Gateway Design for Data Gathering Sensor Networks

Raluca Musăloiu-E.
Computer Science Department
The Johns Hopkins University
ralucam@cs.jhu.edu

vRăzvan Musăloiu-E.
Computer Science Department
The Johns Hopkins University
razvanm@cs.jhu.edu

Andreas Terzis
Computer Science Department
The Johns Hopkins University
terzis@cs.jhu.edu

***Abstract*—Innovation in gateways for data gathering sensor networks has lagged compared to advances in mote-class devices, leaving us with a limited set of options for deploying such systems. In this paper we outline the system design of a gateway that meets the hardware and software demands of data gathering networks and present a prototype implementation of this design using the popular Stargate platform. The gateway communicates with the back-end servers through a periodic query-response protocol which is robust to connectivity outages. Commands from the back-end servers instruct the gateway to wake up the sensor network, perform bulk data downloads, and finally upload the collected measurements to a remote server. We investigate mechanisms to minimize energy consumption at the gateway including completely powering it off in between query intervals and batching data uploads. While we ground our design using the Stargate device and evaluate its power consumption using different long-haul radios, we present several alternative platforms which could become viable alternatives in terms of their cost/performance ratio.**

I. Introduction

It is considerably easier to develop and deploy a wireless sensor network (WSN) application today than it was a few years ago. This is mainly due to the wide availability of mote hardware (*e.g.*, [3], [24]) and open source programming frameworks (e.g., [1], [7], [13]).

There is however a missing piece in this otherwise encouraging trend. What we still lack are software and hardware platforms for WSN gateways that are as easy to use and deploy as those that we have for motes. While some solutions do exist (*e.g.*, [4]), integrating them to an application requires significant effort. One needs to develop custom software for the gateway to communicate with the back-end servers as well as the sensor network. Furthermore, deploying such a gateway requires special energy sources, such as rechargeable batteries coupled with solar panels, that increase the cost and complicate the deployment and maintenance of the network.

We present an attempt to resolve this predicament. Specifically: **(1)** We describe an extensible software framework for WSN gateways with the necessary features for connecting WSN applications to back-end servers. While we ground our design using an environmental monitoring application we developed the software can be easily modified to accommodate other applications. **(2)** We present a battery-operated mote gateway based on the Stargate platform coupled with additional hardware that regulates its long term power consumption. As part of this effort we measure the power consumption of the system under various usage scenarios including the use of different long-haul radios (WiFi and cellular modem). Moreover, we present examples of the expected lifetime of such a gateway when its main purpose is to upload data collected by the sensor network. **(3)** Finally, while the current instantiation uses the Stargate gateway platform, we survey the field of hardware and software alternatives. We do so because in many ways the Stargate is not the ideal platform for low-cost, low-power deployments. It is more expensive compared to other hardware components and the complexity of embedded Linux complicates low-power implementations.

We found that, for the combination of hardware we used, completely powering off the gateway is more efficient than putting it in deep-sleep mode if the inactivity time is larger than $\sim$200-300 seconds, depending on the long haul radio used. While cellular modems offer the flexibility of deploying WSNs anywhere there is cellphone coverage, they do so at a considerable cost. Specifically, we found that the 3G cellular radio we use consumes 4-13 times more energy than an 802.11b card to upload the same amount of data. This difference is due to the lower bandwidth of the cellular modem, the higher current it draws, and the longer time required to establish a link. Finally, we estimated that gateways that are powered by high capacity D-cell batteries and use the proposed techniques, can operate uninterrupted up to multiple years depending on the network's data acquisition rate.

This paper has six sections. In Section II we review environmental monitoring networks and outline one architecture for building such networks. Section III presents the gateway's system architecture, while in Section IV we evaluate the power consumption of the proposed architecture. Finally, we outline potential hardware and software alternatives in Section V and conclude in Section VI.

II. Background

A. WSN Gateway Requirements

The research community has deployed over the past few years various wireless sensor networks serving applications that range from environmental monitoring [26], to structural health monitoring [32], and condition-based maintenance [29]. A common requirement among all these *data gathering* networks is the need for one or more gateways to connect the

WSNs with their remote users. This need is amplified by the observation that most deployments are in remote locations with no wired Internet connectivity.

Data gathering gateways perform common tasks: **(1)** Transfer WSN measurements to remote users, **(2)** Transmit user commands and code updates to one or more of the network's motes. **(3)** Alert the network administrators of potential network faults. This functional commonality suggests the possibility of designing a common gateway platform. Equally important is the ability of the gateway to consume little energy, a feature which will benefit all deployments.

B. Data Gathering WSN Architectures

The centrality of data gathering in WSN applications has motivated the development of a number of network architectures for this application prototype [2], [22], [28], [31]. Most of these architectures stream sensor-collected measurements to the gateway over a multi-hop routing tree, persistently maintained by the network's motes. However, Dutta *et al.* recently suggested that decoupling measurement acquisition and delivery can reduce the network's energy consumption [8]. Having independently reached the same conclusion, we developed *Koala*, a data gathering architecture that can achieve permille (.1%) duty cycles for non-real time data gathering environmental monitoring WSNs [25].

The intuition behind Koala is that routing state maintenance unduly increases the application's energy overhead, because nodes need to periodically exchange messages to discover and computer the tree's paths. Instead, Koala keeps the network disconnected most of the time. When the gateway must collect measurements, it wakes up the network's nodes and constructs routing paths to reliably extract all the collected data. A side effect of this architecture, is that the gateway itself does not need to be always on, further reducing its energy consumption.

Given its benefits, we use the Koala architecture as the basis of the WSN that the proposed gateway connects to. The section that follows elaborates on the design of such a gateway.

