Energy Consumption and Conservation in Mobile Peer-to-Peer Systems

Selim Gurun
Computer Science
UC Santa Barbara
gurun@cs.ucsb.edu

Priya Nagpurkar
Computer Science
UC Santa Barbara
priya@cs.ucsb.edu

Ben Y. Zhao
Computer Science
UC Santa Barbara
ravenben@cs.ucsb.edu

ABSTRACT

Today's mobile devices are growing in number and computational resources. Devices capable of storing gigabytes of digital content are becoming ubiquitous, making them an ideal platform for peer-to-peer content delivery and sharing. However, the always-on communication patterns of P2P networks is not a natural fit for energy-constrained mobile devices. In this paper, we perform a detailed study of energy consumption of a structured P2P overlay on a PDA device. Using actual energy measurements, we present energy consumption results for different type of operations in P2P overlays. Based on these observations, we implement an approach to improve energy conservation on P2P protocols and show some promising preliminary results.

Categories and Subject Descriptors

C.2.2 [**Computer-communication Networks**]: Network Protocols; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement

Keywords

Peer-to-peer, Structured overlays, DHTs, Mobility

1. INTRODUCTION

As they scale down in form factor and grow in network connectivity, today's networked devices are taking us closer to the goal of ubiquitous computing [21]. Virtually all mobile devices today, laptops, handheld PDAs, smart-phones, even MP3 players [17] have some type of wireless connectivity. Each device is capable of storing Megabytes or Gigabytes of digital content. Because of computational and energy constraints, however, these devices have been limited to data access and communication through centralized servers or proxies, greatly limiting their mobility and application access to network resources.

MobiShare'06, September 25, 2006, Los Angeles, California, USA.

These wireless devices provide an attractive platform for deploying wireless peer-to-peer (P2P) applications such as streaming video over wireless, content-sharing and storage, and real-time event notification. Recent advances in peer-to-peer overlays [19, 14, 23] led to the development of Internet-scale distributed applications such as distributed storage [13], scalable multicast [15], and scalable event notification [16]. These structured overlays have been shown to scale to millions of hosts while maintaining low per-node state and low maintenance traffic [2].

While hardware advances have led to improved computational and storage resources, these devices are still severely constrained in energy. Peer-to-peer applications are likely to exacerbate this energy shortage. Unlike existing client-server applications, where the client can perform disconnected operations and utilize intermittent connectivity, P2P applications assume that peers are always on and available to route messages or satisfy queries for data. To the best of our knowledge, no work has studied the impact of this always-on communication model on energy-constrained mobile devices.

In this paper, we study the feasibility of deploying structured overlays on mobile devices from the perspective of energy consumption. The paper makes three contributions. First, we port a second-generation P2P protocol (Chimera) over to an embedded computing platform similar to the HP iPAQ, and quantify its energy consumption using a high-precision measurement platform. We describe our experimental methodology, features and constraints of our target platform, and our power model in Section 3. Second, we measure the energy overhead of P2P applications by comparing a P2P chat program against a client-server clone of MSN Messenger, and present the results in Section 4. Finally, we implement an approach to improve energy conservation for peer-to-peer protocols, and present some promising preliminary results in Section 5.

2. STRUCTURED P2P AND CHIMERA

A structured peer-to-peer overlay is an application-level network connecting any number of nodes, each representing an instance of an overlay participant. Nodes are assigned nodeIds uniformly at random from a large identifier space. Nodes can generate their nodeIds by applying a secure one way hashing function such as SHA-1 to either its IP address or a secure public key. Application-specific objects are assigned unique identifiers called keys from the same space.

The overlay dynamically maps each key to a unique live node, called its *root node*. While a key's root can change with network membership, at any given time in a consistent network, a single node is responsible for each key. To deliver a message based on a key to its root node (*key-based routing* [5]), each node forwards the message using a locally maintained routing table of overlay links.

