Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks

Răzvan Musăloiu-E.
Computer Science Dept.
Johns Hopkins University
razvanm@cs.jhu.edu

Chieh-Jan Mike Liang
Computer Science Dept.
Johns Hopkins University
cliang4@cs.jhu.edu

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

Abstract

*We present Koala, a reliable data retrieval system designed to operate at permille (.1%) duty cycles, essential for long term environmental monitoring networks. Koala achieves these low duty cycles by letting the network's nodes sleep most of the time and reviving them through an efficient wake-up strategy whenever the gateway performs a bulk data download. Unlike other systems which consume energy to maintain consistent network state (*e.g. *routes, sleep schedules, etc.) across the network's nodes, Koala maintains no persistent routing state on the motes. Instead, a basestation calculates the network paths using reachability information collected by the motes. The Flexible Control Protocol (FCP), a protocol we developed, is then used to install this routing information on the network's nodes. This paradigm of operation not only eliminates the overhead of maintaining routing state, but also significantly reduces the complexity of the networking code running on the motes. Results from simulation and an actual implementation on TinyOS 2 indicate that Koala can achieve very low duty cycles under a wide range of download and network sizes.*

1. Introduction

Routing protocols for long term data gathering applications, such as environmental monitoring, need to support low duty cycles ($<1\%$), reliably deliver collected measurements, and operate unattended for long periods of time. In response to these requirements, WSN networking stacks employ techniques to coordinate the nodes' sleep schedules and maintain states at each of the network's motes (*e.g.* routing entries, link quality information, etc.) [3]. In turn, implementing these techniques leads to networking software with increased complexity running on resource constrained motes. This complexity coupled with the unexpected and untested environment in which the network is deployed can lead to failures. In fact, a number of deployments have reported failures related to networking code running on the network's motes [23, 24, 26].

In this paper we present a different approach to this problem that offers equivalent or better efficiency, while avoiding most of the mote-side complexity associated with existing networking protocols. Specifically, our approach requires motes to maintain *no persistent networking state*. This feature not only lowers the complexity of the networking code that runs on the motes, but also removes the overhead of keeping that state consistent throughout the network. This is a significant advantage for low duty cycle applications, for which control traffic can be on par if not higher than application traffic.

The paper makes two main contributions: **I.** The design and development of the **Flexible Control Protocol (FCP)**, a signaling protocol used to install routing paths on the network's motes. A WSN gateway uses FCP to create the multi-hop paths over which it downloads data from the motes. FCP supports ephemeral paths that transmit a single datagram and persistent paths that persist until explicitly torn down. Both paths can offer reliable transfers. **II.** The design and development of **Koala**, a system for reliably downloading bulk data that targets data gathering applications with no real-time requirements. Koala uses FCP to establish network paths, coupled with Low Power Probing (LPP), an efficient technique to wake up the network's motes before a download occurs. Furthermore, Koala leverages the availability of multiple channels in 802.15.4 radios to perform data downloads over different channels, thereby minimizing overhearing costs.

Results from simulations and a testbed of 24 Tmote Sky motes running TinyOS 2, show that the proposed approach can achieve duty cycles of 0.2% or lower, in networks with up to hundreds of motes. We show that channel switching provides considerable benefits, reducing download times by up to five times. Moreover, users can select the desired duty cycle by controlling how frequently they download data from the network. Finally, our results indicate that LPP can wake up a network of 24 motes in under 30 seconds.

This paper has seven sections. In the section that follows we argue for a network-wide control plane and outline

978-0-7695-3157-1/08 $25.00 © 2008 IEEE
DOI 10.1109/IPSN.2008.10

the proposed architecture. Section 3 presents the design of FCP, a signaling protocol used to establish network paths. In Section 4 we present how FCP is used as part of the Koala data gathering system. We evaluate Koala's performance through simulations and results from an early implementation in Section 5. Finally, we present related work in Section 6 and close in Section 7 with a summary.

2. Overview

The motivating application for this work is environmental monitoring [15, 20, 26]. Environmental monitoring, at its basic form, involves reliably gathering the measurements from each of the network's motes at a gateway. Furthermore, in order to observe long term spatial and temporal trends, these networks need to be deployed for one or more years and cover large geographic areas. These requirements mean that such networks must have very low duty cycles ($\sim 0.1\%$) and support tens to hundreds of motes per gateway, deployed in sparse topologies.

In response to the dual requirement of energy-efficient operation and large scale, previous proposals have combined multi-hop routing protocols with duty cycling [3]. These protocols synchronize the sleep schedules of neighboring nodes, collect and disseminate link quality information, and calculate routes through distributed min-cost routing algorithms. On the other hand, experience has shown that implementing complex networking protocols on motes can lead to unexpected failures in the field [10, 24, 26].

These problems led us to investigate strategies for simplifying the mote's routing software. Generally speaking, routing can be divided into the *data* and *control planes*. The first includes all the components necessary to forward packets along a multi-hop path. To do so, it relies on a forwarding table whose entries list the next-hop(s) on the path towards a particular destination. This forwarding table is maintained by the control plane which includes routing protocols that discover and select network paths.

We propose to decouple control and data planes at the mote level, by implementing a *network-wide routing control plane*. The majority of the functionality of this network-wide plane is implemented at a centralized location (*e.g.* a gateway) using information collected by the network's motes. This information is then used to calculate and disseminate the end-to-end paths that motes will use.

The scenario shown in Figure 1 exemplifies our approach. The network's motes in this example take measurements and store them in local flash. A gateway then periodically downloads the collected data. To do so, the gateway first instructs the nodes to collect information about their neighbors. The gateway uses this information to calculate *high quality* network paths, that is paths with consistently low packet loss rate. The next step involves disseminating path information to the network's motes which subsequently use it to forward data back to the gateway. We use the Flexible Control Protocol (FCP) for the data collection and dissemination tasks. In this respect, FCP is used to install (forwarding) paths in the network and is therefore similar to other signaling protocols, such as LDP [1].

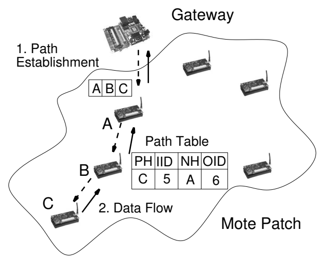

Figure 1. Example of application using the Flexible Control Protocol (FCP).

