A Unifying Link Abstraction for Wireless Sensor Networks

Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao,
David Culler, Scott Shenker, and Ion Stoica

Computer Science Department
University of California, Berkeley
Berkeley, CA 94720

International Computer Science Institute
1947 Center Street. Suite 600
Berkeley, CA 94704

ABSTRACT

Recent technological advances and the continuing quest for greater efficiency have led to an explosion of link and network protocols for wireless sensor networks. These protocols embody very different assumptions about network stack composition and, as such, have limited interoperability. It has been suggested [3] that, in principle, wireless sensor networks would benefit from a unifying abstraction (or "narrow waist" in architectural terms), and that this abstraction should be closer to the link level than the network level. This paper takes that vague principle and turns it into practice, by proposing a specific unifying sensornet protocol (SP) that provides shared neighbor management and a message pool.

The two goals of a unifying abstraction are generality and efficiency: it should be capable of running over a broad range of link-layer technologies and supporting a wide variety of network protocols, and doing so should not lead to a significant loss of efficiency. To investigate the extent to which SP meets these goals, we implemented SP (in TinyOS) on top of two very different radio technologies: B-MAC on mica2 and IEEE 802.15.4 on Telos. We also built a variety of network protocols on SP, including examples of collection routing [53], dissemination [26], and aggregation [33]. Measurements show that these protocols do not sacrifice performance through the use of our SP abstraction.

Categories and Subject Descriptors:
C.2.2 [Computer-Communication Networks] Network Protocols,
D.4.7 [Operating Systems]: Organization and Design

General Terms: Design, Experimentation, Standardization

Keywords: Protocol architecture, link protocols, network protocols, wireless sensor networks, network abstractions

1. INTRODUCTION

Wireless sensor networks (hereafter sensornets) pose many networking challenges. These challenges have motivated a broad set of investigations, which have given us a cornucopia of possible protocols at each level in the system. Many physical links, with widely differing characteristics, have been utilized ([2, 18, 47, 48, 49]). Myriad low power media access protocols have been developed, based on CSMA ([35, 41, 52]), TDMA ([10, 37, 50, 56]), or both ([15, 38]). Numerous topology formation algorithms ([1, 14, 55]), routing protocols ([34, 53]), aggregation algorithms ([27, 33]), and dissemination protocols ([14, 16, 22, 26]) have been proposed.

The extreme resource scarcity of sensornets requires minimizing energy usage while maintaining high reliability and data quality over time-varying and noisy links ([53, 57]). To meet these ambitious goals, most research efforts have emphasized performance more than modularity, with many issues (such as power management, scheduling, and data buffering) handled simultaneously at many levels in a deeply intertwined fashion. As a result, the field has produced a few vertically integrated designs, each with their own interface assumptions, and there is little code reuse.

The response to this situation has been a call for a *sensornet architecture* [3]. Such an architecture would provide much greater modularity to sensornet designs, thereby regularizing assumptions about interfaces, encouraging code reuse, and fostering greater intellectual synergy.[1] Moreover, if the architecture has a "narrow waist" (as does the Internet architecture), then it could effectively decouple many aspects of the software from the underlying hardware. Such a decoupling would be of great benefit given the rapid technological advances in the sensornet arena, particularly in radio transceivers. The authors of [3] make the case that, in contrast to the Internet, the narrow waist (which they call the sensornetwork protocol, SP), not be at the network layer but instead sit between the network and link layers. This "lowering of the waist" is necessary because processing potentially occurs at each hop, not just at the end points, and there are many application-specific multipoint communication patterns (collection, aggregation, dissemination, etc.). As observed in [3], one cannot base the sensornet architecture around the end-to-end delivery of packets, but instead must build upon the lower-level base of best-effort single-hop communications.

The key challenge for SP is providing adequate insulation between the hardware below and the various communication abstractions above while still providing adequate efficiency. It should allow network level protocols to optimize for the underlying link in terms of the characteristics expressed at SP, rather than knowing which particular link and physical layer resides beneath.[2] Moreover, SP must allow network protocols to choose neighbors wisely, taking into account information available at the link layer.