Existing protocols share several important characteristics. To

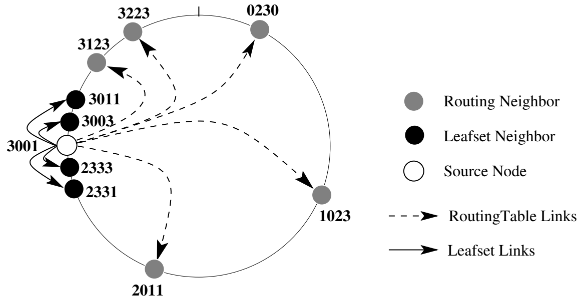

Figure 1: ***Prefix routing.*** **Each overlay node, e.g. `3001` keeps a *leafset* of neighbors closeby in the namespace, as well as a routing table filled with well-chosen nodes distributed across the namespace. Names are represented in base 4.**

Figure 2: Our target board. The Stargate is a small sensor network gateway running Linux, and is almost same as an HP iPAQ.

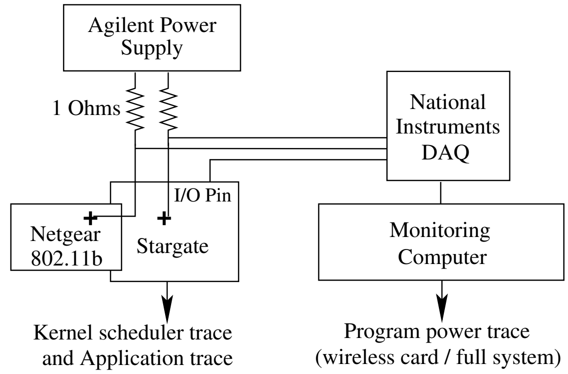

Figure 3: Measurement setup. We place high precision resistors between the power supply and both the PCMCIA and main board power supply connectors. The NI high speed DAQ samples the voltage across the resistors 1000 times per second. We use a third connection between the DAQ and the Stargate's output port to correlate power data with program profile.

support a network of size N, most protocols require per-node routing state that scales as $O(\log N)$ and provide worst case $O(\log N)$ overlay hops between any two nodes. Routing proceeds by forwarding the message incrementally closer in the namespace to the desired key, using routing table links for "long hops" through the namespace.

Chimera [4] is a light-weight prefix-based structured overlay protocol implemented in C as an application library. It builds routing tables using proximity routing techniques from Tapestry [23] and uses per-node *leafsets* for stability across failures like Pastry [14]. A node's leafset is a small number of nodes closest to it in the namespace. Figure 1 shows an example of routing state for a single Chimera node (node `3001`).

3. EXPERIMENTAL METHODOLOGY

Our goal is to perform detailed energy measurements of a handheld device while running a peer-to-peer protocol, and correlate detailed fluctuations in energy with specific function calls executed on the device. Performing detailed energy characterization was a significant challenge. In this section, we give a detailed description of our experimental setup and describe our measurement methodology on the Stargate embedded board.

3.1 Setup and Trace Generation

As our handheld platform, we use the Stargate embedded computer, a popular low-power sensor network gateway nearly identical in architecture to the Hewlett-Packard iPAQ. The Stargate, shown in Figure 2, has no display. Our experiments use a Stargate device with a 400 MHz Intel XScale processor, 64 MB of RAM, a wireless Netgear 802.11b interface, and a 256MB Compact Flash card. The Stargate runs Linux Kernel 2.4.19 as its operating system.

Figure 3 shows our power measurement setup. In addition to the Stargate embedded computer, our measurement setup includes a National Instruments Data Acquisition Board (DAQ) and an Agilent variable power supply. The NI high speed DAQ is an accurate power profiling tool, while the Agilent E3648A power supply is used to prevent fluctuations in DC current and voltage level. We use our measurement platform to generate three separate traces, a *power trace* from the DAQ, an *application trace* to record application profile information, and a *scheduler trace* to monitor kernel scheduling decisions.

The Stargate provides debugging pins and input/output ports that are easy to interface. These pins expose the power supply of Stargate and the wireless card, which we use to monitor their energy consumption. During power profiling, we use one A/D channel of the DAQ to sample the voltage across a resistor that is connected in series to the main power supply and the Stargate. Similarly, we use another A/D channel to sample the wireless card voltage. The DAQ samples each channel concurrently at 1000 times per second, generating a *power trace* which we download for post analysis.