C. An example: Life Under Your Feet

Life Under Your Feet (LUYF) [26], an ongoing research project whose aim is to measure various soil parameters, air temperature, air humidity and light intensity, provides the motivation for this work. Counting timestamps, each LUYF mote generates 24 bytes per measurement sample. Motes generate samples once per second and store them in their local flash until the gateway extracts them through a reliable transfer protocol built using Koala.

III. System Architecture

We consider a multi-tier WSN architecture with one or more mote patches, each with multiple sensing motes. Moreover, every deployment contains one or more gateways that connect these patches to remote users. Gateways include long-haul radios (*e.g.*, WiFi, or cellular data radios (GPRS, EDGE, HSDPA, 1xRTT, EVDO)) used to communicate with back-end servers and a radio to communicate with the WSN, potentially provided by a mote directly connected to the gateway. Finally, back-end servers store the data generated by the networks and provide a management interface to the end-users.

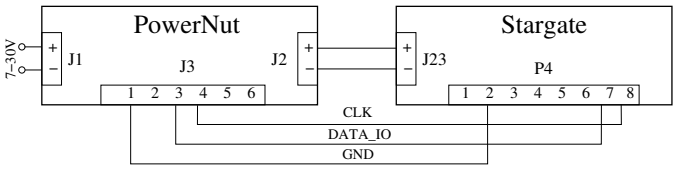

Fig. 1. Schematic of the connection between the PowerNut power control device and the Stargate gateway.

A. Hardware Platform

A survey of the recent research literature suggests that the Stargate gateway [4] is a popular hardware platform for WSN gateways (*e.g.*, [11], [27], [28], among others). While other viable platforms do exist, we selected to ground our design on the Stargate given its popularity and its wide availability.

The Stargate is a single board computer with a 400 MHz 32-bit PXA255 XScale processor, running embedded Linux. It has 32MB of Flash memory and 64MB of SDRAM. Cellular and 802.11 radios can be connected to the gateway's CF and PCMCIA connectors. A separate daughter card provides Ethernet, RS-232, and USB ports as well as an AC power supply. Finally, MicaZ and Mica2 motes can be directly connected to the Stargate, providing connectivity to WSNs with different mote radios. The processor offers two low power modes: a *idle mode*, in which the core clocks stop when the processor is inactive and resume through an interrupt, and a *sleep mode*, in which all processor and peripheral clocks are disabled, except the Real Time Clock (RTC). The SDRAM is placed into self-refresh mode in order to preserve pin states and interrupts are ignored. The system returns to the normal mode via a wake up event, scheduled before going to sleep. We hereafter refer to this second mode as *deep-sleep*.

As Section IV-A shows, the Stargate draws significant current even in deep-sleep mode ($\sim$6-60 mA, depending on the connected peripherals). This means that in order to support long-term deployments large and/or rechargeable batteries coupled with solar panels must be used. Indeed, this is the approach taken by most of the deployments that have used the Stargate up to this date. Since we are interested in gateway designs that use smaller (*e.g.*, D-cell) batteries, we investigate a different operating mode. Specifically, we use a power controlling device to completely power off the gateway when it is not in use. We employ the PowerNut power controller [20], which provides voltage to a downstream device and has the ability to cut the power for a configurable amount of time, upon receiving a command from a controller (in this case the gateway itself). Figure 1 depicts the physical connection between the two devices, while the communication between the controller and the gateway follows a simple serial protocol.

We measured the PowerNut's consumption, when the downstream device is turned off, to be 180 μA in the worst case (58-100 μA typical). Because the gateway consumes no energy in

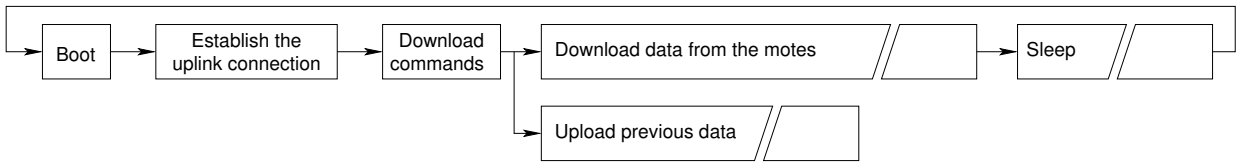

Fig. 2. The gateway's normal operation cycle.

this state and the PowerNut itself consumes very little energy, this solution is more efficient than putting the gateway to deep-sleep mode. On the other hand, the gateway must go through its boot process every time power is restored. This boot sequence, including loading the operating system, can take tens of seconds. Section IV explores the trade-off between powering off the gateway and using its deep-sleep mode.

B. Software Design

Figure 2 depicts the gateway's cycle of operations. After the power controller restores power and the boot sequence completes, the gateway attempts to establish an uplink to the back-end server. Communication between the gateway and the back-end server follows a query-response protocol, in which the Stargate polls the back-end server to retrieve new commands and subsequently reports the results of the commands' execution. We selected this interaction paradigm because it affords the gateway to turn itself off in between queries and allows it to use different IP addresses across queries. This is important because long-haul radios might receive different IP addresses over time, for example when using DHCP. We choose HTTP as the underlying communication protocol, because it follows the query-response paradigm described above and supports authentication and encryption.

The back-end server transmits two main types of commands: **(1)** uploading the previous round of measurements downloaded from the WSN and **(2)** downloading the next round of measurements from the WSN. Because both operations require significant time, the gateway performs them in parallel to minimize the time it is active. Once the upload operation completes, the gateway disables the uplink interface to minimize energy consumption. If the gateway fails to establish a link to the back-end server, it only performs the local download from the WSN. Another special situation occurs when the gateway fails to turn itself off, due to a malfunction of the power controller. In this case, the gateway informs the back-end server of this failure so maintenance can be scheduled.

