speed (MHz)	Supply voltage (V)	Access current (mA)	Sleep current (mA)	Access energy (pJ/bit)	Access energy (pJ/bit/Mbyte)
Elpida	EDL1216AASA	16	16	133	2.3	80	1.5	86.5	5.4
Fujitsu	MB82D01171A-80	2	16	125	2.3	20	0.2	23.0	11.5
Micron	MT48V4M32-10	16	32	100	2.3	100	0.35	71.9	4.5
Micron	MT48V16M16-10	32	16	100	2.3	80	0.35	115.0	3.6
NEC	mPD4664312	8	16	150	2.7	45	0.1	49.6	6.2
Samsung	K4S643233-75	8	32	100	2.3	85	5	61.1	7.6
Samsung	K4S283233-75	16	32	100	2.7	220	6	185.6	11.6
Samsung	K4S561633-1H	32	16	100	2.7	130	6	219.4	6.9

power-down sleep mode during normal operation, except when the program requests remote activity. So for our modification, we only need to calculate the impact of new behavior (our additional traffic) over the original expected behavior (sleep mode). Therefore, we consider the difference between the network link in sleep mode as opposed to actively sending and receiving messages.

These observations modify the original assumption—that all network link energy was a new burden. We subtract the energy required for sleep mode from that required to transmit and receive information. This change represents the new burden on the power source.

Push

Similar to the pull model, the push model also assumes an already available network link. In contrast to the pull model, the network link is always on so that if not actively transmitting, it is in receive-listen mode. Thus external network services can immediately push information—such as e-mail notices and software patches—to the local device.

Just as the pull model reduces the energy drain of the legacy model, the push model reduces the drain further. Because the device for a push model device assumes an always-active receive-mode network, the original design allotted sufficient power for this purpose. Therefore, we subtract the power term for normal receive-mode network links, rather than the smaller power term for a sleep-mode link as in the pull model. That is, we only account for the additional energy of both sending extra messages out and idling the CPU during responses.

Basic analysis

We now analyze in detail both the energy equilibrium point and the energy-delay product for each of these three modes. To provide a quantitative analysis, we obtained technical data for current DRAM and flash memory products.

Using data sheets available from vendors such as Elpida, Fujitsu, Micron, NEC, and Samsung, we selected low-power or mobile-device parts to represent typical commodity part performance. We calculate the energy consumption in terms of picojoules per bit by computing the best-case power consumption listed in each product's electrical characteristics. This gives us a relative measure of energy used in a best-case situation to read or write to the local storage device. During sleep mode, these devices consume very low current but still require some power for refresh functions. Table 1 shows these calculations for DRAM.

Similarly, we calculate energy information from the data sheets published by several network link vendors. Contrary to the work for DRAMs, worst-case power-per-bit is the parameter of interest, as well as the standby or sleep-mode power. In this situation, we consider transmit (Tx) and receive (Rx) modes separately, since some links display different profiles in different operating states. We restricted our search to monolithic, fully integrated network modules to ensure valid power measurements. Using multichip solutions requires external components and glue logic, which make power calculation difficult, if not impossible. Table 2 shows the components we considered and their power calculations.

Table 2. Worst-case energy consumption of typical network links for mobile applications.

