Practical Exploitation of the Energy-Latency Tradeoff for Sensor Network Broadcast

Matthew J. Miller
Department of Computer Science, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
mjmille2@uiuc.edu

Indranil Gupta
Department of Computer Science
University of Illinois at Urbana-Champaign
indy@cs.uiuc.edu

***Abstract*— As devices become more reliant on battery power, it is essential to design energy efficient protocols. While there is a vast amount of research into power save protocols for unicast traffic, relatively little attention has been given to broadcast traffic. In previous work [1], we proposed Probability-Based Broadcast Forwarding (PBBF) to address broadcast power save by allowing users to select a desired tradeoff between energy consumption, latency, and reliability. In this paper we extend our previous work in two ways. First, we introduce a new parameter that allows a tradeoff between reliability and packet overhead to give users more options. Second, we implement PBBF on the TinyOS platform [2] to evaluate it beyond the analysis and simulation from our previous work. Our evaluation demonstrates the tradeoffs possible using PBBF on sensor hardware.**

I. Introduction

The relatively small improvement in battery energy density recently [3] necessitates the need for energy efficient protocols to control the *rate* at which energy is depleted. To this end, many proposed power save protocols increase the time that a device's radio sleeps while providing acceptable latency and throughput. Work in this domain focuses almost exclusively on unicast traffic. Our previous work on Probability-Based Broadcast Forwarding (PBBF) [1] was the first to explore the energy-latency tradeoff for broadcast traffic. Multihop broadcast is used in many wireless network applications, such as discovering routing paths, sinks querying sensors for data, and distributing code updates throughout the network.

With respect to broadcast, power save protocols generally expose two options to the user. First, if no power save is used, then the broadcast can achieve a relatively low latency, but at the expense of large energy costs to listen for broadcasts. The second option is to use the power save protocol. This option conserves much more energy than the first, but has a high latency that may be unacceptable to some applications.

In previous work [1], we proposed a lightweight protocol to augment existing protocols that allows broadcast propagation to be more energy efficient while still achieving a desired latency. In this paper, we extend that work in two ways:

- *Introduce a parameter to control the reliability-overhead tradeoff:* Previously [1], we proposed two parameters (discussed in Section III) that present tradeoffs in energy consumption, latency, and reliability. By introducing a third parameter (described in Section IV) we allow another tradeoff in reliability and packet overhead.[1]
- *Implement PBBF in TinyOS [2] on top of B-MAC [4]:* Our previous work explored PBBF via analysis and simulation. In this work, we implement the protocol in TinyOS [2] (described in Section V) and evaluate the performance in Section VI. Also, our previous work demonstrated PBBF on top of 802.11 Power Save Mode (PSM) [5]; in this work it is implemented on top of B-MAC [4] to demonstrate PBBF's versatility.

II. Related Work

There have been many power save techniques proposed for unicast traffic [6]. Our work [1] was the first to study power save with broadcast traffic by proposing PBBF (Section III). PBBF uses probability-based forwarding for energy efficiency. While PBBF was the first to propose this for power save, other protocols use probabilistic broadcast forwarding for other reasons. Most notably, Haas et al. [7] designed a protocol where nodes only forward a broadcast probabilistically. Thus, the broadcast is capable of reaching most of the nodes in the network while reducing overhead. This is based on the observation that a broadcast flood typically has a high level of redundancy [8]. With PBBF, we try to use this redundancy to reduce the *energy* consumed by the broadcast.

TinyOS [2] is an operating system designed at Berkeley specifically for sensors. It favors simplicity and clean design by using a single-thread of execution and a component-based, modular architecture. B-MAC [4] is TinyOS's default power save protocol. In Section V, we integrate PBBF on top of B-MAC. B-MAC uses preamble sampling which means that the packet preamble is long enough to be detected by all nodes that are periodically sampling the channel in between sleep periods (i.e., the preamble must be slightly longer than the sleep time between sampling periods). When sleeping nodes sample the channel and detect the preamble, they remain on to receive the entire packet. See [4] for more details.

This work is supported in part by a NSF fellowship.

[1]While the third parameter has minor effects on latency and energy consumption as well, these metrics are dominated by the other two parameters.