In addition to the data download operations, the gateway supports the following commands:

1) `TIME <curtime>`
   This command allows the back-end server to update the gateway's clock to `curtime`.
2) `SLEEP <seconds>`
   This command specifies the number of seconds that the gateway should go to sleep at the end of the current activity period. The gateway uses this value to set the time period for deep-sleep or to configure the amount of time the PowerNut power controller will disconnect the power from the gateway itself.
3) `REPORT <count>`
   This command specifies the number of operation cycles after which the gateway must connect to the back-end server to refresh its command file and to return the data it has collected.
4) `SEND_LOG`
   This command is used for debugging and requests the gateway to upload its log file. The log file contains a history of the gateway's actions, including the connections to the back-end server (including failed attempts).
5) `UPDATE <file_location> wsn|gw`
   This command replaces the current binary running on the gateway, or the WSN motes, where `file_location` specifies the new file as an FTP or HTTP URL.

Figure 3 presents a sample gateway log from a simple session with the back-end server. The first line of the log shows the `HTTP GET` request for the command file, followed by the server's response. This is followed by the `HTTP POST` query for uploading a 10 KB file with measurements collected from the network to the back-end server. The session ends with the gateway transitioning to the deep-sleep mode for 18,000 seconds, as instructed by the back-end server through the command file.

C. WSN Interface

There are two types of interaction between the gateway and the WSN: downloading data from the network's motes and reprogramming the motes. We use Koala [25] for the first operation, while the second is accomplished by invoking protocols such as Deluge [6] and Typhoon [21]. In both cases the communication between the gateway and the mote directly connected to it follows the standard TinyOS serial communication protocol [12].

D. Long Haul Connectivity

We consider two alternatives for providing long haul connectivity: WiFi and cellular data radios. The advantage of using a WiFi radio is that WiFi networks are available in many urban areas. On the other hand, cellular data networks, while offering slower data rates at significantly higher cost, have

```
 1 bs_get_data: 64 bytes sent
 2 bs_get_data: 198 bytes received
 3                HTTP/1.1 200 OK
 4 Date: Mon, 05 Mar 2007 00:57:02 GMT
 5 Server: Apache/2.0.52 (Red Hat)
 6 Connection: close
 7 Content-Type: text/plain
 8
 9 TIME 2007-03-04 19:57:02.579513
10 SLEEP 18000
11 REPORT 1
12
13 -----
14 POST /~stargate/motemonitor/results.py HTTP/1.0
15 HOST: polarbear.isi.jhu.edu:80
16 Content-type: multipart/form-data; boundary----------------------------1358580979
   16243791492128236579
17 Content-Length: 10840
18
19 ----------------------------135858097916243791492128236579
20 Content-Disposition: form-data; name="results";
   filename="2007-03-04 19:57:02.579513"
21 Content-Type: application/octet-stream

   [ ... binary data skipped here ... ]

22 ----------------------------135858097916243791492128236579
23 Content-Disposition: form-data; name="submit"
24
25 Submit
26 ----------------------------135858097916243791492128236579--
27
28 -----
29 bs_post_data response: HTTP/1.1 200 OK
30 Date: Mon, 05 Mar 2007 00:57:02 GMT
31 Server: Apache/2.0.52 (Red Hat)
32 Connection: close
33 Content-Type: text/html

   [ ... server response skipped here ...]

34 bs_send_data: 10792 bytes sent
35 deep sleeping... bye-bye...
```

Fig. 3. Sample log of a session between the gateway and the back-end server: request command file via HTTP GET (line 1), server response (lines 2-12), data file upload via HTTP POST (lines 13-28), server response (lines 29-34), enter deep-sleep mode (line 35).

much wider coverage. In addition to network availability and link capacity, one must consider the power consumption of these radios. In Section IV we evaluate two radios from this perspective, an AmbiCom 802.11b WL1100 CF card with a maximum output power of 20 mW, and the Sierra Wireless AirCard 860 HSDPA PCMCIA card.

IV. Evaluation

This section answers three questions: **(1)** When is it preferable to power off the gateway as opposed to using deep-sleep mode, **(2)** which long haul connectivity method offers higher energy efficiency for sending the same amount of data and, **(3)** what is the expected lifetime of a battery-operated gateway.

In all cases, we measure energy consumption in terms of the current drawn by the gateway when it is powered by a regulated 5 V voltage source. Current drawn is measured by recording the voltage drop over a 5% precision 1 Ω resistor connected in series with the whole system.

A. Deep-sleep vs. Power-off Mode

We first measure the current the gateway draws when it is in deep-sleep mode. As the first column of Table I indicates, the average current consumption is 6.28 mA with no attached cards. Adding the WiFi card increases this value to 22.93 mA while the addition of the 3G card increases the current to an average of 41.81 mA. The daughter-board is even more power-hungry, requiring 62.78 mA even with no attached USB devices. The table also includes energy consumption figures during normal operation with active and inactive long-haul

State	Deep-sleep (mA)	Active (mA)	
		Net up	Net down
with DB, no cards	62.78	-	144.28
without DB, no cards	6.28	-	82.56
without DB, WiFi card	22.93	306.14	101.98
without DB, 3G card	41.81	247.48	123.42

TABLE I
Current consumption for different combinations of long-haul radios, with and without the Stargate daughter-board (DB).

Configuration	Time (s)	Average Current (mA)
with DB, no cards	34.70	267.66
without DB, no cards	34.29	176.37
without DB, WiFi card	35.30	202.94
without DB, 3G card	35.29	279.51

TABLE II
Duration and current consumption during the boot sequence.

radios. The time to enter and resume from the deep-sleep mode also varies depending on the configuration, ranging from 0.51 seconds with no attached cards, to 2.19 seconds with the WiFi and 2.97 seconds with the 3G card. These figures indicate that even though the Stargate can quickly resume from deep-sleep mode, its consumption in that state is prohibitively high when low duty cycles are used.

In order to evaluate when operating in deep-sleep mode is more efficient than powering off the device, we must also measure how much energy is consumed during the boot sequence. Even though, as Table II suggests, the time to boot is similar across all configurations, the 3G card draws more current on the average (279 mA) than the WiFi card (202 mA).