One challenge in our measurement design lies in correlating power data with the application runtime events on the Stargate. To address this, we make a third connection between the DAQ device and a Stargate output interface (GPIO Port 26). To access the GPIO port, we use its control file, */proc/platx/gpio/GPCTL*, a special file that is directly mapped to a kernel device driver. We use this file in non-buffered mode to send set/clear commands to the GPIO port.

We built a user library to control access to GPIO port and record profile information. The library includes simple functions for controlling the monitor. The profiled application calls `monitor_init` to open the GPIO control file, resets the port and allocate a log buffer to store application power profile. The `monitor_start()` function signals the start of profiling to the DAQ and records a timestamp. `monitor_stop` function clears the line. Both start and stop functions take a character pointer as a parameter, which acts as an event id for the user to distinguish consecutive start/stop events in the log trace. The context of the character pointer is transparent to the user library. Finally, `monitor_end` resets the GPIO port and writes the application log buffer to a file. We call this data as *application trace*.

We continuously log kernel scheduling decisions to capture energy behavior of operating system and application threads. Since

State	Power
Transmit	≤ 1.75 W
Receive	≤ 1.25 W
Power Save	0.4 W

Table 1: Wireless Card Specifications.

we use a general-purpose operating system, many daemons, which are lightweight tasks, continue to run at the background for regular maintenance, even when the system is totally idle. Furthermore, The threads in our overlay or application may be interrupted or blocked during system calls. To accurately match up energy consumption with specific threads and functions, we need to know when threads block and when context switches occur. The *scheduler traces* that we capture provides this information.

To capture the scheduler trace without interfering with operating execution execution, we developed a light-weight (less than %2 overhead) device driver. This device driver records all scheduling decisions into a ring that we allocate in kernel space. The ring has 16K entries, enough to capture all scheduler decisions for our monitoring period. On each scheduling decision, we record the thread identifier, and the scheduling decision time in microseconds resolution. A device file, */proc/pmulog*, acts as an interface to this buffer. When the user reads this file, the device driver displays the latest ring contents in ASCII format. After each experimental run, we use this driver to redirect */proc/pmulog* into a file for offline analysis.

3.2 Offline Analysis

Once the data collection is over, we combine the application, scheduler and power trace files to compute energy consumption per task and per function. The application trace and the scheduler trace files both include timestamps generated by the Stargate device accurate to the level of microseconds. These two traces are easily merged in time sequential order. The power trace however, is timestamped using the DAQ clock at the granularity of milliseconds. We correlate the power trace and the application/kernel traces by using trigger events across the GPIO port to synchronize timestamp values between the Stargate and the DAQ.

As we described, the DAQ samples the voltage on the resistor. To compute the power, we first compute the current (*i.e.* V/R) and then interpolate the instantaneous voltage readings. The energy consumption from t_1 to t_2 is,

$$E(\text{in Joules}) = \sum_{t=t_1}^{t_2} V_t \times \frac{(5.0 - V_t)}{1 \text{ Ohms}} \times 0.001 \text{ seconds}$$

In this equation, the first term gives voltage measurements at time t, the second term gives the current on resistor (in series to the circuit), and the last term is the time (1 millisecond).

4. MEASUREMENTS AND RESULTS

In this section, we present our measurement results in detail. We take two orthogonal approaches to measuring energy consumption. First, we measure Chimera's power consumption under different operating scenarios to try to isolate different Chimera components. Second, we compare overall energy consumption running a decentralized chat client on Chimera against a traditional client-server implementation.

4.1 Overlay Scenarios

Our measurement includes three different application scenarios for the P2P overlay. First, in the *Control Overhead* experiment,

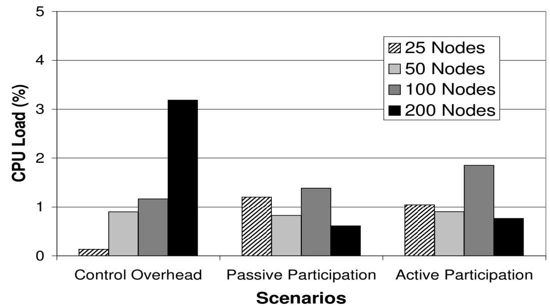

Figure 4: Chimera CPU Load grouped by scenario. Our results show Chimera to be exceptionally light weight. The CPU load introduced was less than 4% in all cases.