Proceedings of the Fifth Annual IEEE International Conference
on Pervasive Computing and Communications Workshops(PerComW'07)
0-7695-2788-4/07 $20.00 © 2007 IEEE

III. Probability-Based Broadcast Forwarding [1]

In this section, we review our previous work. Probability-Based Broadcast Forwarding (PBBF) can be used with any power save protocol that has the following characteristics:

1) Nodes are scheduled to sleep at certain times and can be awakened on-demand when a neighbor wishes to communicate.
2) Some mechanism ensures that all of a node's neighbors are awake at the same time to receive a broadcast.

In [1], we use IEEE 802.11 PSM [5] as the base protocol to demonstrate PBBF and, in Section V, we use B-MAC [4] as the base protocol. The goal of PBBF is to achieve a specified reliability, with high probability, while allowing a wide-range of tradeoffs in energy consumption and latency. Specifically, we focus on two definitions of reliability in this work: (1) the average fraction of nodes that receive a broadcast and (2) the average fraction of broadcasts received by a node.

PBBF introduces two new parameters to a power save protocol: p and q. The first parameter, p, is the probability that a node rebroadcasts a packet in the current active time even though not all neighbors may be awake to receive the broadcast. With probability $(1-p)$, the node waits to send the packet according to the power save protocol. The second parameter, q, is the probability that a node remains on after the active time when it normally would sleep (the length of time that a node remains on is a parameter of the power save protocol being used). With probability $(1-q)$, the node sleeps as it would in the original power save protocol. Even with these modifications, a node still only rebroadcasts a packet once. In Section IV, we introduce a third parameter that allows a node to rebroadcast a packet twice for added reliability.

Figure 1 shows pseudo-code of changes to any sleep scheduling protocol required for PBBF. The original sleep scheduling protocol is a special case of PBBF with $p = 0$ and $q = 0$. The *always-on* mode (i.e., no active-sleep cycles) can be approximated by setting $p = 1$ and $q = 1$. PBBF may be slightly different from *always-on* in this case. For example, in 802.11 PSM, there is extra byte overhead (e.g., sending advertisements) and temporal overhead (i.e., PBBF cannot send data packets during the advertisement window).

Intuitively, p and q have the following effects:

Energy: As q increases, energy consumption increases. Changing p has a negligible effect on this metric.

Latency: As q increases, latency decreases, provided that $p > 0$. As p increases, latency decreases, provided that $q > 0$.

Reliability: As q increases, reliability increases, provided that $p > 0$. As p increases, reliability decreases, provided that $q < 1$. When p increases, there is a greater probability that a node rebroadcasts the packet immediately. Thus, for a fixed $q < 1$, there is a greater chance that some of its neighbors do not receive the broadcast since they chose to sleep.

If the listed conditions (e.g., $p > 0$ for latency and reliability as q increases) are not met, then the metric is unaffected.

```
SLEEP-DECISION-HANDLER()
 1  /* Called at the end of active time */
 2  /* If stayOn is true, then remain on; else sleep*/
 3  stayOn ← false
 4
 5  if DataToSend = true or DataToRecv = true
 6    then
 7         stayOn ← true
 8    else if UNIFORM-RAND(0, 1) < q
 9           then stayOn ← true


RECEIVE-BROADCAST(pkt)
 1  /* Called when broadcast packet pkt is received */
 2  if UNIFORM-RAND(0, 1) < p
 3    then SEND-BROADCAST(pkt)
 4    else ENQUEUE(nextPktQueue, pkt)
```

Fig. 1. Pseudo-code for PBBF.

```
RECEIVE-BROADCAST(pkt)
 1  /* Called when broadcast packet pkt is received */
 2  if UNIFORM-RAND(0, 1) < p
 3    then SEND-BROADCAST(pkt)
 4         if UNIFORM-RAND(0, 1) < r
 5           then ENQUEUE(nextPktQueue, pkt)
 6    else ENQUEUE(nextPktQueue, pkt)
```

Fig. 2. Pseudo-code for r parameter in PBBF.