[1]Zigbee proposes a classic layered architecture, but each layer assumes a specific instance of the surrounding layers; *e.g.,* the routing layer assumes the IEEE 802.15.4 link and physical layers. An architecture built on static technologies is destined for obsolescence.
[2]This is in contrast to IEEE 802.2 [46] which provides a uniform syntactic interface to various link layers, but the code above takes specific actions based on the particular protocol beneath, whether it is ethernet, 802.11, and so on.

Rather than strict, opaque layering as used in the Internet, SP must be *translucent*; it should provide enough information so that the network and link layers can cooperate to achieve effective use of communication resources, but it must do so in a way that is independent of the link layer. To be useful, SP must be easy to program against, providing network protocols with a few simple choices rather than a bewildering set of hard-to-configure parameters.

While [3] advocates the existence of SP, it does not propose a concrete service, interface,or protocol. In this paper, specifically Section 2, we describe our preliminary design for SP. We translate the general notion of a decoupling interface for wireless sensor networks into a concrete proposal for a unifying link-level abstraction. Multiple network protocols cooperatively optimize with link protocols through our abstraction. This abstraction is novel in how it promotes cooperation across the link and network layers to utilize limited resources efficiently.

Our SP design is simple, giving network layer protocols only one bit to express their desires about the urgency and reliability[3] of a message. At the same time, SP allows link layers and network protocols to cooperate by maintaining and exposing a shared neighbor table and message pool.

It takes many years of hard use to evaluate an architecture, since its goal is not to excel in one particular circumstance but instead to perform adequately in a wide range of scenarios. Our design of SP was based on our experience with a variety of link and network protocols, and we have done the thought-experiment of asking how one might use them above and below SP (Section 7). While these musings were encouraging, they are hardly convincing. To provide more concrete evidence, we report in this paper on a smaller set of cases that we have implemented and measured.

We show the effectiveness of SP through a TinyOS implementation on two very different link layers. We demonstrate that a single implementation of a network protocol yields power-aware operation on links with differing power management strategies. In fact, we show that optimizations fall out naturally by providing a unifying abstraction that we have not seen implemented in monolithic approaches. Finally, we show multiple network protocols gaining mutual benefit by cooperating on a common link abstraction.

2. SP DESIGN

Most deployed sensor networks consist of dense patches of wireless nodes, where each patch is connected to the Internet via one or more gateway nodes, either directly or through some form of transit network [5, 13, 43, 44]. Unlike the Internet, aggregate communication is prevalent in sensor networks, whereas point-to-point communication is rare. Although each patch is often an edge network, it must address many of the issues associated with the Internet. Each node potentially operates as a data source, data sink, and/or router. In addition to generating sensor data, nodes must form and maintain routes, and forward traffic. The nodes must achieve this despite noisy, time-varying, and even intermittent connectivity.

These challenges have motivated a wide range of work at the physical, data link, media access, and network layers. While there are many options and possibilities at each layer, they are rarely interchangeable. Instead, resource constraints, power management, and application specific processing have forced many studies to make numerous assumptions about the surrounding networking abstractions. The variations in these assumptions prevent integrating individual advancements into a complete solution. For example, the

[3]Reliability refers to a best effort transmission of the message, not a guarantee that it will be received by the intended recipient(s).

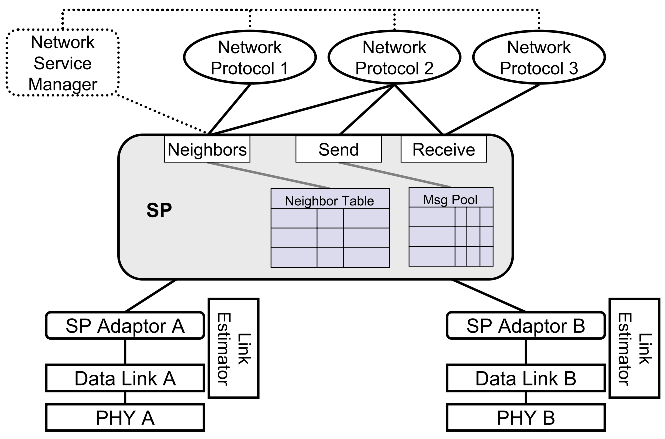


Figure 1: Conceptual view of SP architecture. Network services interact with various link protocols through SP's shared neighbor table and message pool.