Compared to routing protocols implemented at the mote level, this approach provides multiple benefits in addition to reduced complexity. First, motes do not incur the overhead of persistently maintaining routing state. Moreover, the network-wide view provides the ability to perform other optimizations. For example, a node can establish two disjoint paths to the same destination to improve reliability and/or load balancing. At the same time, we require a centralized entity to establish these network paths. Most deployments however already include a gateway for connecting the WSN to the Internet.

FCP can be combined with other protocols and network services to implement custom applications. One such a example is *Koala*, an ultra-low power system for reliable data retrieval. Koala uses FCP to establish reliable network paths for downloading sensor data. To achieve ultra-low duty cycles Koala couples FCP with *Low Power Probing* (LPP), a novel mechanism for waking up the network. Unlike Low Power Listening (LPL), in which receivers periodically poll the channel for long preambles from senders, nodes in LPP send explicit probes and wake up if their probes are acknowledged. Moreover, Koala leverages the availability of multiple frequency channels in 802.15.4 radios to reduce overhearing.

3. Flexible Control Protocol

Figure 2 represents the position of the Flexible Control Protocol (FCP) relative to other protocols in the emerging TinyOS 2 networking stack. Specifically, FCP sits directly above the Active Message layer which offers the ability to send unicast and broadcast messages to nodes within the same broadcast domain. In turn, FCP provides upper layers

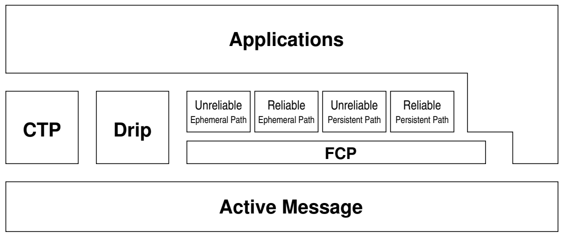

Figure 2. The Flexible Control Protocol (FCP) and its relations to other protocols in the emerging TinyOS 2 network protocol architecture.

the ability to send one or more messages across multi-hop paths with or without end-to-end reliability.

Because motes do not keep any persistent routing state, a network path[1] must be established before it can be used to carry traffic. For this reason, FCP includes a path establishment phase that installs entries on the path tables of each of the nodes on the path. Nodes subsequently use these entries to forward packets, until they are explicitly or implicitly removed, thus relinquishing allocated resources.

3.1. FCP Services

Depending on whether paths are used to transmit one or multiple data packets, FCP provides *ephemeral* and *persistent* network paths.

Ephemeral Network Path. This service is equivalent to a source route since the data packet carries the network path it should follow in addition to the application's data. Intermediate nodes do not establish any state but rather forward the packet based on the path encoded in it. This service is useful when a node wants to send a short message, such as a command, to another node in the network. It has the advantage of not incurring the delay and the overhead associated with establishing the path. On the other hand, the maximum amount of application data that an ephemeral path can carry is limited to a single radio packet, minus the space necessary to store the path information.

Persistent Network Path. Unlike ephemeral paths, persistent network paths must be established before they can be used. This establishment phase requires intermediate nodes to allocate entries on their path tables. These entries are used to forward subsequent packets that do not carry source routes. At the end of a successful establishment phase the nodes at both ends of the path receive a path identifier that they use to forward traffic in both directions.

[1]FCP network paths are analogous to virtual circuits or MPLS label-switched paths (LSPs) but with no QoS attributes associated to them.

Both path types can offer end-to-end reliability, meaning that the destination will generate acknowledgments for each of the packets it receives. Moreover, intermediate FCP nodes will attempt to deliver the packets up to a maximum number of times and notify the sender if the path is no longer available.

3.2. Implementation

A major part of FCP relates to updating the entries in nodes' path tables during path establishment and termination. As Figure 1 illustrates, each entry in this table has four entries: Previous Hop (PH), Incoming ID (IID), Next Hop (NH), and Outgoing ID (OID). Packets that arrive from upstream node PH, carrying the IID identifier are forwarded to downstream node NH after their identifier has been changed to OID.

To establish an end-to-end path, the source first constructs a PATH_OPEN FCP control message which includes the end-to-end route. This route is a sequence of addresses corresponding to the network nodes that the message should traverse. FCP is agnostic about the addresses used; the only requirement is that such an address can be used to reach a node that is within the same broadcast domain of the current node. One can use MAC addresses but other identifiers, such as TinyOS node identifiers can also be used.

The network path establishment phase starts after the source node forwards the PATH_OPEN message to the next hop. Path IDs have local scope, having to be unique only between the same pair of nodes. This means that the identifier space can be relatively small, reducing the overhead of allocating and carrying path IDs. Once an outgoing ID has been allocated, the PATH_OPEN message can be forwarded to the next downstream node until it reaches its final destination. To reduce FCP's memory footprint the path table has an upper limit (the current implementation supports up to 32 concurrent connections). If a node's path table is full, the node replies with a PATH_CLOSE message which uses the source route included in the original PATH_OPEN message. Intermediate nodes that receive the PATH_CLOSE message remove the corresponding path entry before forwarding the message upstream.

FCP uses link-layer retransmissions to improve the probability of successful packet delivery. We adopted this mechanism, used both for data and control packets, because it has been shown that link-level retransmissions enhance the reliability of WSN multi-hop paths considerably [21]. To provide link-layer retransmissions, we leverage the hardware acknowledgments that modern radio chips, such as the CC2420 offer [25]. Compared to application-level acknowledgments, hardware acknowledgements are faster because they are sent immediately after the radio receives the original packets.

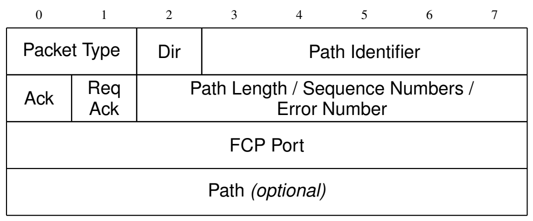

Figure 3. The FCP header.

If a packet can not be forwarded after five attempts, the link is considered to have failed and the network path is terminated. After all retransmission requests have failed, the node upstream of the failed link returns a PATH_CLOSE message to the path's source. We note that end-to-end acknowledgments, generated by the destination for reliable paths, are different from link-level acknowledgments and retransmissions. Their purpose is to inform the source that the destination has received its packets.