We can now answer to the first question by comparing the average current the gateway draws as a function of the inactivity time. We define inactivity time as the time that the device is in deep-sleep mode in one case and the time that the device is turned off and is booting otherwise.

Figure 4 presents this trade-off pictorially. It is easy to see that the average current consumption in deep-sleep mode is not a function of the length of the inactivity period. On the other hand, average current drawn in the power-off mode is a decreasing function of the inactivity time, as current is drawn only during the boot sequence, while consumption is otherwise almost zero (the PowerNut switch consumes at most 180 μA).

Figure 5 presents an alternative view of the trade-off by presenting the total current drawn from the power source, again, as a function of the inactivity time. The energy costs corresponding to the power-off mode are now constant, determined only by the energy used to boot the device, while the values for the deep-sleep mode increase linearly with the inactivity time. This makes the power-off mode preferable when low duty cycles are desired. Specifically, the Stargate can stay in deep-sleep mode up to 16.5 minutes when no cards are attached to it (point C), after which it is preferable to use the power-off solution. While this case is not realistic we include

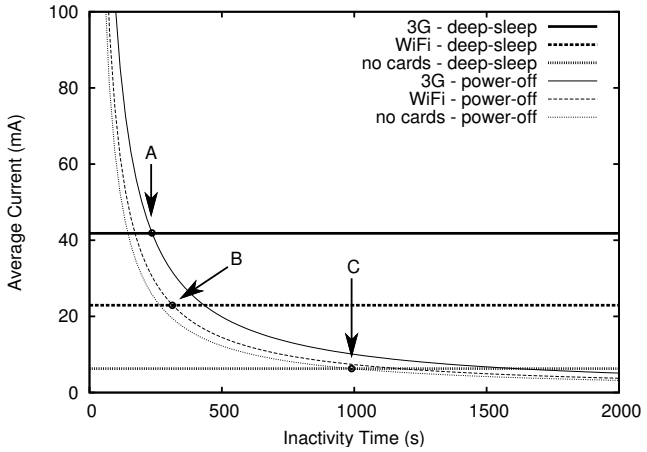

Fig. 4. Average current drawn using different radios as a function of the inactivity time.

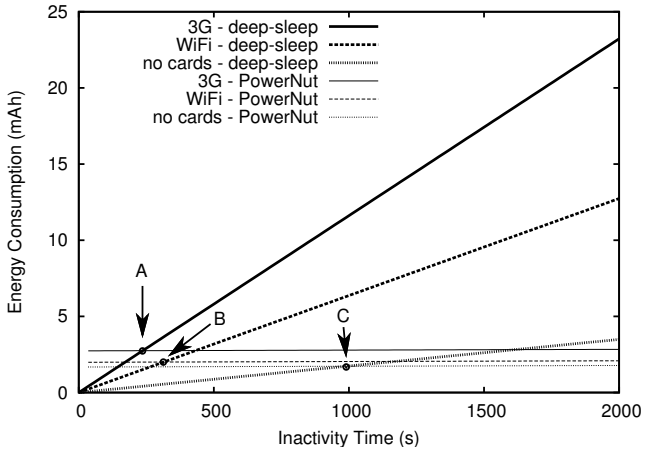

Fig. 5. Total energy consumed using different radios as a function of the inactivity time.

it as a model of an ideal PCMCIA/CF card that consumes no energy when not in use. When the WiFi card is used, putting the device in deep-sleep mode consumes less power only when the inactivity time is less than 314 seconds (point B), while for the 3G card the time is approximately 236 seconds (point A). Note that the inactivity time must be greater than 35 seconds in the power-off mode, as this is the device boot time.

B. Energy Efficiency of Long-haul Radios

To answer the second question, we measure the amount of energy consumed during the transfer of files with sizes ranging from 100 KB to 1 MB, calculated as the product of current drawn multiplied by time required to perform the transfer. We consider the scenario in which the gateway is required to wake-up and transfer a file. Thus, the energy expenditure includes the energy used during the boot process when the power-off mode is used.

The total transfer time consists of the actual transfer time plus the time required to activate the interface and establish the network connection. That is, the time necessary to setup a PPP dial-up connection for the 3G card (which, in our experiments ranged from a few seconds to tens of seconds) or the time to acquire an IP address using DHCP, for the WiFi card ($<$ 1 sec.). Figure 6 shows the total energy consumption required in each case, averaged over five runs. In addition, Table III presents the average transfer time and the relative energy efficiency of the two radios. Our results suggest that the system consumes much less energy using the WiFi card (about 1.2 mAh in deep-sleep mode and 3.8 mAh in power-off mode for a 1 MB file) compared to using the 3G card (about 16 mAh in both modes).

The difference between WiFi and 3G is visible in both deep-sleep and power-off modes. This is because setting up the PPP connection is considerably slower compared to acquiring an IP with DHCP, but also because the 3G cellular connection is significantly slower than 802.11b. Concentrating on the results from the 3G radio, the difference between the deep-sleep and

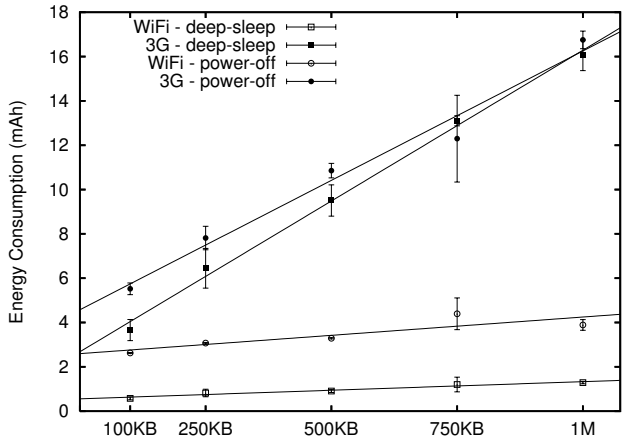

Fig. 6. Average energy consumption for different file sizes, using the WiFi and 3G radios.