PSFQ transport protocol assumes that nearby nodes can overhear transmissions [51], while TDMA based MAC protocols, such as IEEE 802.15.4 [49], S-MAC [56], T-MAC [50], and TRAMA [37] can make this an invalid assumption.

The goal of SP is to provide a unified interface to a wide range of data link and physical layer technologies that allows network layer and above protocols to operate efficiently through link independent optimizations. By providing a unified interface, SP can offer a number of advantages over integrated approaches. In particular, it allows multiple network protocols and link technologies to coexist and evolve independently of each other in the same way the IP layer allows transport protocols and link layer technologies to evolve independently in today's Internet.

There is one other important advantage of having the SP layer positioned under the network layer, that is, it makes SP equally relevant and useful to both the single-hop and multi-hop networks. Thus, the design of SP is at large agnostic to whether multi-hop or one-hop sensor networks become prevalent in the future.

Although we considered a variety of alternative design points, this section describes the approach that we see as most effective in achieving this goal. Experience has shown that current single and multi-hop protocols cannot efficiently operate independently of each other: they must share information. The layers above and below must be decoupled, but must also cooperate. The principal design challenge in SP is defining how they cooperate in a simple but expressive way. This section describes a particular set of concepts that form an SP abstraction. The abstraction could be implemented in virtually any operating system, but the description would not be complete without describing its interface for some meaningful execution model. We ground the SP concepts with a concrete implementation on the TinyOS operating system [25].

2.1 Description

Figure 1 shows the general SP architecture: SP bridges the link and network layers by providing link independent abstractions to build efficient network protocols. Multiple network protocols coexist on a node. Each network protocol is identified by a protocol id, similar to a protocol type in IP or an AM identifier in TinyOS. Unlike the Internet where there is only one network protocol (IP), SP supports many network protocols which implement a variety of functions, such as collection for data delivery, dissemination for code updates, aggregation, and others. The primary goal of SP is to enable multiple network protocols to coexist and work efficiently. The SP abstraction may be implemented on a variety of link technologies that expose different physical technologies, encodings, framings, media access mechanisms, collision avoidance protocols, and power management mechanisms. A node employs one or more link technologies, depending on its hardware capabilities.

SP performs three main operations: (1) data transmission, (2) data reception, and (3) neighbor management. Data transmission and reception are message oriented, with a variable message length. Underlying link protocols dictate a maximum data unit (MDU). Network protocols may operate relative to the link's MDU by querying its size through SP. Next, we discuss the three operations performed by SP.

Data Reception: A message arriving on a link interface is dispatched to its associated network protocol. Optional message filtering may occur at or above the SP layer and discard messages not destined for the node's local address or for the broadcast address. SP takes no position on higher level naming and scheduling issues other than that nodes have an address on each link interface.

Data Transmission: This operation is implemented using a shared *message pool* data structure at the SP layer. Network protocols submit messages to the pool for transmission. Messages may consist of multiple packets; however, network protocols only dispatch each packet to SP when the link is available. Messages are specified with control information for lower layers, such as reliability and latency requirements. The pending messages in the pool may be inspected by the link layer and other network protocols that optimize their behavior based on the pool's content. After transmitting a message, SP provides feedback to the network protocol. This feedback includes various information, such as congestion status, that may help the network protocol to optimize its behavior.