Vendor/model	Protocol type	Range (m)	Speed (Kbps)	Supply voltage (*V*)	Transmission/ receiving currents (mA)	Sleep-mode current (μA)	Worst-case energy consumption transmission/ receiving (μJ/bit)
AMI Semi/ASTRX1	SpreadS	300	40	3.3	14/25.0	10.0	1.155/2.063
AMI Semi/A519HRT	Modem	Unavailable	1.2	5.0	0.6/0.6	Unavailable	2.500/2.500
CSR/BC2-Ea	Bluetooth	100	1,500	1.8	53/53	20.0	0.064/0.064
MuRata/LMBTB027	Bluetooth	100	1,000	1.8	60/58	30.0	0.108/0.104
NovaTel/Expedite	Wireless	Unavailable	38.4	3.3	175/130	5.0	15.039/11.172
OKI Semi/MK70	Bluetooth	100	921.6	3.3	115/72	Unavailable	0.412/0.258
Option/GlobeTrotter	GSM	Unavailable	116	3.3	550/50	50.0	15.647/1.422
Radiometrix/BiM-UHF	UHF	30	40	5.0	21/16	1.0	2.625/2.000
Siemens/SieMo S50037	Bluetooth	20	1,500	3.3	120/120	120.0	0.264/0.264
UTMC/UT63M1xx	Bus	Unavailable	1,000	5.0	190/40	Unavailable	0.950/0.200
Vishay/TFBS560x	IrDA	Varies	1,152	5.0	120/0.9	1.0	0.521/0.004
Wireless Futures/ BlueWAVE 1	Bluetooth	100	115.2	3.3	60.9/60.9	50.0	1.745/1.745
Cypress*/ CYWUSB6941,2	W-USB	10	1,000	3.3	120/135	20.0	0.396/0.446
Bermai*/BER7000	802.11a	50	54,000	3.3	454/364	3,030	0.028/0.022

* Only approximations.

For our analysis, we demonstrate a conservative extreme: best-case local storage versus worst-case network links for remote storage.

Although neither of these models is generally realistic, they demonstrate the extreme bounds where network links are more effective than local storage. Thus, in actual application, network links will be more efficient than we demonstrate here.

To understand more exact characteristics of mobile devices that use remote storage, we define models for memory, networks, and CPUs.

Best-case memory

To construct the best-case memory power model, we carefully choose to ignore certain effects in the CPU-to-memory interaction. Because flash memory is substantially slower than DRAM, a device based on our proposed scheme copies the application from flash memory to DRAM for faster execution and then places flash memory in deep-sleep mode or uses V_{DD} gating to disable it. Therefore, we ignore the contribution of flash memory to the total energy. We also ignore the effects of initiating and waiting for memory access and assume all accesses begin instantaneously at the DRAM device's maximum supported rate.

Moreover, we define the transition from idle or sleep mode to active mode as instantaneous. We choose minimal V_{DD} and current consumption at all times and ignore energy drawn by refresh operations. We also assume that any accessed code or data is in the DRAM and does not load from flash memory.

This constitutes a best-case memory model. For our analysis, we use the Fujitsu FCRAM model MB82D01171A. This 2-Mbyte DRAM has the lowest power consumption (in terms of picojoules per bit) of all the devices listed in Table 1.

Worst-case network

For this analysis, we restrict the additional traffic needed to support the network memory model to unalterable content such as programs, static global data, and so on. We further model the request for code or data to a remote server as fully encapsulated in a 64-byte packet. It is possible to reduce or expand

this packet size based on the network topology and error-handling needs, but at 64 bytes, the packet has sufficient storage space for a wide range of requests. The response packet, a variable-payload version of the request packet, will consist of 20 bytes for control information, followed by the actual, variable-size payload. We use these values as the basis of our client-server system implementation.[4]

We assume that for the total count of DRAM chips, at least one is for mirroring part or all of flash memory. Based on the working-set principle, the system only needs a small fraction of this space at any given moment. So rather than store a large mirror image, an energy-efficient design should only reserve sufficient space for the worst-case working set in local DRAM, eliminating excess DRAM. By using the network link to access applications, we could also shrink the flash memory such that it contains only a boot image and not all the applications that the device could ever run. This also reduces the burden of pushing massive code patches out to all systems in the network. Because steady-state mode changes occur relatively infrequently,[2,5] the need to load new code and data from the network will also occur infrequently.

The worst-case network model uses typical V_{DD} with worst-case current consumption in all cases. With slower transfer rates, higher current consumption, and a long duration of remote-server processing T_{Srv}, the network appears unattractive for energy savings at first glance. However, we will demonstrate that this is not the case.

The analysis that follows implicitly uses the concept of one computational task running at a single time on the mobile device. So we model the CPU as completely idle during the time it takes to process additional network transactions to receive new code (using the best-case zero-overhead local memory access). Normally, the CPU would be busy with other work during this time. If multiple tasks were present, the CPU could simply switch to the next task and continue processing. This would not add to the energy overhead of sitting idle and delaying all work and thus is not the worst-case scenario for network impact.