Network paths are terminated in two other cases. First, each of the communication endpoints can explicitly terminate an existing path by sending a PATH_CLOSE control message. Finally, FCP uses soft state and therefore intermediate nodes automatically remove table entries that have not been recently used (the current timeout is set to 20 seconds). If a node receives a data message carrying an unknown path identifier it responds with a PATH_CLOSE message.

Figure 3 illustrates the FCP header, starting with the Packet Type field. Because network paths are bidirectional, the Dir field indicates the direction in which source routes and path identifiers should be processed. The Path Identifier field stores the identifier used by the previous hop and is used to insert an entry in the path table in the case of PATH_OPEN messages or to lookup the next hop for data messages. The Req Ack and Ack fields are used to request and return acknowledgments respectively. The next field is used as sequence number, acknowledgment number, error code, or as the length of the source route depending on the packet type. The FCP Port field indicates the receiving FCP application or service at the destination node. Last, the header may include an optional routing header which is a list of node identifiers used as a source route.

4. Koala

As Figure 2 implies, FCP provides useful communication abstractions. In practice, FCP will be coupled with other protocols to develop a complete application.

We provide an example of this paradigm through *Koala*, a system for reliably extracting bulk data from duty cycled networks. We envision an environmental monitoring network in which motes store collected measurements in their local flash memory until a gateway extracts them. Such a gateway does not have to be always present; rather it can periodically join the network, establish network paths and reliably retrieve data before it withdraws. This model of operation matches well with the architecture described in Section 2, as resources are not consumed to maintain persistent routing state.

4.1. Low Power Probing

Because current mote radios consume as much energy in idle listening mode as when they transmit or receive [25], nodes must maximize the time they keep their radios turned off. This means that a mechanism is necessary to wake up the network prior to a download operation. One potential solution would be for nodes to keep synchronized sleep schedules as in [3, 28]. Doing so would however require motes to maintain persistent network state (*i.e.* their neighbors' sleep schedules) which contradicts our philosophy of simplifying mote-level networking code.

Low Power Listening (LPL), in which nodes periodically sample the channel for signs of activity and transmitters send long preambles to generate such activity, offers an appealing alternative for waking up non-synchronized nodes. While initially presented in the context of bit-stream radios ([17]), LPL has been adopted to packet based radios, in which case the preambles consist of a continuous stream of packets [2].

LPL however was designed for waking up individual nodes rather than the whole network, as Koala requires. While using LPL in broadcast mode is possible, doing so requires transmission of maximum length preambles, leading to packet storms that impede the collection of neighborhood data (described next). The underlying reason is that, unlike the unicast case, the sender does not know when all intended receivers have woke up. For this reason, it cannot terminate the preamble's transmission early. False negatives, situations in which a node fails to correctly detect a preamble, represent an even bigger threat. While not a significant issue in the unicast case –a false negative can be detected due to the lack of an acknowledgment, thus scheduling a retransmission at the sender– missed detections can cause nodes to completely miss the opportunity to wake up and participate in a download.

Low Power Probing (LPP), a technique we developed, addresses these problems by replacing passive channel probing at the receiver with active probing. Specifically, nodes periodically broadcast short packets requesting acknowledgments. If such an acknowledgment is received, the node wakes up and starts acknowledging other nodes' probes, otherwise it goes back to sleep. Figure 4 provides a graphical representation of LPL and LPP. LPP replaces Clear Channel Assessment (CCA) samples at the receiver

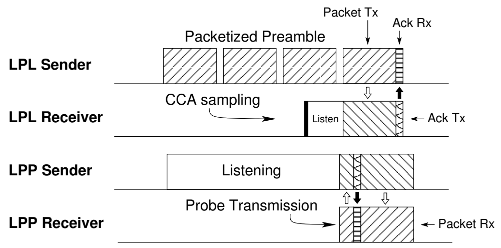

Figure 4. A simplified representation of LPL for packet-based radios and LPP. Preamble and packet durations are not drawn to scale.

Algorithm 1 Lower Power Probing

```
procedure SLEEP($interval$)
  loop
    TURNRADIOOFF()
    DEEPSLEEP($interval$)
    TURNRADIOONWITHACKDISABLED()
    $r \leftarrow$ SENDPROBE()
    if WASACKED($r$) then
      ENABLERADIOACKS()
      return
```

with transmissions of short packets. In turn, this obviates the need for long preambles thus reducing the level of contention on the radio channel.

Algorithm 1 presents LPP in pseudo-code. Enabling and disabling of acknowledgments is necessary to avoid false positives when the probes of two or more nodes cause them to wake each other up by mistake. When the SLEEP() procedure returns, the node keeps its radio on until the next time the procedure is called. This procedure executes only at the network's motes. The wake up operation is initiated by a gateway which enables its radio's acknowledgements and starts listening for probes from the network's nodes.

4.2. Neighborhood Collection

While the gateway selects the routes in Koala, its decisions are driven by information that the network's motes collect. Specifically, once awake, each node collects its neighborhood by recording the identities of its neighbors as well as the quality of its links from these neighbors, defined as the Received Signal Strength (RSSI) of the received packets. These RSSI values are collected from the wake up probes (and the acknowledgments to these probes) received by the node's neighbors. Furthermore, to accelerate the neighborhood collection process, nodes send periodic beacons which are also acknowledged, generating bi-directional link information[2].

We require two properties from the beaconing scheme: to generate a bounded amount of traffic overhead, independent of node density, and to be fair. To achieve these properties, nodes select their beaconing intervals from an exponential distribution and suppress their transmission if they receive a beacon before their timer expires. The memoryless property of the exponential distribution ensures fairness, while suppression limits the total number of beacons. Generating an exponential distribution from the uniform distribution requires computing $\log(x)$ with $x \in [0, 1]$. In practice, we found that approximating $\log(x)$ with the first term of its Taylor series

$$\log(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} \dots$$

produced satisfactory results.

[2]A node stops transmitting beacons once it participates in a download operation.