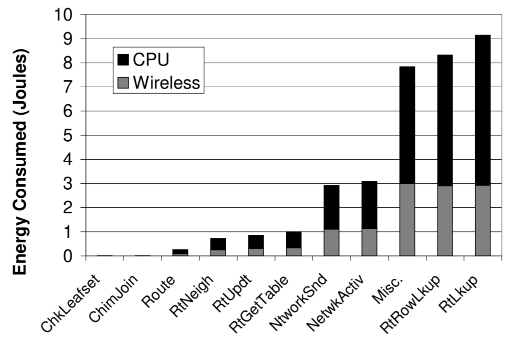

Figure 5: Detailed Energy Consumption of Chimera (Control Overhead scenario, 200 nodes). We find that most energy is used for functions related to routing updates (RtLkup and RtRowLkup).

we measure Chimera's energy costs while joining and maintaining a structured overlay without any application traffic. In the second *Passive Participation* scenario, we try to isolate the energy overhead incurred by the mobile node as a participating peer in the overlay. Our node does not initiate any application activity and only forwards traffic for other peer who are exchanging chat messages. Finally, in our *Active Participation* scenario, the mobile node actively sends and receives chat messages [1].

Throughout these experiments, we vary the size of the total overlay network size to observe its impact on local energy consumption. We ran each scenario on networks of size 25, 50, 100, and 200. For each experiment, we run all other Chimera overlay nodes on two laptop computers connected to our Stargate via wireless 802.11b connections in ad hoc mode. Nodes are spread across the two laptops evenly, with each node running a complete instance of Chimera on a separate port. Node arrival is simulated using the Poisson process ($\lambda = 0.5$) [1]. Each node is created with a predefined session time, generated using the Pareto distribution (shape 0.78, scale 30), after which the node leaves. Each node randomly chooses another node to message after a time interval generated using the Pareto distribution (to model inter-message arrival time) [6].

We first discuss energy consumption rates of the wireless interface, since it is generally regarded as a significant consumer of energy in handheld devices. Previous research [18] shows communication energy consumption is a linear function of the time the wireless card stays in transfer and receive states. According to specifi-

[1]We implemented a generic distributed peer-to-peer chat protocol on Chimera where nodes forward messages based on the destination host's nodeId.

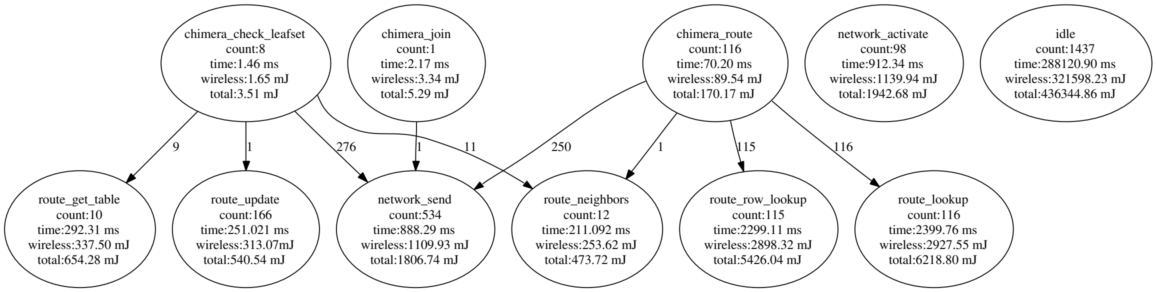

Figure 6: **Chimera call graph and energy consumption per thread (for scenario Control-Overhead-200).**