Our analysis assumes the removal of one DRAM chip, although it's possible to reduce flash memory as well as DRAM chips. Our network link model is the CSR BC2-Ea, a fully integrated Bluetooth module. This module exhibits a starting time of 10 µs and a settling time of 5 µs in the internal analog-to-digital converter for gain control.

A transition from active to sleep mode in the network module occurs in under 1 ms. As we will demonstrate later, the server processing time for requests is set to 10 ms. Compared with such a relatively long time for server processing, the transition time between active and sleep mode is small; we can ignore it and other third-order effects of the network module design.

Mobile CPU

Our CPU model for the mobile device is the DEC SA-110, a processor that runs at 0.5 W during times of high computational loads and 0.02 W during idle periods, when it operates at 160 MHz.[6] Several interesting factors arise from using this particular processor as our representative model.

The SA-110 can transition between idle and active mode with effectively no delay. It does so using its two separate clock domains. In idle mode, the internal bus clock and clock grid stop signaling. The actual steps to enable idle mode include toggling a register, loading an uncachable address, and waiting for an interrupt; these steps take only a few instructions. This processor recovers after receiving an interrupt and restoring the original register values. Because the transition between active and idle modes is nearly instantaneous, we do not model the time necessary for it in the CPU.

Initial impact

Given that network transmission speeds lag substantially behind the bandwidth of local memories, the bounds on T_{Srv} will depend on the network speed. With increasing payload size in transfers, the remote server's processing time becomes less important than overall network performance. Figure 2 illustrates the boundaries as T_{Srv} varies from zero to one second.

It is unreasonable to assume there will be zero processing overhead on the remote storage system. The CPU has to receive the incoming network request, invoke interrupt handlers, search memory, and so on. Using our already existing client-server system as a basis,[4] an Intel Pentium III 800-MHz system

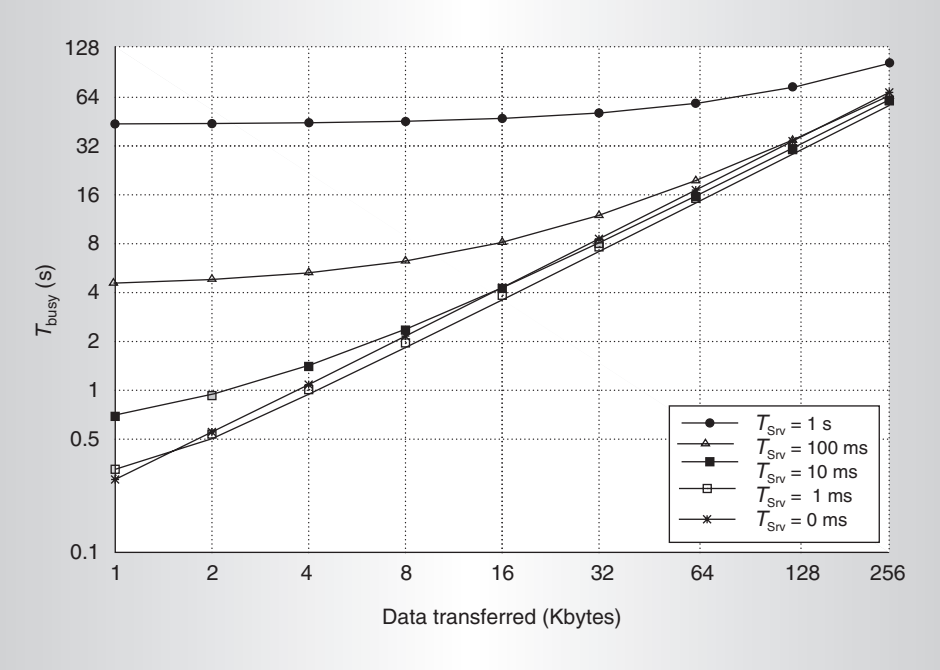

Figure 2. Each line represents the T_{busy} equilibrium point for different remote-server processing times T_{Srv}. The network transmission speed is the limiting factor during payload transfers, shown as the asymptote when T_{Srv} = 0 ms.