Algorithm 2 Neighborhood Collection

```
procedure NEIGHBORHOODCOLLECTION($bs$)
  $add \leftarrow$ INITQUEUE($bs$)
  while QUEUEEMPTY($add$) $= False$ do
    $node \leftarrow$POPQUEUE($add$)
    $path \leftarrow$BUILDPATH($node, parent, bs$)
    $r \leftarrow$SENDNEIGHBORHOODREQ($node, path$)
    if $r \neq Empty$ then
      for each $(n, rssi)$ in $R$ do
        UPDATENEIGHBORHOOD($n, rssi$)
        if INQUEUE($n, add$) $= False$ then
          $parent[n] \leftarrow node$
          $r \leftarrow$APPENDQUEUE($n$)

procedure BUILDPATH($n, p, s$)
  $r \leftarrow$INITLIST(s)
  while $n \neq s$ do
    $r \leftarrow$APPENDLIST($p[n]$)
    $n \leftarrow p[n]$
  return $r$
```

The gateway uses unreliable persistent FCP paths to collect the nodes' neighborhood information every time it wakes up the network. It does so by following the procedure outlined in Algorithm 2. In summary, the gateway uses the neighbor information it collects directly, to download the neighborhoods of its immediate neighbors. Using this information, the gateway extends its network knowledge by another hop. Then, for each two-hop neighbor *x*, the gateway selects the link between *x* and its existing one-hop neighbors which has the highest RSSI value (say *y*). The path to *x* then is built by extending the path to *y*. The advantage of this approach is that new paths are always constructed by extending existing high-quality paths. The algorithm terminates after the gateway retrieves neighborhood information from all the nodes.

4.3. Routing Path Selection

Routing path selection is a two-step process that starts once the gateway retrieves neighborhood information from all the network's nodes. The gateway first computes the depth of each of the network's nodes through a breadth-first search (BFS) of the collected network topology, in which all links are initially considered equivalent. During the second step, we compute a path from each of the network's nodes back to the gateway. We do this by starting from the selected node and randomly following a good link towards a neighbor that is closer to the gateway (*i.e.* at a higher level of the BFS tree). In this context, good links are those with RSSI values higher than -70 dBm. We use this threshold to ensure that only stable links with low packet loss are used for data downloads [13]. We randomly choose among good links rather than selecting the best link to exercise multiple links. This way, the load of retrieving data is distributed more evenly among the network's nodes. If a path fails, indicated by an FCP error, the gateway selects an alternate path to download data from the current node.

4.4. Channel Switching

Once the network is active and the gateway has selected the paths it will use, it starts to download data sequentially from each of the network's nodes. However, downloading large blocks of data ($\sim$ $100-200$ KB) over multi-hop paths can take from tens of seconds to minutes depending on link conditions. Nodes that do not participate in the download waste energy during this time. Therefore, it is desirable to put these nodes to sleep. However, due to the way LPP works, even if such nodes go to sleep, they will be awaken because active nodes will acknowledge their probes.

To avoid these spurious wake ups the gateway instructs all nodes on the current download path to switch to a different frequency channel[3] before the download starts. Once the download completes, the gateway instructs the nodes to return to the common command channel. Both operations use reliable ephemeral FCP network paths. To switch the path $N_1, N_2, \dots N_k$, the gateway initiates k sequential channel switch requests starting from the node farthest from it (*i.e.*, N_k) and ending with N_1. If all channel switch operations are successful, the gateway initiates the download operations. Because the same path is re-used to download data from all k nodes, as soon as a download completes the gateway asks the source node to return to the command channel and go to sleep.

Algorithm 3 provides a formal description of both channel switch operations. While a number of failures can occur during a channel switching operation, Koala will eventually recover from all of them because motes return to the command channel if no FCP activity is detected within a certain amount of time.

4.5. Data Download

The gateway uses reliable persistent FCP paths to download the data from the network's motes. The only remaining

[3]802.15.4 radios provide 16 non-overlapping frequency channels.

Algorithm 3 Channel Switching

```
procedure PathSwitching($path$)
    $s \leftarrow$ InitStack()
    $t \leftarrow$ InitStack($path$)                    ▷ Last element is the top
    $c \leftarrow$ RandomChannel()
    while StackEmpty($t$) $= False$ do
        $node \leftarrow$ PopStack($t$)
        $r \leftarrow$ SendChSw($c, s$)
        if $r = Failed$ then
            break
        else
            PushStack($s, node$)
    while StackEmpty($s$) $= False$ do
        $node \leftarrow$ PopStack($t$)
        if NeedsDownload($node$) then
            Download($s$)
        NodeDeSwitching($s$)

procedure NodeDeSwitching($p$)
    for $node$ in $p$ do
        if NeedsDownload($node$) then
            SendChSwWithSleep($CmdChannel, p$)
            return
    SendChSwitch($CmdChannel, p$)
```

challenge is to select the appropriate inter-packet interval with which the source should inject packets to the network, to avoid collisions with copies of its packets forwarded upstream. It is easy to show that in a download path in which each node interferes only with its predecessor and successor, the source should inject one packet for every three time slots (*i.e.* time necessary to transmit a single packet over a single hop) to avoid collisions. However the correct inter-packet delay is not known in the general case, because the interference graph is not known.

One could derive the optimum delay $OptDelay(m)$ for paths of length m for the worst case scenario in which all nodes interfere with each other. In practice however deployments are sparse and therefore this approach will produce suboptimal results. One way we can reduce this delay bound is by using the collected neighborhood information. To do so, for each node n on the download path p, we compute $PathNeighbors(n, p)$ which is the number of neighbors that n has in p. The inter-packet delay can then be set to the largest $PathNeighbors()$ value since it represents the maximum interference at any single hop on the path.

However, both approaches do not consider the impact of hop-by-hop retransmissions in the face of packet loss. Such retransmissions increase the time required for a packet to "clear" an upstream hop and thus require larger and, more importantly, dynamically adjustable inter-packet delays at the source. For this reason, we opted for an alternative approach in which the gateway uses the acknowledgments that FCP generates to keep a running estimate of the path's round trip time (RTT). The source then injects a new packet every RTT/2 seconds. The rational is that after RTT/2 seconds the last packet most likely has exited the path, and it is safe to send the next packet. Algorithm 4 presents the full gateway's logic. We acknowledge that this approach

Algorithm 4 Data Download

```
procedure DOWNLOAD($path$)
    $start \leftarrow$ GETTIME()
    $r \leftarrow$ SENDNEIGBORHOODREQ($p$)
    $rtt \leftarrow$ GETTIME()$-start$
    if $r = Failed$ then
        return
    SENDDOWNLOADREQ($p, rtt/2$)
    repeat
        $r \leftarrow$ RECEIVEDATADOWNLOAD()
    until $r = Failed \bigvee$ LASTPACKET($r$)
```

is suboptimal, transmitting slower than the optimal sending rate especially over paths with long RTTs. As part of our future work we plan to address this limitation by exploring the benefits of using sophisticated rate control algorithms such as the ones in [9, 16].