the power-off modes decreases as the file size increases, to the point that for a 1 MB file, there is, on average, almost no difference (173 seconds and 175 seconds respectively). The reason is that the time required for the boot process is combined with larger and highly variable time of the PPP connection establishment (the standard deviation of the overall time is 15 seconds). On the other hand, the difference between the deep-sleep and power-off modes persists across all data sizes for the WiFi radio: the total time is 14 seconds on average when the deep-sleep mode is used and 53 seconds in the power-off mode. Nonetheless, because we are interested in low duty cycles (*i.e.*, long inactivity times), as the results from the previous section indicate, powering off the gateway is more efficient than using the deep-sleep mode even when download times are included.

Figure 7 illustrates the current consumption during the transfer of a 1 MB file when using the 3G card in the power-off mode. Section A corresponds to the boot sequence after the

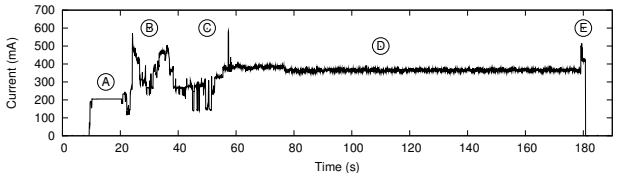

Fig. 7. Current drawn during the transfer of a 1 MB file using the 3G radio in the power-off configuration.

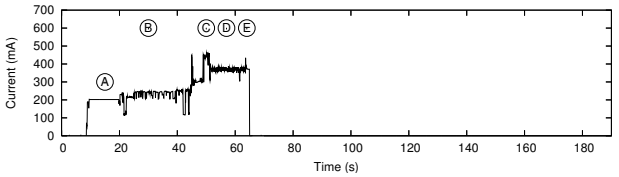

Fig. 8. Current drawn during the transfer of a 1 MB file using the WiFi radio in the power-off configuration.

Stargate is powered on, section B corresponds to the operating system loading, PPP dial-up connection is set in section C, followed in section D by the actual transfer, and a short section E in which the connection is terminated. Figure 8 presents the same stages when the WiFi card is used.

C. Lifetime Estimation

We can now estimate the lifetime of a battery-operated gateway for each of the four configurations (deep-sleep/power-off mode with WiFi/3G radio).

To do so, we consider a WSN of $N = 25$ motes, each with storage capacity of M bytes. Each mote generates measurements at a rate of B bytes/min. We assume that there are no latency requirements in delivering the measurements to the back-end server so the only constraint is to offload the data before the motes overflow their local storage. Therefore, the gateway must retrieve the motes' measurements after each of the motes has collected $\alpha \cdot M$, $(\alpha \leq 1)$ bytes of data. For example, when $\alpha = 0.25$, the gateway must collect 256 KB of data from each mote (6.25 MB in total), approximately every 10 days when the data generation rate $B = 18$ bytes/min and approximately every day if $B = 180$ bytes/min. Table IV provides a list of all the model's parameters.

To estimate the gateway's expected lifetime, we must compute its total energy consumption over time. This consumption is the sum of three factors: **(1)** the energy consumption while the gateway is sleeping (either in deep-sleep or power-off mode), **(2)** the consumption when the gateway is actively collecting mote data, and **(3)** the consumption while the gateway uploads data to the back-end server. We estimate each of these three factors next.

a) Energy consumption in sleep mode.: In deep-sleep, the gateway draws constant current of 23 mA with the WiFi card and almost double the current (42 mA) with the 3G card. On the other hand, when the gateway is turned off the only current drawn is by the power controller ($\sim$180 μA).

b) Energy consumed to retrieve WSN data.: Figure 9 indicates the amount of time required to reliably download the measurements from all of the network's motes, as a function of the per-mote data that the gateway downloads [25]. We derive the energy consumption by multiplying the download time by the current drawn by the gateway when the long-haul interfaced is down (102 mA for WiFi and 124 mA for 3G). Note that connectivity to the WSN is provided by a mote that is directly connected to the gateway and is power independently.

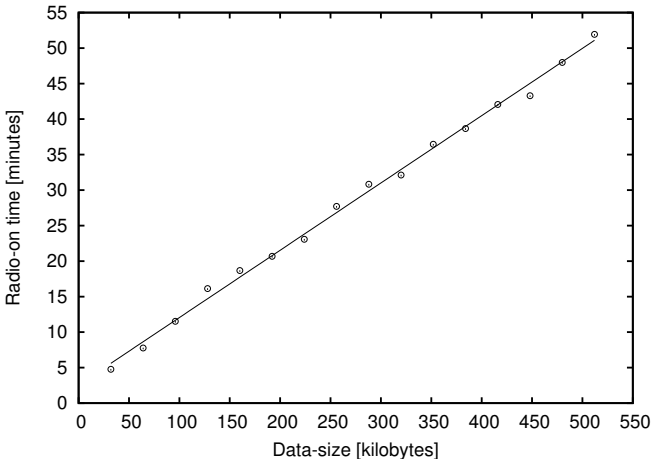

Fig. 9. Total time the gateway is active downloading data as a function of the per-mote download size for a network with 25 nodes. These download times were reported in [25].

c) Energy consumed to upload data.: Finally, as Figure 6 indicates, this amount is a linear function of the total amount of data uploaded to the back-end server. Note that in the power-off configuration this energy includes the cost of booting the gateway (cf. Sec.IV-B).