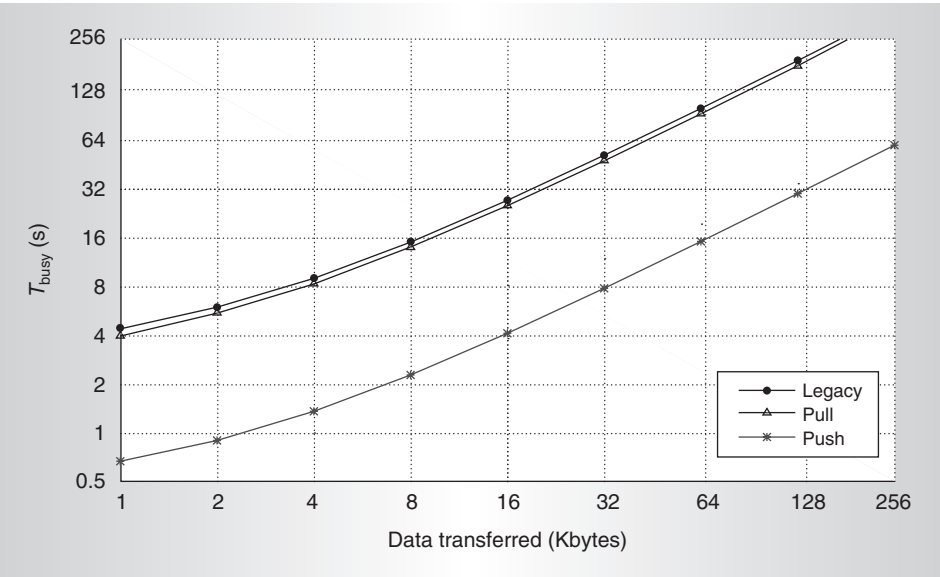

Figure 3. Comparing the energy-delay equilibrium characteristics of legacy, pull, and push models when T_{Srv} = 10 ms.

running RedHat Linux 8.0 is capable of processing and responding to requests in under 10 ms. During this time, the server is also running a fully interactive XWindows desktop with multiple open applications (gcc, gdb, and so on). Therefore, we use 10 ms as an approximate remote-server processing time. Although we can optimize the remote server and make it arbitrarily powerful, it will serve multiple targets, so we would expect similar response times.

To compare the legacy, pull, and push models using our established T_{Srv} of 10 ms, we again plot the necessary T_{busy} to reach the energy-delay equilibrium point. Figure 3 demonstrates the tradeoffs between the three models. Any value of T_{busy} beyond the times shown in this figure indicates that using remote storage is more energy efficient than local storage.

The legacy model presents the worst energy-delay product result. We have added a network link to a design that did not originally incorporate one. For the network link to be more efficient than local DRAM requires a significant amount of time T_{busy} spent in computation.

The pull model provides better energy-delay results than the legacy model, as you might expect from subtracting the sleep-mode power. The improvement turns out to be small compared to the energy costs associated with transferring the data as well as remote server processing time T_{Srv}. The actual difference between the legacy and pull models is slightly less than 8 percent. This indicates that adding a network link to a legacy system that uses a pull-based communication model has a small impact compared to that of the energy consumed by local storage devices.

The push model uses the least additional energy and thereby benefits most from using