Once the gateway finishes downloading data from all the intended nodes, it leaves the network or goes to sleep. The rest of the network should also go to sleep when this happens. We achieve this behavior using the Drip dissemination protocol [12]. Specifically, the gateway periodically (once every five seconds) disseminates monotonically increasing values for key K. Each mote that receives an updated K value resets its internal timer (set to 15 seconds). If on the other hand, the mote does not receive a new value before its timer expires, it goes to sleep.

5. Evaluation

5.1. Methodology

The metric we use to evaluate Koala's performance is the total time required to download a certain amount of data from every network node. We chose this metric because it represents the energy cost associated with Koala's operation. To identify how this cost is distributed across the different Koala phases described in Section 4, we further divide the total time into the time necessary to wake up the whole network and the time required to download data from all of its nodes. Finally, we evaluate the efficiency of LPP by comparing it with two versions of LPL included in the TinyOS 2 distribution.

We evaluate Koala's performance across different dimensions by varying the length of the LPP probing interval, the size and the diameter of the network, and the amount of data downloaded from each mote with and without channel switching. This evaluation is based on a combination of results from simulation and a prototype implementation. We use the TOSSIM simulator, which we enhance to simulate all the components of the CC2420's radio stack, other than LPL. In this way, the simulation and the implementation use identical FCP and Koala codebases.

TOSSIM requires the user to supply the gain of the links used in the simulated topologies. We compute these gains

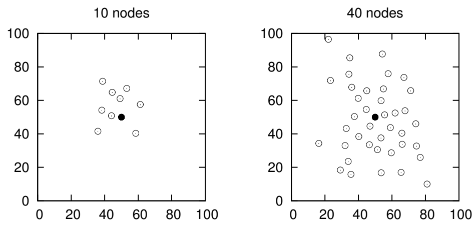

Figure 5. Examples of random topologies with ten and forty nodes.

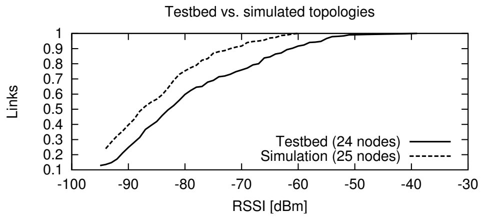

Figure 6. CDFs of link RSSIs for the 24-node testbed and randomly generated topology.

using the Log Distance Path Loss model with a path exponent of four, to approximate challenging signal propagation environments. Furthermore, we model noise using the CPM model recently added to TOSSIM [11]. All simulations use the `meyer-heavy.txt` noise trace from [11].

We generate different simulation topologies using an iterative approach. First, we place the gateway in the middle of the field. We then incrementally add motes to the topology by selecting an existing mote and generating a new mote location according to a two-dimensional uniform distribution, such that the new node is within communication range (RSSI > -80 dBm). This requirement ensures that the topology is connected, while still having some lossy links. Moreover, to avoid clustering of multiple nodes around the gateway, we impose a minimum distance corresponding to a RSSI of -60 dBm between any two nodes. This second requirement reflects the reality that nodes will not be placed very close to each other, to maximize spatial coverage. Figure 5 depicts two sample topologies generated through this procedure.

While simulations allow us to study different network scales and system parameters, testbed experiments provide full realism. For this reason, we use a testbed of 24 Tmote Sky motes, deployed throughout a single floor of an office building. The testbed's topology is approximately linear, matching the building's layout. Figure 6 compares the CDFs of the link RSSI values from the testbed and a 25-node simulated topology.

5.2. LPP vs. LPL

We compare two LPL implementations included in TinyOS 2.x with LPP by measuring the energy consumed during a single passive or active probe. To do so, we measure the duration and the current draw during each operation by recording the voltage drop on a 10 Ω resistor placed in series with a Tmote Sky mote. Figure 7 presents one such experiment, while Table 1 summarizes the results over all experiments. The time intervals included in that table correspond to the total duration of each operation including the time necessary to turn on the radio.

Due to the significant changes in the LPL implementations in TinyOS 2.x[4], we tested not only the one included in the current release (2.0.2) but also two versions from the previous release (2.0.1). While LPL sampling is in theory very fast (a CCA sample requires 128 μsec on the CC2420), in practice implementations sample the medium multiple times to increase robustness to random noise and transmission gaps. These gaps are generated because preambles in the case of packet-based radios, such as the CC2420, are generated by repeatedly transmitting the same packet. Furthermore, XMAC [2] (which LPL 2.0.2 and LPL 2.0.1 Ack in Table 1 are modeled after), introduces longer gaps between successive packet preambles, allowing the intended receiver to transmit an acknowledgment. From the receiver's perspective, the main difference between the three versions, as Table 1 indicates, is the amount of time that the radio's CCA circuit is used in order to reliably detect the presence of a sender's preamble.

We note that in the case of LPL we exclude the cases in which a 100 ms timeout was triggered. Such a timeout occurs when the probe mistakenly claims that there is channel activity but no packet arrives within 100 ms. During this time the node keeps its radio on waiting for a packet transmission. In our testbed these timeouts occurred in 14% of all the LPL probe experiments we performed.

Even with this unfavorable comparison, LPP is on the average only 32% more expensive than the current version of LPL. This is to be expected since LPP transmits an actual packet, while LPL only uses the radio's CCA circuit. This difference also underlies the larger variability of LPP operations compared to LPL. Nonetheless, the impact of LPP's higher cost can be managed by adjusting the probing interval, as Figure 8 indicates.