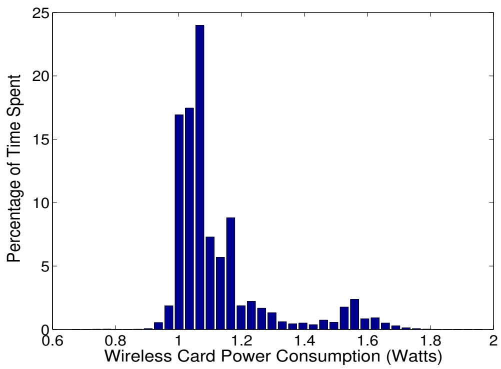

Figure 7: **Histogram of Wireless Card Power Consumption.**

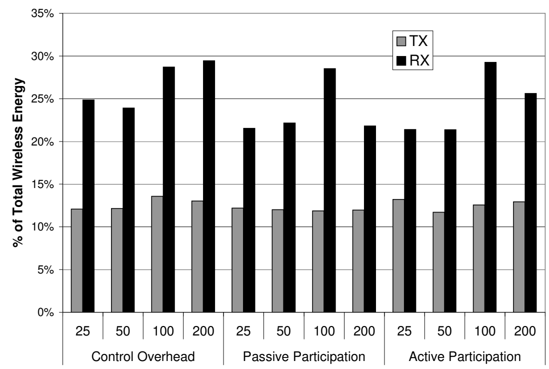

Figure 8: **Wireless Energy Consumption Breakdown. Energy for receiving data increases with network size, but no similar pattern visible for energy spent on data transmissions.**

cations, our card's maximum power consumption in power savings mode is 0.4 watts, 1.25 watts in receive mode and 1.75 watts in transmit mode [12]. We summarize these values in Table 1.

It is important to monitor CPU idleness to identify the CPU load and power consumption that is due to Chimera. The idle task is a special task that the Linux scheduler switches to when no runnable processes exist in the task queue. In Stargate, the idle task puts the CPU in a low power mode by turning off clocks to most of the CPU components. Then the CPU stays in low-power mode until an external interrupt or a timer wakes it up, providing significant energy savings. By monitoring the idle task, we measure the exact CPU load and energy consumption introduced by Chimera.

Since Chimera is the only application running on Stargate, we estimate its CPU load as the percentage of time not spent on the idle process. The results plotted in Figure 4 show that Chimera is extremely light-weight, on average only utilizing 1-2% of the CPU time in our experiments. Among all scenarios, Chimera generated the highest load (3.8%) when the node was performing node join into a network of 200 nodes. We breakdown the energy consumption into detailed components in Figure 5, where the columns show energy consumption of most CPU intensive functions, with each function's energy broken down into communication and computation costs. The most significant energy consumers are route_lookup (RtLkup) and route_row_lookup (RtRowLkup), accounting approximately a quarter of all energy that is consumed by Chimera [2].

4.2 Wireless Energy Consumption

To characterize and better understand the energy cost of communication, we analyze the traces that we collect for the wireless card. Figure 7 shows the histogram of wireless card power consumption per time samples for one experiment (Control Overhead scenario, with 100 nodes). We see three peaks in the histogram: one around 1.06 watts, one around 1.16 watts and one around 1.55 watts. The peaks correspond to the maximum energy expended in the idle, receive, and transmit states specified by the manufacturer (note the histogram also indicates measurement noise distributed normally around the peaks). The energy consumption increases only slightly when the card starts receiving data, this is because the RF radio must be on even during the idle state to listen for incoming packets. Note that we do not see any peaks around 0.4 watts, the expected energy level for power savings mode. This is because the wireless card–device driver combination that we use seems unable to enable 802.11b power-saving mode in ad-hoc mode which we run our experiments in.

We produce power-consumption histograms for each experiment, and break down the results into transmit and receive energy costs, as shown in Figure 8. To produce this figure, we take the histogram data, and assume that the wireless card is sending packets if the power consumption is greater than 1.36 (i.e. $(1.55 + 1.16)/2$) watts, and receiving packets if the power consumption is between 1.36 and 1.11 watts. We assumed the wireless card to be idle at all other times. The wireless power data shows that the receive energy consumption increases as we add more nodes to our network. For example, increasing node count from 50 to 100, increases the wireless receive energy consumption 5% in the Control Overhead scenario. Overall, approximately 10% of all wireless energy is consumed during data transmission and 25% while receiving data.