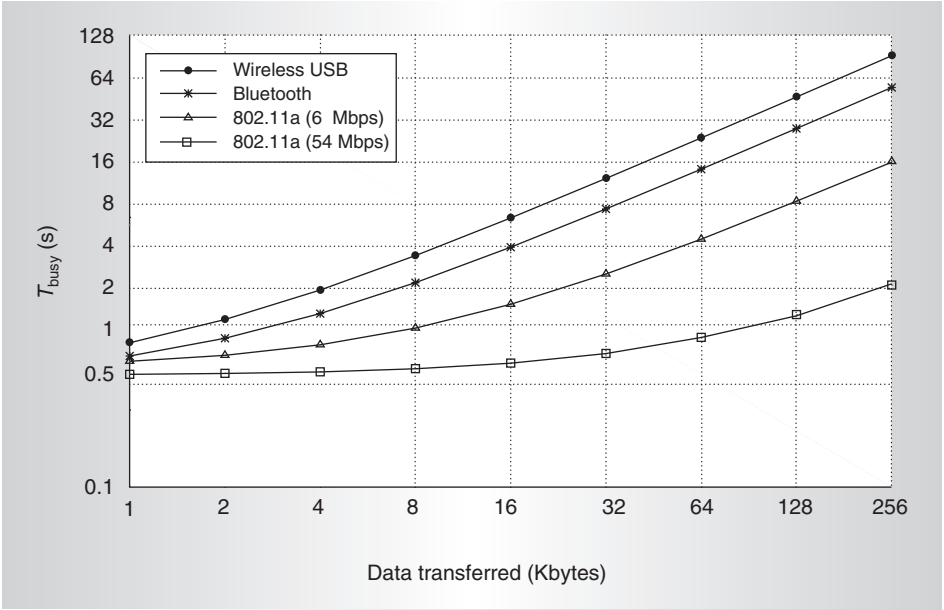

Figure 4. Comparing the push model's energy-delay equilibrium characteristics with Bluetooth, Wireless USB, and 802.11a network modules when T_{Srv} = 10 ms.

remote storage. Because designers expected devices following this model to keep the network link in receive mode at all times, the only extra energy for accessing remote storage is the energy of the transmit operations.

Assuming the change of a 1-Kbyte page and a T_{Srv} of 10 ms, the legacy model requires a minimum interval of 4.33 s before it becomes beneficial to access a remote server. With the pull model, the required busy time falls to 3.99 s; with the push model, the time drops to 0.69 s. In relative comparison, this same 1-Kbyte page of code loaded across the network with T_{Srv} = 10 ms will present a total application delay of 16 ms to the user while accessing the network. This includes sending, processing, and returning a payload through the network.

Transfer of a larger page size might be more realistic to consider, however. For a 16-Kbyte change and T_{Srv} = 10 ms, the legacy model requires 26.6 s between transfers, and the pull model, 24.5 s. The push model reduces this time to a mere 4.2 s. The delay the user experiences while the network transfer occurs is 185 ms.

Portability

The preceding results come from a very low-power Bluetooth integrated module. To compare these results to those for other network types, we now consider two alternate network interfaces: Wireless USB and 802.11. Neither of these alternatives comes in complete monolithic solutions, but instead comprises two or three highly integrated chips with minimal, external glue logic. The estimates for the Cypress wireless USB chipset and the Bermai integrated 802.11a chipset include only the main chip components. In these estimates, we do not consider power consumption of the glue logic, and therefore these numbers are slightly smaller than they should be in a worst-case scenario. In particular, the 802.11a chipset has especially large currents in any mode of operation, even before considering the glue logic components.

Using the push model as a baseline, we compare the CSR BC2-Ea solution to both the Cypress and Bermai solutions. Figure 4 displays the results of this comparison. The surprising result from this figure is that the very power-hungry 802.11a network is a much better selection than the low-power Bluetooth or similar modules. The substantially higher data rate causes the limiting factor to be the remote-server processing time rather than the network link speed.

This comparison is against a push model, which assumed sufficient built-in power to