We combine the three factors to derive the results shown in Figures 10 and 11. These figures show the cumulative energy consumption as a function of time in deep-sleep and power-off modes for both long-haul radios, when the gateway downloads data every one and every ten days respectively. Based on these consumption rates, we estimate gateway lifetimes in two scenarios. First when the gateway is powered by standard AA-cell lithium batteries with a capacity of 3,000 mAh and second when the gateway is powered by special Lithium Thionyl Chloride batteries with 19,000 mAh of capacity [30]. Table V presents the expected gateway lifetimes in both scenarios.

It is evident that considerable energy is consumed in deep-sleep mode, preventing the gateway from completing even a single cycle of operation with the 3,000 mAh battery, when data is collected every ten days. On the other hand, when the gateway operates in power-off mode and motes generate 18 Bytes/min, it can last for 293 days with the WiFi radio and 171 days with the 3G radio using a 3,000 mAh battery. Furthermore, the 19,000 mAh battery extends the gateway's lifetime to 5 years and 2.96 years, respectively. Even at the higher data generation rate of 180 Bytes/min, the gateway

	Deep-sleep						Power-off					
	WiFi			3G			WiFi			3G		
Size	mAh	s	mAh/MB	mAh	s	mAh/MB	mAh	s	mAh/MB	mAh	s	mAh/MB
100KB	0.58	7.3	5.91	3.66	40.7	37.48	2.63	38.7	26.89	5.52	62.9	56.53
250KB	0.82	9.9	3.36	6.45	68.9	26.41	3.07	44.7	12.58	7.81	87.1	32.00
500KB	0.91	10.9	1.86	9.51	106.9	19.47	3.29	46.7	6.73	10.85	116.8	22.23
750KB	1.20	14.0	1.64	13.10	137.9	17.89	4.39	58.8	6.00	12.3	131.9	16.79
1MB	1.29	14.4	1.32	16.06	173.2	16.44	3.89	53.1	3.99	16.75	174.5	17.16

TABLE III
AMOUNT OF TIME AND ENERGY CONSUMPTION FOR DIFFERENT FILE SIZES, USING THE WIFI AND 3G RADIOS.

Parameter	Description	Value
N	number of WSN nodes	25
α	storage threshold	0.25
M	mote storage capacity	1 MB
B_1	data generation rate	18 bytes/min
B_2	data generation rate	180 bytes/min

TABLE IV
PARAMETERS USED IN THE LIFETIME ESTIMATION MODEL.

Setup	3,000 mAh		19,000 mAh	
	days	data	days	data
data collected once every 10 days				
WiFi - deep-sleep	5.41	0	34.20	18.75 MB
3G - deep-sleep	2.95	0	18.75	6.25 MB
WiFi - power-off	293.16	175 M	1860.79	1143.75 MB
3G - power-off	171.83	100 M	1071.95	656.25 MB
data collected once every 1 day				
WiFi - deep-sleep	5	25 M	31.54	193.75 MB
3G - deep-sleep	2.66	12.50 M	16.58	100 MB
WiFi - power-off	47.45	287.50 M	300.29	1850 MB
3G - power-off	22.16	131.25 M	137.45	843.75 MB

TABLE V
LIFETIME ESTIMATES UNDER DIFFERENT OPERATING MODES WITH THE WIFI AND 3G RADIOS.

can operate uninterrupted for 137-300 days when the larger battery is used. These results are very encouraging because they indicate that powering data gathering gateways through small batteries is indeed feasible.

V. ALTERNATIVES

Even though the Stargate is a viable gateway platform, it has a number of shortcomings. The most important one is high energy consumption: even in its deepest sleep mode the device consumes enough power to preclude long deployments using batteries. Other issues include long boot time, lack of hardware schematics, and high cost.

Considering these limitations, we present alternative solutions in the remainder of this section. Most of them are less expensive but also less powerful than the Stargate. At the same time their resources should be adequate for all but the most demanding gateway applications.

A. Hardware Platforms

The ideal gateway should have enough processing power and memory to handle the tasks described in Sections III-B and III-C, persistent storage to buffer sensor measurements, and interfaces for WSN and long-haul radios. At the same time, it should consume very little energy in all operating modes (active, idle, deep-sleep), provide open programming tools, and be inexpensive.

Considering these desirable characteristics, Table VI provides a summary of the various single board computers (SBC) we reviewed. The major drawback of the Flashlite [16] is its lack of any low-power mode—we measured an average current drawn of 170 mA—therefore an external power control module (such as the PowerNut) is necessary. On the positive side, the device boots quickly (in 1-2 seconds), the architecture is very simple and well understood, and the built-in RS-232 can be used to connect to various long-haul radios. The Mod5270 [17] provides frequency scaling and its processor is supported by GCC. On the other hand it includes an unnecessary Ethernet connector and lacks board voltage regulation. The RCM4510W RabbitCore [18] offered by Rabbit Semiconductor provides a wide range of power management features (CPU frequency scaling, power regulation, amount of on-board memory and flash) and offers a battery-backed real time clock and an on-board 802.15.4 radio. Unfortunately, no free compilers are available for this platform.

Moving on to more powerful (and expensive) platforms, the Wildfire 5282 is a SBC based on the MCF5282 [15], a more powerful Freescale Coldfire microprocessor. It includes a socket for SD/MMC memory cards and a battery-backed RTC that can be used to wake the system from deep sleep mode. The Imote2 [5] from Crossbow offers low deep sleep consumption and ample resources, it is however the most expensive option.