5.3. Wake-up Performance

Several factors determine the time required to wake up every node in the network: the size of the network, its topology, and the probing frequency. We evaluate the impact of the first two factors through linear topologies with varying number of nodes, while keeping the distance between neighboring nodes constant at 10 feet. For the propagation model and the noise trace we use, this distance means that on average each node can communicate reliably with its two closest neighbors, while rare transmissions from as far as ten hops away will be successful. LPP probes are sent every one second.

The upper line in Figure 9 represents the average amount of time to wake up the whole network, while the lower line represents the average amount of time a node *waits* once awake for the whole network to wake up. Thereby, the difference between the two lines represents the average time necessary to wake up a node. As expected, the wake-up time increases linearly with the network's diameter.

We also study the effect of increasing the probing inter-

[4]A comprehensive description and evaluation of the major LPL versions implemented in TinyOS 2 can be found in [14].

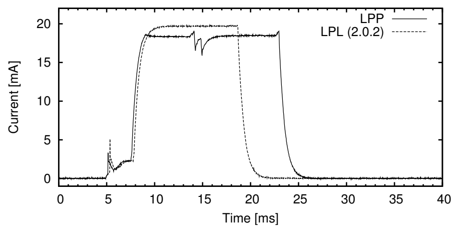

Figure 7. Current consumption during a LPL (TinyOS 2.0.2) and a LPP operation.

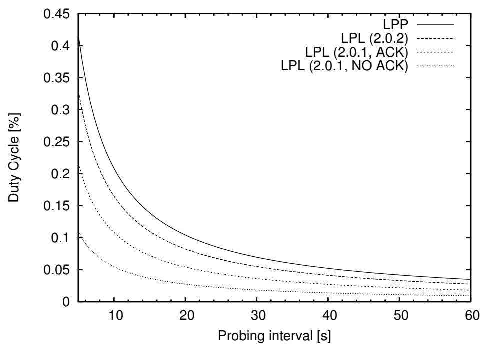

Figure 8. Node duty cycles for LPP and LPL with varying probing intervals.

Mechanism	Time (ms)			Energy (mW)		
	mean	stdev	increase	mean	stdev	increase
LPP	20.82	2.71	+26%	8.73	1.488	+32%
LPL 2.0.2	16.46	0.10		6.58	0.016	
LPL 2.0.1 Ack	10.81	0.07	-34%	3.75	0.003	-77%
LPL 2.0.1 NoAck	5.46	0.08	-66%	0.43	0.002	-97%

Table 1. Summary statistics for LPL and LPP.

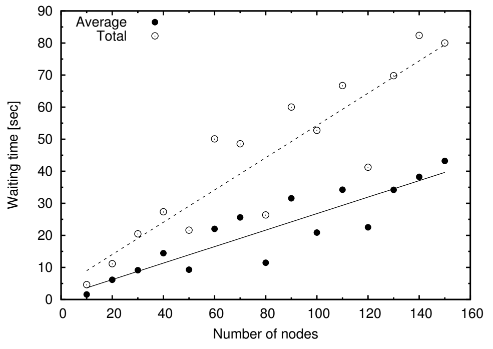

Figure 9. Network wake-up times and node waiting time as a function of the size of a linear network. Lines correspond to the best-fit linear regressions of the experimental data.

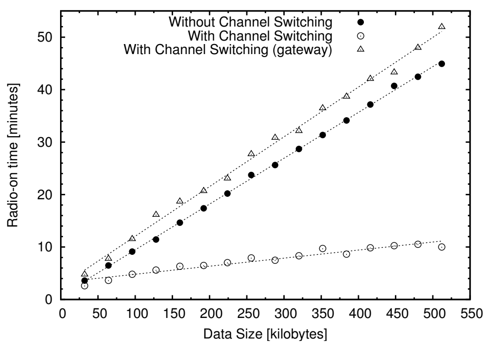

Figure 11. Average active time, with and without channel switching, for the gateway and the members of a 25-node random topology as a function of download size.

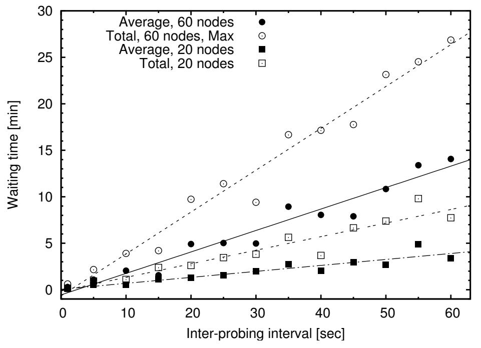

Figure 10. Relation between the network and node wake-up times and LPP probing interval for linear networks.

val on network wake up time using the same linear topologies. As Figure 10 suggests, while increasing the probing interval reduces a node's duty cycle (cf. Fig.8) it also causes network wake-up times in the order of minutes thereby neutralizing or even negating the energy savings associated with the smaller duty cycles.

Finally, we measure the time to wake up all the nodes in our testbed. To do so, we randomize each node's boot time to ensure that they wake up at different times and use one second probing intervals. Given these conditions, the average network wake-up time was 29 seconds.

5.4. Data Download

Next, we investigate the amount of time necessary to download data using Koala. We evaluate the potential benefits of channel switching and investigate the effects that download and network size have on download time. In all experiments we use LPP probing interval of 20 sec, because it provides duty cycle of 0.1% (see Fig.8).

First, we vary the amount of data retrieved during each download operation for a random topology of 25 nodes. As Figure 11 illustrates, the importance of channel switching becomes evident for download sizes larger than 32 KB. For these sizes, it pays off to switch the nodes to a different channel before doing a download because the remaining nodes can go to sleep while the download is in progress. Moreover, channel switching happens fast. We measured that on average it takes 96.34 ms to switch all the nodes on a download path for simulated linear topologies of up to 100 nodes, and 230 ms on our testbed.

While channel switching is beneficial for individual network nodes, it also increases the total time to download data from the whole network. The reason is that the gateway must wake up the remaining nodes and (re)collect neighborhood information and establish routing paths, after it finishes a download operation. This additional time is represented in Figure 11 as the difference between the total time the gateway is active when channel switching is used and the time required when channel switching is disabled (in which case the gateway as well as the motes keep their radio on for the same amount of time).