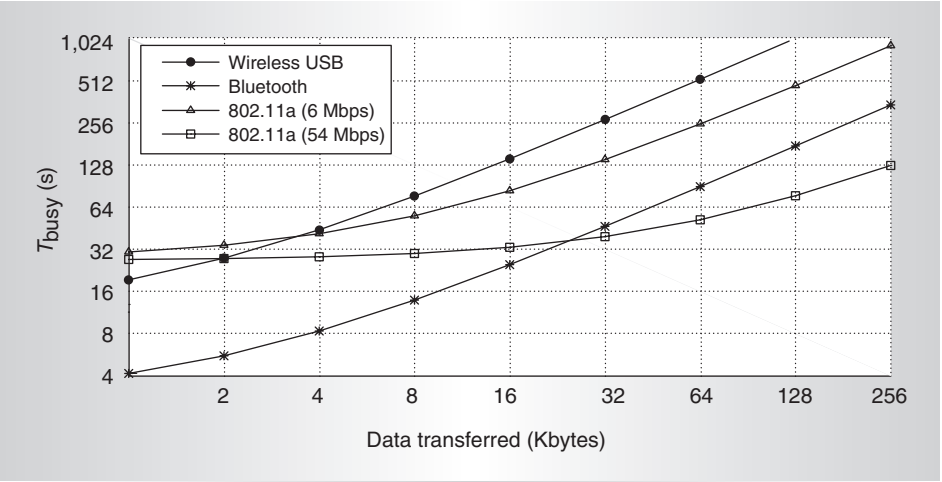

Figure 5. Comparing the pull model's energy-delay equilibrium characteristics with Bluetooth, Wireless USB, and 802.11a network modules when T_{Srv} = 10 ms.

mode. Comparing to the pull model, shown in Figure 5, we can see a more illustrative example of the substantial power drain involved in 802.11 chipsets. Note that the initial energy cost of the 802.11 network far exceeds other options, but that if the typical payload transferred in the network is greater than or equal to 32 Kbytes, the 802.11 network is a better design choice.

Although these results do not specifically tie to any estimated average transfer size, they show interesting trends. Ultimately, the typical payload size will depend entirely on the application and the network support provided by commercial companies. This work shows that careful analysis of the types of applications and data transmitted over the network, and the characteristics of those applications, can prove intuitive design directions incorrect. In some cases, increasing local storage might be the wrong approach to longer battery life.

Related work

Using the network to access RAM is novel as a low-power mechanism. Prior work concentrated on using remote memories for high performance, avoiding accesses to slow disks, or to expand memory for working sets of code or data.[7,8] Other work examining the network in power-limited devices has concentrated and minimizing usage[9] and optimizing protocols.

Researchers are focusing on finding ways to improve a network's overall energy efficiency. Previous research shows that ad-hoc relaying improves wireless local area networks.[10] Other research looks at tying battery level with ad-hoc routing methods to increase network robustness as well as node runtimes.[11]

Using the availability of low-power, short-range devices such as those based on Bluetooth, researchers are building larger energy-efficient networks. These new systems compete with more traditional network options.[12] Such prototypes strengthen the viability of using limited embedded hardware for larger projects.

With each generation of network technology, data rates increase and power consumption decreases. Next-generation technology such as ultra wideband networking will likely have a higher data rate and use less power than current Bluetooth devices. Such networks will also have similar, if not better, ranges.

As network links approach local DRAM in terms of performance characteristics such as bandwidth and power, the arguments for moving to network-based storage become more compelling.

Conventional wisdom has focused on minimizing network use. Here, we mean to cast doubt on such a broadly general rule and to encourage designers to reconsider how devices will be used as a key to minimizing power.

For a reasonable, average working set of 32 Kbytes transferred via a network, the link is more efficient than local DRAM if transfers occur less frequently than every 4.24 s with a user-experience delay of only 0.185 s, using a push model for comparison. This assumes the network link replaces one local 2-Mbytes DRAM chip. Removal of larger or multiple components makes network usage even more efficient.

Today, network links use 10 to 100 times the energy of DRAM during accesses, but consume 10 to 100 times less energy during sleep. Network devices continue to approach low-power DRAM performance characteris-

tics in terms of speed and power, making network memory increasingly attractive, in particular, in ubiquitous environment. The reduction in part count, and thus price, can aid in the marketing of disposable devices such as cell phones.

This model, in which users download applications on demand, also provides a mechanism for pay-per-use services, such as for custom games or video players. We are working on detailed simulations to study the impact of this network memory model on overall network congestion. MICRO

Acknowledgments

We thank the reviewers for their useful comments in restructuring this article. This work was funded in part by the National Science Foundation under grants CCR-98-76180, CCR-01- 21638, and EIA-99-72872.