IV. PBBF Extension

As mentioned in Section III, the PBBF parameters p and q provide a tradeoff in energy consumption, latency, and reliability for broadcast dissemination. Now, we propose another PBBF parameter that induces an overhead tradeoff in addition to the aforementioned metrics. We denote this parameter as r. When a sensor decides to immediately transmit a broadcast packet according the p parameter (as described in Section III), it will broadcast the packet a second time with probability r. If the packet is broadcast for a second time, then the second transmission is advertised according to the sleep scheduling protocol's original protocol. The pseudo-code for this PBBF extension is shown in Figure 2.

We can see that the r parameter induces an overhead tradeoff into PBBF. By increasing r, we increase the reliability of a broadcast at the expense of increasing the packet overhead in the network. At the extreme, if $r = 1$, then reliability should be close to 100% regardless of the p and q values, but each node is broadcasting every packet twice. This gives users yet another control parameter to achieve a desired tradeoff in the energy consumption, latency, reliability, and overhead planes.

V. Implementation

We implemented PBBF in TinyOS 1.1.15 [2] for the Mica2 Mote [9] sensors. This serves as a proof-of-concept for the protocol and provides results from a real-world communication environment. PBBF is implemented on top of a different sleep scheduling protocol than the 802.11 PSM protocol that was

Proceedings of the Fifth Annual IEEE International Conference
on Pervasive Computing and Communications Workshops(PerComW'07)
0-7695-2788-4/07 $20.00 © 2007 IEEE

the basis for the simulations in [1]. This demonstrates the versatility of PBBF. Additionally, we added the extension to the PBBF protocol described in Section IV.

We chose to implement PBBF in TinyOS [2] since this is a widely used open-source sensor operating system. Its adoption in the research community has led to a relatively stable system with significant documentation. For hardware, we use the Mica2 [9] platform since it has two power save protocols implemented for it.

The two power save protocols available on the Mica2 platform were S-MAC [10] and B-MAC [4]. Either would have been appropriate for our PBBF implementation. We chose B-MAC over S-MAC for reasons discussed in [11].

As described in Section II, B-MAC uses preamble sampling for in-band power saving. Sensors wake up according to a specified duty cycle and carrier sense the channel. If the channel is idle, the sensor returns to sleep until the next scheduled carrier sense period. If the channel is busy, the sensor continues listening to channel in anticipation of receiving a pending data packet. When a node has data to transmit, it attaches a preamble longer than the duty cycle in order to guarantee that all nodes will carrier sense the channel at some point during the preamble and continue listening.

To implement PBBF on B-MAC, we made these changes:

- When a node carrier senses the channel idle during its duty cycle, with probability q, it continues listening to the channel until its next scheduled carrier sensing period.
- When a node has a packet to rebroadcast, with probability p, it transmits the packet without the long preamble. In this situation, most of the node's neighbors will not carrier sense the preamble and, hence, not receive the broadcast packet at that time. With probability $(1-p)$, the node will rebroadcast the packet with the long preamble so that its neighbors will carrier sense it and receive the subsequent data packet.
- When a node rebroadcasts the packet *without* the long preamble (as discussed in the previous item above), with probability r, it will broadcast the packet a second time. This second broadcast will use the long preamble.

The architecture we used for our implementation is shown in Figure 3. The solid arrows in the figure represent the interface that connects two modules. The notation $A \xrightarrow{I} B$ indicates that component B implements interface I and that component A uses B's implementation of interface I. The dashed arrows indicate the message type that the connected module uses to send and/or receive via `GenericComm`. Details about the interfaces and packet types are in [11]. The `GenericComm`, `UART`, and `CC1000Radio`[2] components are already implemented in TinyOS. We made some modifications to the CC1000Radio modules, but used these components, for the most part, in their current TinyOS instantiation. We now describe the functionality of each component from Figure 3.

DummyBcastSrc: This is the application to test PBBF. The node with ID 0 is set as the broadcast source and transmits a broadcast periodically at a desired rate. Non-source nodes that receive broadcast packets pass information to the `Stats` module to collect experimental data.

`DummyBcastSrc` also serves as the link to the serial port (UART) for communication with a computer. The module also passes control packets (e.g., what p, q, and r parameters to use for a particular run) to `CtrlPktHandler`. Finally, it maintains all the timers for when an experimental run ends and when statistics are sent back to the broadcast source.