Given the low control traffic required by Chimera, we are somewhat surprised by the high amount of energy consumed for data reception. We believe there is one additional contributing factor. The

[2] This measurement was performed using Chimera v1.04 released in March 2006. The current release is v1.10 from June 2006.

Model	Transmit	Receive	Idle
TMSNC	12 (J)	18 (J)	65 (J)
Control (200 nodes)	13 (J)	25 (J)	70 (J)
Passive (200 nodes)	13 (J)	23 (J)	68 (J)
Active (200 nodes)	13 (J)	25 (J)	70 (J)

Table 2: Comparing energy consumed (Joules) of TMSNC against Chimera-chat.

two laptops that we use to run the remaining Chimera nodes communicate amongst themselves via their 802.11b interfaces. Even though the packets that these nodes exchange are not relayed through the Stargate, the wireless card must listen to RTS/CTS packets and decode packet headers to understand where the packets are destined for. Finally, given that the power consumed between idle and receive states are similar, some noise in our power measurements can result in the inclusion of idle states into our result.

4.3 Client/Server versus Peer-to-Peer

In addition to understanding Chimera's own energy consumption patterns, we want to quantify the additional energy costs of running a peer-to-peer application compared to a similar application using a client-server model. In this section, we compare the energy consumption of our Chimera-based Chat application to that of a client-server based chat application.

Text-based MSN Client (TMSNC) [20] is an open-source MSN client for Linux. It is light-weight and suitable for most Linux-based handheld devices. TMSNC and Chimera are significantly different in their software architecture. Chimera has a multi-threaded structure, with one application thread, one network thread and many worker threads. TMSNC, since it is only designed as a client, has a much simple programming structure with only one thread that contains a single, large event handling loop. Both applications are similar in functionality. And while not representative of all applications, comparing these simple applications should give us an idea of minimal energy costs introduced by the peer-to-peer model.

To evaluate TMSNC's energy consumption, we installed two TMSNC client, one to an x86 Linux desktop and the other to the Stargate. We instrumented TMSNC's source code to monitor initialization, maintenance (such as heartbeat and control messages) and message send/receive costs. We also modified each TMSNC node to generate chat messages using the same Pareto distribution as Chimera. TMSNC initializes using login information and begins the session by contacting *messenger.hotmail.com* via https for authentication. It then updates the status of contact list via the server. While this initialization process is costly (5.5 Joules in our setup), when amortized over the session its overall impact is low.

Our measurements show that TMSNC generates an average CPU Load of 1.68%, roughly the same as our Chimera clients. In terms of wireless energy consumption, we find that TMSNC has lower cost. We measured its energy consumption for an observation period of 100 seconds, and plot its values along with those of Chimera for the same time period in Table 2. As we can see, the peer-to-peer Chimera-chat uses only slightly more energy than the client-based TMSNC. While this ratio will likely change for applications that rely on servers for more computation, it does mean any inherent energy costs of maintaining a peer-to-peer overlay is minimal.

4.4 Experiences with Other P2P Overlays

We also tried to port other P2P protocols to our embedded platform. Some implementations had very large code bases, others had dependencies on platform-specific libraries. In addition to Chimera, we tried to port two other protocols. FreePastry [8] is an open source implementation of Pastry [14] written in Java using Java runtime Environment 1.4.2. Simmud [9] is an experimental massively multiplayer game based on server clusters also developed using Java runtime 1.4.2.

While protocols like FreePastry and Simmud obtain improved performance and portability with Java, the Java 1.4 run-time environment is significantly more complex and unsupported by any JVM on our embedded platforms. Compared to C, even the much smaller Blackdown Java VM 1.3.2 on Stargate makes much stronger demands on CPU load, resulting in significantly higher energy consumption and implementations that are much less attractive for mobile systems. While we were able downgrade parts of Simmud to Java 1.3 after reimplementing the references to a few 1.4 classes, we could not do the same for FreePastry, since it made extensive use of the non-blocking I/O operations introduced in JRE 1.4.

5. P2P ENERGY CONSERVATION

We observe that a large percentage ($> 64\%$) of energy consumed by the wireless interface is spent waiting for transmission during idle periods. This is required because peer-to-peer nodes do not know when they will receive messages.