References

1. NTT Japan, *Bluebird Project,* 2003, http://www.ntts.co.jp/java/bluegrid/en/.
2. G.A. Abandah and E.S. Davidson, "Configuration Independent Analysis for Characterizing Shared-Memory Applications," *Proc. 12th Int'l Parallel Processing Symp.*, 1998, pp. 357-398.
3. J.B. Fryman et al., *Energy Efficient Network Memory for Ubiquitous Devices*, tech. report GIT-CERCS-03-05, Georgia Institute of Technology, 2003.
4. C.M. Huneycutt, J.B. Fryman, and K.M. Mackenzie, "Software Caching Using Dynamic Binary Rewriting for Embedded Devices," *Int'l Conf. Parallel Processing*, 2002, IEEE CS Press, pp. 621-630.
5. R. Batchu et al., *A Study of Program Behavior to Establish Temporal Locality at the Function Level*, tech. report DCS-TR 475, Rutgers Univ., 2001.
6. J. Montanaro et al., "A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, Nov. 1996, pp. 1703-1714.
7. D. Pnevmatikatos and E.P. Markatos, "On Using Network RAM as a Non-Volatile Buffer," *Cluster Computing*, vol. 2, no. 4, 1999, pp. 295-303.
8. S. Dwarkadas et al., "Cashmere-VLM: Remote Memory Paging for Software Distributed Shared Memory," *Proc. 13th Int'l Parallel Processing Symp. and 10th Symp. Parallel and Distributed Processing* (IPPS/SPDP 1999), IEEE CS Press, 1999, pp. 153-159.
9. P.J.M. Havinga and G.J.M. Smit, "Energy-Efficient Wireless Networking for Multimedia Applications," *Wireless Communications and Mobile Computing,* Wiley, 2001, pp. 165-184.
10. M. Kubisch et al., *Applying Ad-Hoc Relaying to Improve Capacity, Energy Efficiency, and Immission in Infrastructure-Based WLANs*, tech. report, Tech. Univ. Berlin, 2002.
11. D. Kim et al., "Power-Aware Routing Based on the Energy Drain Rate for Mobile Ad Hoc Networks," *Proc. 11th Int'l Conf. Computer Comm. and Networks* (ICCCN 2002), IEEE Press, pp. 565-569.
12. S. Baatz et al., "Building Efficient Bluetooth Scatternet Topologies from 1-Factors," *Proc. IASTED Int'l Conf. on Wireless and Optical Communications* (IASTED 2002), Acta Press, 2002, pp. 300-305.

Joshua B. Fryman is a PhD student at the College of Computing, Georgia Institute of Technology; he also has several years in industry, working in the cable and satellite TV area. His research interests include low-power architectures, embedded systems, and high-performance computing. Fryman has a B.S. in Computer Engineering from University of Florida. He is a student member of the IEEE and ACM.

Chad Huneycutt is a PhD student at the College of Computing, Georgia Institute of Technology. His research interests include soft computer architectures and dynamic compilation techniques. Huneycutt has a B.S. in Computer Science from Furman University. He is a student member of the IEEE, ACM, and Upsilon Pi Epsilon.

Hsien-Hsin (Sean) Lee is an assistant professor in the School of Electrical and Computer Engineering, Georgia Institute of Technology. His research interests include microarchitecture, memory systems, information security, and autonomic systems. Lee has a BSEE from National Tsinghua University in Taiwan, and an MSE and PhD in computer science and

engineering from the University of Michigan, Ann Arbor. He is a member of ACM, IEEE, Tau Beta Pi, and Sigma Xi.

Kenneth M. Mackenzie is a member of the technical staff at Reservoir Labs, New York. His research interests include parallel systems, embedded computing, and compilers. Mackenzie has an SB, SM, and PhD from the Department of Electrical Engineering and Computer Science at the Massachussetts Institute of Technology. He is a member of ACM.

David. E. Schimmel is an associate professor in the School of Electrical and Computer Engineering, Georgia Institute of Technology. His research interests include algorithms and interconnection networks for parallel, reconfigurable, and asynchronous computer architectures; and the impact of technology on systems. Schimmel has a BSEE and PhD from Cornell University. He is a member of IEEE, Tau Beta Pi, and Eta Kappa Nu.

Direct questions and comments about this article to Joshua B. Fryman, College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA, 30332-0280; fryman@cercs.gatech.edu.