SimplePbbfBcast: The main functions of this module are duplicate suppression and queuing for `DummyBcastSrc`'s broadcast packets. Control packets are also passed to `CtrlPktHandler` and `Stats` is notified of *every* broadcast sent or received.

CtrlPktHandler: This module handles incoming control packets by setting p, q, and r to the values specified in the packet. This is done via its connection to `Pbbf`.

Stats: This module keeps track of statistics for our experiments and aggregates the information in packets to send back to the broadcast source. Via `DummyBcastSrc`, it keeps track of the end-to-end latency of received packet as well as the total number of unique, application-layer received packets. `SimplePbbfBcast` informs `Stats` of the total number of data packets sent and received. `Pbbf` signals to `Stats` when a packet was transmitted twice due to the r parameter (as discussed in Section IV). `CC1000Radio` signals this component whenever the radio switches to and from sleep mode to track the total fraction of time spent sleeping.

This module also provides an end-to-end retransmission scheme for extra reliability in reporting experimental stats. This is somewhat useful since the link layer

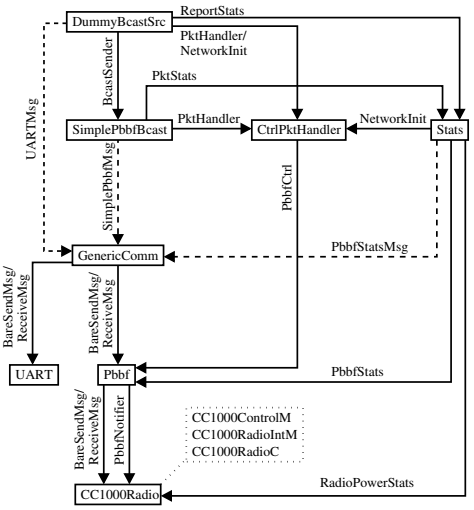

Fig. 3. TinyOS architecture for PBBF implementation. The solid rectangles are modules (`CC1000Radio` is an abstraction for the three modules listed in the dotted lines). The solid arrows represent the major interface(s) that connect modules. The incoming dashed lines to `GenericComm` represent the message types the connected module uses.

[2] `CC1000Radio` is an abstraction for the modules listed in the dotted lines.

Proceedings of the Fifth Annual IEEE International Conference
on Pervasive Computing and Communications Workshops(PerComW'07)
0-7695-2788-4/07 $20.00 © 2007 IEEE

retransmission scheme seems to occasionally fail.

GenericComm: *(Existing TinyOS module)* This serves to multiplex and demultiplex packets in TinyOS based on the packet type. Essentially, the packet type serve as ports do in traditional TCP/UDP communications

UART: *(Existing TinyOS module)* This component provides the lower level communication with the serial port.

Pbbf: This is the actual implementation of the PBBF protocol. It is placed between `GenericComm` and the `CC1000Radio` components. `GenericComm` is analogous to the network layer and `CC1000Radio` provides the medium access and the physical layer.

The p, q, and r values that `Pbbf` uses are input from `CtrlPktHandler`. B-MAC notifies `Pbbf` of a decision point for whether to sleep via the `PbbfNotifier` interface. At this point, `Pbbf` compares the current q value to a random number to decide whether to tell the radio to sleep, as would be normal operation, or continue listening to the channel, which is part of PBBF. For every packet received from `GenericComm`, `PBBF` decides, based on the p and r values, whether to use a long preamble and whether to transmit the packet twice, respectively. This layer also provides link layer retransmissions since this feature is not implemented in lower layers (i.e., `CC1000Radio`).

CC1000Radio: *(Existing TinyOS modules)* These components provide lower level communication with the CC1000 radio [12] on Mica2 Motes. Additionally, the B-MAC [4] implementation is integrated into these components.

VI. Experimental Results

To test our work, the broadcast source was attached to a laptop via a MIB510CA board. This sensor was also the sink for reporting statistics back to the laptop. Our Motes used the 433 MHz band. We were constrained to using only nine Motes total, so the other eight Motes served as broadcast receivers.

We only experimented on a topology where all of the devices are in range of the broadcast source (and each other) since statistics reporting was too unreliable in a multihop setting. This setup was also useful since the number of Motes was limited and PBBF relies on some amount of density to operate efficiently. Most importantly, this simple scenario is sufficient for demonstrating key properties of PBBF.