A simple power-saving technique is to adaptively switch the wireless card to a low power state when no communication is observed. An energy-aware P2P protocol can monitor its incoming communications, and allow the interface to "sleep" if no communication is received for some parametrized *idle-time*. The node would wake up after some preset sleep period and transmit buffered outgoing messages and receive incoming messages. Nodes can detect sleeping neighbors via missing acks and buffer messages for some finite period of time.

There are some challenges that need to be addressed. In an ad-hoc network environment, neighboring nodes need to synchronize their wake-up and sleep time to provide enough overlap for some minimal data transfer rate. In addition, the length of each sleep session must be set carefully. Since waking up the wireless card is not instantaneous, sleep periods that are too short may provide minimal energy savings. Meanwhile, periods that are too long may significantly reduce achievable routing throughput, and may cause neighbors to incorrectly assume the node has failed or left the area.

To estimate the potential energy savings possible using such an approach, we perform simulations of nodes running Chimera-chat in each of its three participation scenarios. We run each scenario with different values of *idle-time*, and plot the potential energy savings in Figure 9. In addition, we plot the average queuing delay added to each forwarded message in Figure 10. Clearly, a short idle timeout can result in significant power savings, while adding higher delay to forwarded messages.

6. RELATED WORK

While we are unaware of studies of overlay networks on energy constrained devices, projects have examined how to adapt P2P protocols to efficiently utilize nodes with heterogeneous resources. Accordion [11] allows different peers to choose its number of neighbors, thereby managing the tradeoff between higher maintenance costs and improved routing performance. Hetero-Pastry [2] provided similar capability in the context of Pastry.

Application layer wireless interface management can provide energy savings. Stemm et al. [18] investigated wireless card energy consumption at both the physical and network layers. Kravets et al. showed that a similar policy in transport layer can provide 40% power savings for handhelds [10]. Numerous protocols at the phys-

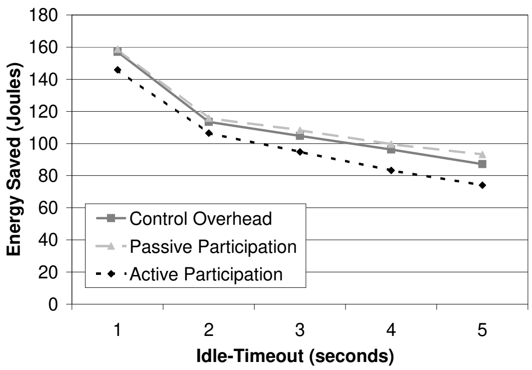

Figure 9: Potential power savings using a idle-sleep energy-aware P2P protocol.

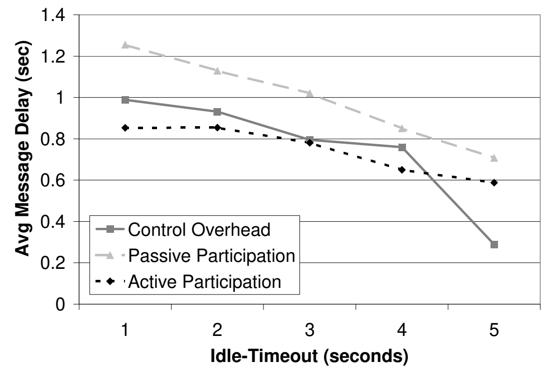

Figure 10: Potential message delay using a idle-sleep energy-aware P2P protocol.

ical layer were suggested for ad-hoc networks, including conserving energy by turning off redundant nodes in SPAN [3] and a similar protocol called GAF [22]. Both SPAN and GAF uses geographical information to choose network coordinators. Finally, the PowerScope project [7] performed power consumption measurements with setups similar to ours.

7. CONCLUSIONS AND ONGOING WORK

To the best of our knowledge, prior work has not examined peer-to-peer protocols from an energy-consumption perspective. This paper takes a first step by performing real energy measurements on a deployed protocol and application. Preliminary results show that it is feasible to deploy light-weight P2P protocols and applications on low-power, embedded devices. In addition, there is significant potential for power savings if overlay messages can be batched and sent periodically.