The most attractive gateway platforms among the ones we reviewed, come from Gumstix Inc. [14]. Their SBCs use the PXA255 and PXA270 processors and some have built-in Bluetooth radios and MMC sockets. Moreover, they support a wide range of expansion boards that provide serial ports, USB host ports, WiFi, GPS, and expose the CPU pins. While figures about the current drawn in deep sleep mode are not publicly available, the current draw in idle mode is 10-20 mA @ 4.5V. The up-to-date support for Linux and a thriving community that has embraced the Gumstix products

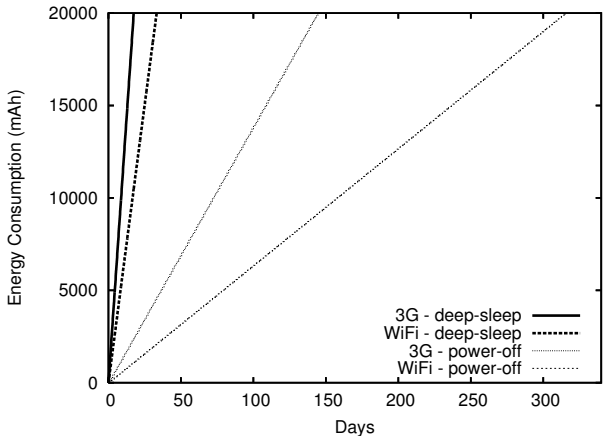

Fig. 10. Lifetime estimation for a 19,000 mAh battery. Data is collected every 1 day.

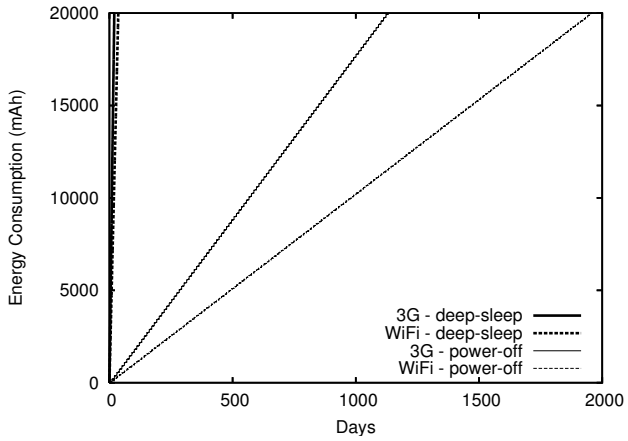

Fig. 11. Lifetime estimation for a 19,000 mAh battery. Data is collected every 10 days.

for robotics and networking applications make this platform extremely attractive.

Considering the similarities between the Gumstix devices and the Stargate, long boot times can still be an issue. On the other hand, we believe boot times can be reduced by customizing the Linux kernel. We plan to implement the gateway design described in this paper to the Gumstix platform in the near future.

B. Operating Systems

The fact that the Stargate runs Linux simplifies the system designer's work. Unfortunately, most of the alternative hardware platforms do not have the resources to run Linux. On the other hand, considering the simplicity of tasks that a data gathering gateway has to perform, it is debatable whether anything other than a proper set of drivers, a simple file-system, and support for concurrency is necessary. We review a number of alternatives next.

μC/OS-II [23] is an RTOS specifically designed for high portability. It is written in C and provides features such as preemptive scheduling and multitasking but it lacks dynamic code loading. The code is publicly available and it is free for educational use. eCOS [9] is another open source RTOS which offers a high degree of configurability for maximum performance. Unfortunately it does not support many of the platforms we mentioned. Finally, FreeRTOS [10] is another Open-Source RTOS. It has features very similar to μC/OS-II, but is less mature and it does not support as many platforms as μC/OS-II. In the commercial realm VxWorks [19] is a mature and feature-rich solution.

At this point, it is unclear whether a customized version of the Linux kernel can reach the level of energy efficiency that some of the 'leaner' operating systems described above achieve. We plan to quantitatively measure the relative merits of these alternatives as part of our future work.

VI. Discussion

We have shown that completely disconnecting a gateway from power when inactive is a promising approach for deploying efficient, battery-operated gateways used in data gathering applications. Some of the inefficiencies observed in our Stargate-based prototype are the consequence of the long boot sequence and could be removed by using a "leaner" operating system on the gateway. We also found that cellular modems should be used only when no other alternatives for long-haul connectivity exist, as they are considerably less efficient than WiFi radios. On the positive side, we estimated that even when cellular radios are used, gateways powered by high capacity D-cell batteries that use the power-off mode can last from hundred of days up to multiple years, depending on the amount of data the WSN generates.

Based on our experience with this prototype we argue that the research community should investigate alternatives to the current high-cost, high-complexity, power-hungry gateways deployed in testbeds today. At one extreme, motes coupled with long-haul radios connected through a serial interface could act as gateways. An equally important direction worthy of further investigation is the use of simpler, low-power SBCs such as those presented in Section V. We argue that a proper abstraction layer, similar to what TinyOS has provided for motes, can enable the use of a wide range of such devices as WSN gateways.

Acknowledgments

We would like to thank our anonymous reviews for their suggestions to improve the paper. Moreover, we extend our gratitude to Robert Szewczyk for his insightful comments and guidance. Andreas Terzis and Răzvan Musăloiu-E. are supported in part by NSF grant CNS-0546648 and a grant from the Johns Hopkins University Applied Physics Lab. Any opinions, finding, conclusions or recommendations expressed in this publication are those of the authors and do not represent the policy or position of the National Science Foundation.