In our experiments, the source transmitted a broadcast every 2.5 s. Each experiment ran for 30 s, which results in 11 packets being sent per run (the first packet is not sent immediately when the test commences). Each data packet uses the standard TinyOS format with 2 synchronization bytes, 5 header bytes, 2 CRC bytes, and a payload of 29 bytes. The default preamble adds an additional 8 bytes, though, as described in Section II. B-MAC increases the preamble length according to how much power saving is desired. In our tests, we set the B-MAC parameters to have a duty cycle of 135 ms and preamble size of 371 bytes. We note that when a sender decides to transmit immediately, according to the p parameter in PBBF, the preamble size is set to the default 8 bytes for that particular packet. We also note that the version of B-MAC we used carrier senses the channel for 8 ms once every duty cycle. If the channel is carrier sensed busy, then B-MAC extends the time that it is awake for 32 ms. At the end of this 32 ms interval, B-MAC carrier senses again and will sleep or extend its listening for another 32 ms depending on if the channel is idle or busy, respectively. For statistics collection, once the sensor has run the experiment for the specified 30 s length, it switches power save off for 10 s and reports its data.

The metrics that we measured are:

- **Fraction of Time Not Sleeping:** Obtaining fine-grained energy measurements for the Motes requires special equipment. Thus, we use a coarse-grained metric where we track how much time a node spends with its radio not in the sleep state over the course of an experiment. Thus, the larger the fraction of time not sleeping, the more energy is generally being consumed by the radio.
- **Average Broadcast Latency:** This is the average latency from the time a packet is sent at the sender's application layer until the *data* begins transmission over the radio (i.e., after the preamble and synchronization bytes are transmitted). For this, we use the time stamping described in [13]. Again, this is not as fine-grained of a metric as we would like. However, this technique obviates the need for time synchronization among the nodes which would induce a large amount of complexity and overhead to our implementation. We only compute the latency for nodes that received a given broadcast.
- **Unique Data Packets Received:** The average fraction of broadcasts sent by the source that are received by listening nodes.
- **Total Data Packets Received:** This is a benchmark for the receive overhead of the protocol. It is measured as the average total broadcasts received divided by the number of broadcasts sent by the source. Since sensors filter duplicate broadcast packets (with respect to the source and sequence number), the total data packets received is greater than or equal to the unique data packets received.
- **Total Data Packets Sent:** For brevity, these results are omitted. Please see [11] for details about this metric and the corresponding results.

To test the effects of p, q, and r, we set their values to 0.0, 0.3, 0.7, and 1.0 and ran one experiment (with multiple broadcasts) for each of the 64 possible combinations of these three variables using these four values. The goal of this section is to give some intuition of how the r parameter affects these metrics. In a real system, an administrator would adjust each of the three parameters to achieve desired tradeoffs. For brevity, we only present the results in which the r parameter is the independent variable. More results and discussion is available in a tech report [11] (which also shows that our implementation has the same trends observed in simulation for energy consumption, latency, and reliability as a function of the p and q parameters).

Figure 4 shows energy consumption. When $p = 1$, $q = 0$,

Proceedings of the Fifth Annual IEEE International Conference
on Pervasive Computing and Communications Workshops(PerComW'07)
0-7695-2788-4/07 $20.00 © 2007 IEEE

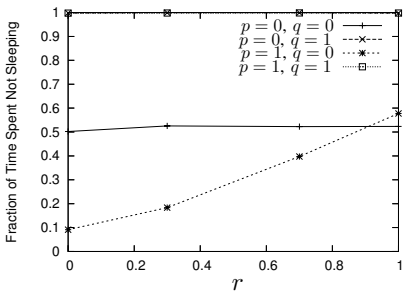

Fig. 4. Energy consumption. $q = 1$ curves overlap at the top.

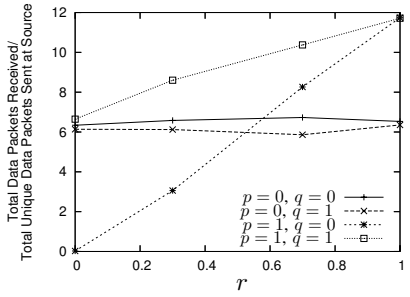

Fig. 6. Reception overhead.