As future work, we can expand energy tests in scale, and also include more complex applications, including storage-, computation- and bandwidth-intensive applications. In addition, we are studying the problem of designing an energy efficient peer-to-peer protocol. We believe that by providing feedback from P2P protocol to physical layer, it is possible to better manage network interface power states without any adverse affects to other applications that may be executing concurrently. Finally, as a long-term goal, we are considering the use of energy availability in neighbors in the construction of peer-to-peer routing tables.

Acknowledgments

We would like to thank the anonymous reviewers for providing useful comments on this paper. This work is partially supported by DARPA BAA #04-11 and by the NSF under CAREER Award CNS-0546216.

8. REFERENCES

[1] BALACHANDRAN, A., VOELKER, G., BAHL, P., AND RANGAN, P. Characterizing user behavior and network performance in a public wireless lan. In *Proc. of Sigmetrics* (Marina del Rey, CA, June 2002).

[2] CASTRO, M., COSTA, M., AND ROWSTRON, A. Debunking some myths about structured and unstructured overlays. In *Proc. of NSDI* (Boston, MA, May 2005).

[3] CHEN, B., JAMIESON, K., BALAKRISHNAN, H., AND MORRIS, R. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks 8* (Sept. 2002), 481–494.

[4] Chimera structured overlay network. `http://current.cs.ucsb.edu/projects/chimera`.

[5] DABEK, F., ZHAO, B., DRUSCHEL, P., KUBIATOWICZ, J., AND STOICA, I. Towards a common API for structured P2P overlays. In *Proc. of IPTPS* (February 2003).

[6] DEWES, C., WICHMANN, A., AND FELDMANN, A. An analysis of internet chat systems. In *Proc. of Internet Measurement Conference* (Miami, FL, October 2003).

[7] FLINN, J., AND SATYANARAYANAN, M. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proc. of WMCSA* (1999).

[8] Freepastry. `http://freepastry.org`.

[9] KNUTSSON, B., LU, H., XU, W., AND HOPKINS, B. Peer-to-peer support for massively multiplayer games. In *Proc. of IEEE INFOCOM* (Hong Kong, March 2004).

[10] KRAVETS, R., AND KRISHNAN, P. Application-driven power management for mobile communication. *Wireless Networks 6*, 4 (2000), 263–277.

[11] LI, J., STRIBLING, J., MORRIS, R., AND KAASHOEK, M. F. Bandwidth-efficient management of dht routing tables. In *Proc. of NSDI* (Boston, MA, May 2005).

[12] NETGEAR, INC. *User Guide for the Netgear 802.11b wireless compact flash card MA701*, 2003 August.

[13] RHEA, S., ET AL. Pond: The OceanStore prototype. In *Proc. of FAST* (San Francisco, April 2003).

[14] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware* (November 2001).

[15] ROWSTRON, A., KERMARREC, A.-M., DRUSCHEL, P., AND CASTRO, M. SCRIBE: The design of a large-scale event notification infrastructure. In *Proc. of NGC* (November 2001), ACM, pp. 30–43.

[16] SANDLER, D., MISLOVE, A., POST, A., AND DRUSCHEL, P. Sharing web micronews with peer-to-peer event notification. In *Proc. of IPTPS* (Ithaca, NY, February 2005).

[17] SMITH, T. Slim devices adds 802.11g to wireless mp3 player. Channel Register, March 2005. `http://www.slimdevices.com`.

[18] STEMM, M., AND KATZ, R. H. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications E80-B*, 8 (1997), 1125–31.

[19] STOICA, I., ET AL. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM* (August 2001).

[20] TMSNC web page – `http://tmsnc.sourceforge.net`.

[21] WEISER, M. The computer for the twenty-first century. *Scientific American 265*, 3 (September 1991), 94–104.

[22] XU, Y., HEIDEMANN, J., AND ESTRIN, D. Geography-informed energy conservation for ad hoc routing. In *Proc. of MobiCom* (New York, NY, 2001).

[23] ZHAO, B. Y., HUANG, L., RHEA, S. C., STRIBLING, J., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A global-scale overlay for rapid service deployment. *IEEE JSAC 22*, 1 (January 2004).