Next, we investigate the effect that network size has on the time that node radios are active. Figure 12 presents the per-node, as well as the gateway time for linear topologies ranging from 10 to 100 nodes. Inter-packet delay varies in these cases from 10.77 ms to 37.35 ms for the 100-node topologies. While some packets were lost using these delays, the average loss rate was very low ($\sim 3 \times 10^{-5}$). It is evident that download time grows linearly with network size. At the same time, downloads take longer in long linear networks, requiring the gateway to be active for up to two hours. In practice however, we do not envision that Koala will be used in such long networks. Furthermore, multiple

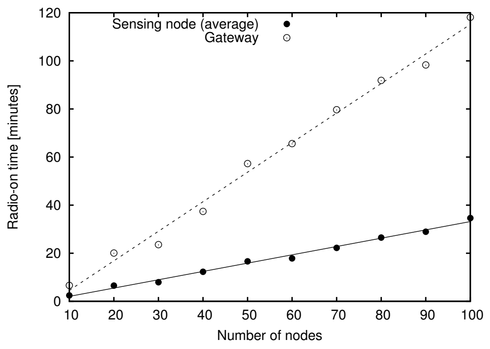

Figure 12. Average active time for the gateway and the members of a variable length linear network. The download size is 128 KB and channel switching is used.

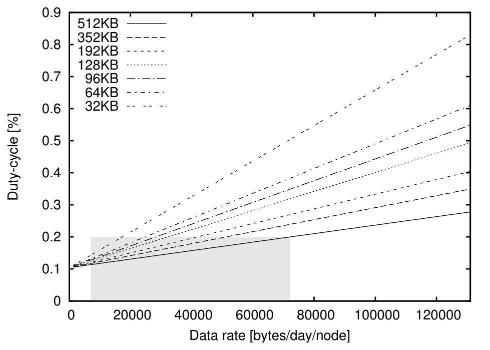

Figure 13. Overall duty cycle as a function of the per-node data acquisition rate. The gray area corresponds to the acquisition rates and duty cycle reported by Dozer [3].

gateways can be used to reduce the network's diameter.

Koala provides the flexibility of running the network at different duty cycles by adjusting the amount of data retrieved during each download operation. Intuitively, downloading larger blocks of data in a single operation is more efficient than using multiple smaller operations because the constant overhead of waking up the network is incurred only once. On the other hand, one needs to wait until the motes have collected the appropriate amount of data before a download operation can occur. Therefore, the data retrieval latency is a function of the data acquisition rate and the download size.

Figure 13 illustrates these trade-offs by presenting the overall duty cycles achieved by different download sizes as a function of the amount of data nodes collect per day. The computed duty cycles include all the system costs, including the cost of (repeatedly) waking up the network, collecting neighborhood information and installing routes, and the cost of downloading data. This is the reason why all duty cycles start at 0.1%, because this the cost of running LPP alone. Figure 13 also illustrates the duty cycles achieved by Dozer, the most efficient data gathering protocol to this date [3]. Specifically, the two vertical lines represent the amount of data Dozer nodes generate[5], while the horizontal line represents the lower duty cycle reported in [3].

While the last result suggests that Koala can achieve ultra-low duty cycles, we want to better understand how efficient it is and whether it can be further improved. We do so by presenting network-wide statistics across 20 randomly generated topologies with 25 nodes. The download size in all cases is 128 KB. The top chart of Figure 14 presents the total time that nodes keep their radios on, including time spent to wake up the network (including neighborhood collection and route setup), download data, and stay idle listening. The second chart shows the amount of time nodes transmit their *own* data, while the third chart shows the amount of time nodes transmit their own data as well as the data of other nodes. The variation in download time is due to the different inter-packet delays various nodes use, while the outliers in the amount of time spent on the download channel correspond to nodes which are close to the gateway and thus spend more time acting as relays.

Overall, the results show that nodes spend 60% of their time in the download channel forwarding data on behalf of other nodes. Furthermore, nodes spend 85% of their active time idle listening. This time includes the time nodes wait to go back to sleep while the gateway downloads data from other nodes and the time required to wake up the network. This high overhead can be reduced in two ways: first, by using a faster LPP probing rate once the network is awake and second, by optimizing the order in which the gateway performs the downloads.

Finally, the two charts on the right side of Figure 14 present corresponding results across five testbed runs. Total download times in this case are not only higher, due to the testbed's harsher operating conditions, but also more variable due to intermittent interference.

6. Related Work

Dozer [3] is the first data gathering protocol that achieves permille ($\sim 0.1\%$) duty cycles. While Koala shares the same goal, it achieves it using a radically different approach. Unlike Dozer, Koala does not require synchronized sleep schedules, nor does it require motes to persistently maintain routing trees. This strategy conserves energy, but on

[5]Dozer nodes transmit one packet every 120 seconds, corresponding to 720 packets per day. If payload size ranges from 10 to 100 bytes, then nodes generate between 7,200 and 72,000 bytes per day.

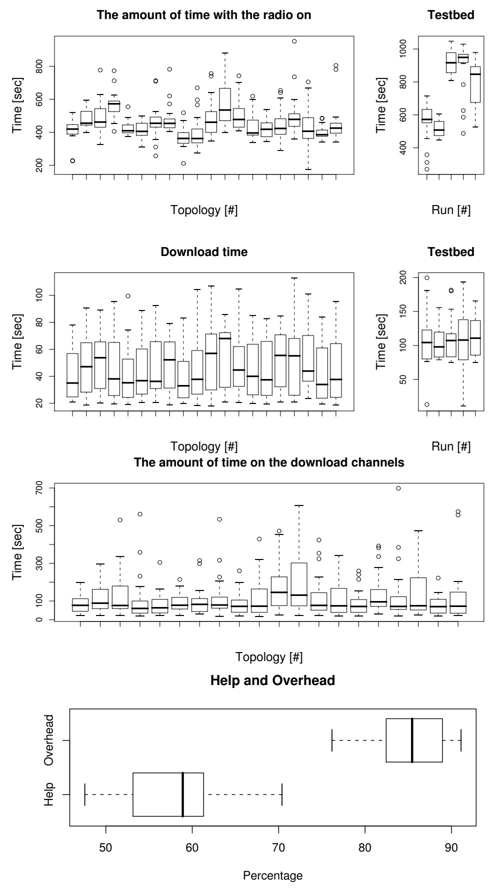

Figure 14. Activity statistics 128 KB downloads across 20 random topologies of 25 nodes.

the other hand, the synchronization allows Dozer to continuously inform the gateway about the network's health and also deliver the measurements with a much smaller delay. Centroute [22] introduced the concept of having a central node use source routes to control the routing decision in a network. However, Centroute is customized for data gathering applications, while FCP provides more general communication abstractions. Furthermore, Koala couples FCP with an efficient mechanism for waking up a network of non-synchronized duty-cycled nodes. Flush [9] recently proposed a protocol to efficiently transfer bulk data under different network scales. Unlike Flush, Koala includes mechanisms for network-wide wake up and for determining and establishing multi-hop network paths. At the same time, Koala would benefit by the rate control algorithms developed in Flush and RCRT [16].