	Flashlite 168	Mod5270	RCM4510W RabbitCore	Wildfire 5282	Verdex XM4	Imote2	Stargate
Architecture	16-bit (Intel 186)	32-bit (Coldfire)	8-bit (Rabbit)	32-bit (Coldfire)	32-bit (ARM)	32-bit (ARM)	32-bit (ARM)
MCU	R8822	MCF5270	Rabbit 4000	MCF5282	PXA270	PXA271	PXA255
Max. frequency	33 MHz	147 MHZ	29.49 MHz	64 MHz	400 MHz	416 MHz	400 MHz
RAM	512KB	2MB	512KB	6MB	64MB	256KB+32MB	64MB
Flash	512KB	512KB	512KB	512KB	16MB	32MB	32MB
Persistent storage	DiskOnChip (up to 48MB)	SD/MMC via SPI	SD/MMC via SPI	SD/MMC	CF/MMC/SD via ext. boards	SD/MMC	CF
Low Power Mode	No	Yes	Yes (freq. scaling down to 2 kHz)	Yes (wake-up using RTC)	Yes	Yes	Yes
Features	2×RS-323 40 GPIO	3×UART SPI, I2C, Ethernet	6×UART 4×SPI (shared) RTC, 802.15.4	3×RS-323 Ethernet, RTC	WiFi, GPS via extension boards	3×UART, 2×SPI I2C, SDIO, USB I2S, AC97	2×UART, I2C PCMCIA, CF
Compiler	several free	GCC	Dynamic C	GCC	GCC	GCC	GCC
OS	XDOS			Linux	Linux	Linux, TinyOS	Linux
Price	$69	$79	$89	$199	$129	>$400	$425
Consumption	170mA@5V	130mA@3.3V	20μA-150mA@3.3V	-	10-220mA@4.5V	390μA-66mA@4.5V	6-300mA@5V
Manufacturer	JK microsystems	Netburner	Rabbit Semiconductors	Intec Automation	Gumstix	CrossBow	CrossBow

TABLE VI
COMPARISON OF ALTERNATIVE SINGLE BOARD COMPUTERS (SBCS)

REFERENCES

[1] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *ACM/Kluwer Mobile Networks Applications (MONET), Special Issue on Wireless Sensor Networks*, 10(4):563–579, August 2005.

[2] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN)*, 2007.

[3] Crossbow Corporation. MICAz Specifications. Available at http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf.

[4] Crossbow Corporation. Stargate Gateway (SPB400). Available at: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf, 2004.

[5] Crossbow Inc. Imote2. Available at: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf, 2007.

[6] Deluge: TinyOS Network Programming. Available at http://www.cs.berkeley.edu/~jwhui/research/projects/deluge/.

[7] Adam Dunkels, Bjŏrn Grŏnvall, and Thiemo Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors 2004 (IEEE EmNetS-I)*, November 2004.

[8] Prabal Dutta, David Culler, and Scott Shenker. Procrastination Might Lead to a Longer and More Useful Life. In *Proceedings of HotNets-VI*, November 2007.

[9] eCOS. Available at: http://ecos.sourceware.org/.

[10] FreeRTOS. Available at: http://www.freertos.org/.

[11] Omprakash Gnawali, Ben Greenstein, Ki-Young Jang, August Joki, Jeongyeup Paek, Marcos Vieira, Deborah Estrin, Ramesh Govindan, and Eddie Kohler. The TENET Architecture for Tiered Sensor Networks. In *Proceedings of the ACM Sensys Conference*, November 2006.

[12] Ben Greenstein and Philip Levis. TEP 113: Serial Communication. Available at: http://www.tinyos.net/tinyos-2.x/doc/html/tep113.html, June 2006.

[13] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System Architecture Directions for Networked Sensors. In *Proceedings of ASPLOS 2000*, November 2000.

[14] Gumstix Inc. Gumstix - way small computing. Available from: http://gumstix.com/, 2007.

[15] Intec Automation Inc. Wildfire 5282. Available from: http://www.steroidmicros.com/micros/micro.aspx?ID=WF-5282, 2004.

[16] JK Microsystems Inc. Flashlite 186, 2004.

[17] Netburner Inc. Mod5270. Available from: http://www.netburner.com/products/core_modules/mod5270.html, 2007.

[18] Rabbit Semiconductor Inc. RCM4510W RabbitCore. Available from: http://www.rabbitsemiconductor.com/products/RCM4500W/, 2007.

[19] Wind River Inc. VxWorks 6.x - real time operating system.

[20] JK microsystems Inc. PowerNut embedded power controller. Available at: http://www.jkmicro.com/products/powernut.html, 2007.

[21] Chieh-Jan Mike Liang, Andreas Terzis, and Razvan Musaloiu-E. Typhoon: A Reliable Data Dissemination Protocol for Wireless Sensor Networks. In *EWSN '2008 5th European Workshop on Wireless Sensor Networks*, 2008.

[22] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of 2002 ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[23] Micrium. uC/OS-II - the real-time kernel. Available at: http://www.micrium.com.

[24] Moteiv Corporation. Tmote Sky. Available at http://www.moteiv.com/products/tmotesky.php.

[25] Răzvan Musăloiu-E., Mike Liang, and Andreas Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. Technical report, Johns Hopkins University, Hopkins InterNetwork Research Group, November 2007.

[26] Răzvan Musăloiu-E., Andreas Terzis, Katalin Szlavecz, Alex Szalay, Joshua Cogan, and Jim Gray. Life Under your Feet: A Wireless Sensor Network for Soil Ecology. In *Proceedings of the 3rd EmNets Workshop*, May 2006.

[27] Jeongyeup Paek and Ramesh Govindan. RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 305–319, New York, NY, USA, 2007. ACM.

[28] L. Selavo, A. Wood, Q. Cao, A. Srinivasan, H. Liu, T. Sookoor, and J. Stankovic. Luster: Wireless Sensor Network for Environmental Research. In *Proceedings of ACM Sensys*, November 2007.

[29] Ivan Stoianov, Lama Nachman, Sam Madden, and Timur Tokmouline. Pipenet: A Wireless Sensor Network for Pipeline Monitoring. In *Proceedings of the Information Processing in Sensor Networks Conference (IPSN)*, April 2007.

[30] Tadiran. Lithium Cells - Extended Temperature Series. Available at: http://www.tadiran.com/tlh59c.php.

[31] Gilman Tolle, Joseph Polastre, Robert Szewczyk, Neil Turner, Kevin Tu, Phil Buonadonna, Stephen Burgess, David Gay, Wei Hong, Todd Dawson, and David Culler. A Macroscope in the Redwoods. In *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2005.

[32] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A Wireless Sensor Network for Structural Monitoring. In *Proceedings of SenSys 2004*, November 2004.