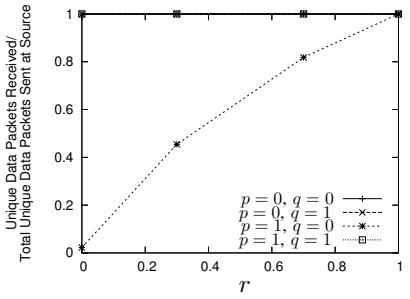

Fig. 5. Reliability of broadcast. All curves except for $p = 1, q = 0$ overlap at the top.

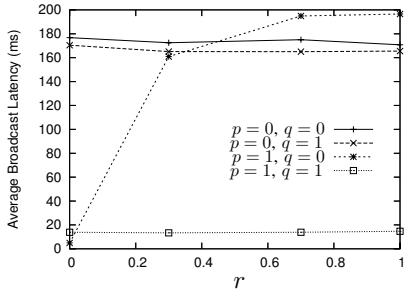

Fig. 7. Average broadcast latency.

we can see that the nodes use more energy due to the increase in reliability that the increasing r is providing. The reliability improvement with r is illustrated in Figure 5.

Figure 6 shows the overhead for receptions (the results for transmission overhead follow similar trends [11]). These results show that the overhead doubles when $p = 1$ and $q = 1$ as r goes from 0 to 1. This occurs because when $p = 1$, each sensor will transmit each broadcast once when $r = 0$ and twice when $r = 1$. When $p = 0$, we see no effects on the overhead, with respect to r, as expected. When $p = 1$ and $q = 0$, then the overhead is zero when $r = 0$ due to the lack of reliability. The increasing reliability with r causes the overhead to increase linearly.

Figure 7 shows the average latency. As expected, this metric is primarily dominated by the p and q values instead of r. The only exception is $p = 1, q = 0$, where the latency is very low when $r = 0$ because the reliability is very low (Figure 5) and only a few low-latency packets are received. When the r value is larger, then more broadcasts are received according to the original power save protocol. This increase both latency and reliability as seen in Figure 7 and Figure 5.

VII. Conclusion

In previous work [1], we proposed PBBF, a lightweight protocol that allows lower latency broadcast propagation in power save networks in a energy efficient manner. With PBBF, a user has fine-grained control over the energy consumption for a broadcast to achieve a desired latency and reliability.

In this work, we have proposed a PBBF extension for improved reliability at the cost of increased packet overhead. Additionally, we designed and implemented a PBBF architecture in TinyOS [2]. Our evaluations show the energy consumption, latency, reliability, and overhead tradeoffs possible using PBBF on Mica2 [9] hardware.

References

[1] M. J. Miller, C. Sengul, and I. Gupta, "Exploring the Energy-Latency Trade-off for Broadcasts in Energy-Saving Sensor Networks," in *IEEE ICDCS 2005*, June 2005.

[2] TinyOS Community Forum, `http://webs.cs.berkeley.edu/tos`.

[3] T. Starner, "Thick Clients for Personal Wireless Devices," *IEEE Computer*, vol. 35, no. 1, pp. 133–135, January 2002.

[4] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *ACM SenSys 2004*, November 2004.

[5] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.

[6] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, "A Survey of Energy Efficient Network Protocols for Wireless Networks," *ACM Wireless Networks*, July 2001.

[7] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-Based Ad Hoc Routing," in *IEEE Infocom 2002*, June 2002.

[8] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," in *ACM MobiCom 1999*, August 1999.

[9] MICA2 Mote Datasheet, `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-05_A_MICA2.pdf`.

[10] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," in *IEEE Infocom 2002*, June 2002.

[11] M. J. Miller and I. Gupta, "An Implmentation of Probabilistic Broadcast to Address the Energy-Latency Tradeoff," University of Illinois at Urbana-Champaign, Tech. Rep., August 2006.

[12] Chipcon CC1000 Datasheet, `http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf`.

[13] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The Flooding Time Synchronization Protocol," in *ACM SenSys 2004*, November 2004.

Proceedings of the Fifth Annual IEEE International Conference
on Pervasive Computing and Communications Workshops(PerComW'07)
0-7695-2788-4/07 $20.00 © 2007 IEEE