Tenet represents a tiered architecture in which low-tier nodes (*e.g.*, motes) are driven by master nodes to perform simple tasks [7]. It showed that by constraining the actions and simplifying the decisions that sensing nodes must make, it is possible to implement functionality equivalent to existing mote-centric applications, while simplifying the mote-level code. While Koala shares Tenet's goal of simplifying the mote-level code, it does not constrain the application code that motes can run. Instead, it replaces mote-centric routing protocols (*e.g.* MintRoute [27]) with gateway-driven communications paths.

SP [18] and the Chameleon/Rime stack [4] represent two proposals for organizing the WSN network architecture. Even-though FCP implements some of the functionality covered by SP (*e.g.*, neighborhood management), they address different problems. FCP is concerned with establishing end-to-end paths, while SP is concerned with communications within the same broadcast domain. As a matter of fact, FCP can be implemented on top of SP. FCP is similar with the Rime stack but due to its tight integration with Chameleon, Rime has a much more specialized role compared to FCP.

Dutta *et al.* recently argued about the benefits of disconnected operation in data gathering sensor networks [5]. Koala presents the first architecture that realizes the benefits theorized in [5]. Finally, a number of proposals have proposed decoupling routing from forwarding in the context of the Internet [6, 8, 19]. Their focus is different: they address the scalability and consistency of the routing information, while our goal is to simplify the mote's networking stack and minimize the overhead associated with maintaining networking state.

7. Summary

We present Koala, a system for reliable bulk data retrieval from duty-cycled networks that represents a different point in the design space. We argue that decoupling the routing control and data planes simplifies the motes' software and lowers the maintenance overhead. Koala couples FCP with efficient protocols for network-wide wake up and downloading bulk data, producing an end-to-end system that can achieve ultra-low duty cycles.

Acknowledgements

We extend our gratitude to our shepherd, Philip Levis, and the anonymous reviewers for their insightful comments and their help in improving this paper.

This research was supported in part by NSF grant CNS-0546648 and by the U.S. Department of Homeland Security (Grant Number N00014-D6-1-0991) through a grant awarded to the National Center for the Study of Preparedness and Critical Event Response (PACER) at the Johns Hopkins University. 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 Department of Homeland Security and the NSF.

References

[1] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. LDP Specification. *RFC 3036*, Jan. 2001.

[2] M. Buettner, G. Yee, E. Anderson, and R. Han. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. In *Proceedings of the 4th ACM Conference on Embedded Sensor Systems (SenSys)*, 2006.

[3] N. Burri, P. von Rickenbach, and R. 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.

[4] A. Dunkels, F. Österlind, and Z. He. An Adaptive Communication Architecture for Wireless Sensor Networks. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2007.

[5] P. Dutta, D. Culler, and S. Shenker. Procrastination Might Lead to a Longer and More Useful Life. In *Proceedings of HotNets-VI*, 2007.

[6] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers. In *Proceedings of the ACM SIGCOMM workshop on Future Directions in Network Architecture*, Aug. 2004.

[7] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The TENET Architecture for Tiered Sensor Networks. In *Proceedings of the ACM Sensys Conference*, Nov. 2006.

[8] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Meyers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communications Review*, Oct. 2005.

[9] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, P. L. David Culler, S. Shenker, and I. Stoica. Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. In *Proceedings of ACM Sensys 2007*, Nov. 2007.

[10] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2006.

[11] H. Lee, A. Cerpa, and P. Levis. Improving Wireless Simulation Through Noise Modeling. In *Proceedings of the Sixth International Conference on Information Processing in Wireless Sensor Networks (IPSN'07)*, 2007.

[12] P. Levis and G. Tolle. TEP 118 Dissemination. Available at `http://www.tinyos.net/tinyos-2.x/doc/html/tep118.html`.

[13] S. Lin, J. Zhang, G. Zhou, L. Gu, J. A. Stankovic, and T. He. ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 223–236, New York, NY, USA, 2006. ACM.

[14] D. Moss and P. Levis. BoX-MACs: Exploiting Physical and Link Layer Bounrdaries in Low-Power Networking.

[15] R. Musăloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life Under your Feet: A Wireless Sensor Network for Soil Ecology. In *Proceedings of the 3rd EmNets Workshop*, May 2006.

[16] J. Paek and R. Govindan. Rate-Controlled Reliable Transport for Sensor Networks. In *Proceedings of the ACM Sensys*, 2007.

[17] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM.

[18] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A Unifying Link Abstraction for Wireless Sensor Networks. In *Proceedings of ACM SenSys*, Nov. 2005.

[19] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang. Network-wide decision making: Toward a wafer-thin control plane. In *Procedings of the ACM SIGCOMM HotNets Workshop*, Nov. 2004.

[20] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. LUSTER: Wireless Sensor Network for Environmental Research. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2007.

[21] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, Apr. 2003.

[22] T. Stathopoulos, L. Girod, J. Heidemann, and D. Estrin. Mote herding for tiered wireless sensor networks. Technical Report CENS-TR-58, University of California, Los Angeles, Center for Embedded Networked Computing, December 2005.

[23] R. Szewczyk, A. Mainwaring, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proceedings of SenSys 2004*, Nov. 2004.

[24] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a Sensor Network Expedition. In *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN '04)*, Jan. 2004.

[25] Texas Instruments. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Available at `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf`, 2006.

[26] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. A Macroscope in the Redwoods. In *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2005.

[27] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges in reliable multihop wireless sensor networks. In *Proceedings of ACM Sensys 2003*, 2003.

[28] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of IEEE INFOCOM 2002*, 2002.