Neighbor Management. SP allows the link layer and network protocols to cooperatively maintain an effective summary of useful immediate neighbors. This is achieved through a *neighbor table* data structure, which maintains information about link quality and power scheduling (when the node should be awake or asleep). SP mediates the interactions between network protocols and one or more specific links. Rather than a rigid separation of these layers, SP allows network and link layers to cooperate through its neighbor table and message pool structures.

Below, we present the two main data structures maintained by the SP, the neighbor table and the message pool, in more detail.

2.2 Neighbor Table

Typically, network protocols maintain information about their neighbors in order to make informed decisions for routing, aggregation, and dissemination. Similarly, the link layer maintains information about the state of the link to particular neighbors. The mutual interest in neighbor-related information has often led to monolithic designs. For example, MintRoute [53] in the TinyOS distribution combines link reliability information for its direct neighbors with path metrics (*e.g.,* hop count, expected path cost) for routing to a root node. Likewise, slotted link protocols presented in [37, 50, 56] monitor neighbors to maintain synchronization and connectivity. Whereas network protocols sometimes include link functionality, such as in MintRoute, and link protocols like S-MAC sometimes include network functionality, UNPF [4] proposes a single unified layer must include both link and routing information. Such a monolithic approach is not suitable for enabling innovation at the network layer in sensornets.

As sensor networks mature and multiple network protocols coexist, it becomes increasingly attractive to share information among various network protocols, rather than require each of them to maintain its own table. The *neighbor table* is the main repository of this shared information. It enables cooperation between network protocols and the link layer, and allows SP to decide when to listen, receive, transmit, and sleep. The insertion and eviction of entries in the neighbor table are deferred to network and link protocols to cooperatively decide which entries belong in the table.

An entry in the neighbor table usually consists of the address of the neighbor, link quality, and scheduling information. For added flexibility, the table is extensible—network services and link protocols may add columns to it, such as routing gradients or coordinates. Entries in the neighbor table are indexed by a combination of destination address and network interface: SP assumes that all nodes accessible through a given link interface have unique link addresses. The format of the addresses are not specified by SP to allow different link addressing modes.

SP requires scheduling information in the neighbor table indicating when each neighbor is expected to be awake and asleep. Since power is the critical resource, both network and link protocols use neighbor schedule information to determine which actions to take and when these actions should be performed. However, the horizon of this information within SP is limited. When the known communication schedule of a neighbor expires, SP asks the network and link layers to determine a new schedule. For example, a slotted MAC layer may respond with the next beacon slot whereas a rendezvous-based network protocol may respond with the next meeting time.

Studies have shown that in many cases the set of candidate neighbors (*e.g.,* recently heard nodes) is much larger than the set of useful neighbors (*e.g.,* neighbors which can provide a reasonable reliable link) and too large to retain in the memory of most microcontrollers [53, 57]. Thus, neighbor table management is critical. SP enforces as little policy as possible on neighbor management; instead, it implements management mechanisms that are applicable for a wide range of uses.

If the link detects a new neighboring node, it notifies SP. SP asks each network protocol whether the node should be added to the table. When a neighbor's scheduled active period has expired, network and link protocols are notified so that they can update the neighbor's schedule information. When a node is evicted by a protocol, all other network protocols are notified of the eviction. When multiple network protocols are present, rather than define the resource sharing policy, SP depends on the presence of an optional *network service manager* to mediate resource conflicts.

2.3 Message Pool

The *message pool* allows network protocols to request message transmissions. The transmission interface enables the network protocol to exert a degree of control over lower level message processing, and provides feedback from the link layer.

A key design issue of the message pool is how much information to expose to network and link protocols. The more visibility the link layer has on potential future transmissions, the better it can schedule traffic to reduce energy and avoid contention. For example, a slotted or TDMA link might want to transmit messages with different destinations in the same slot, whereas a CSMA link with preamble sampling might want to batch messages to a given destination so that a single long preamble can wake up the destination for the entire batch. These link optimizations motivated our decision to use a pool. As evident from this discussion, the network protocols should give the link layer enough flexibility to schedule traffic efficiently. Storage available for the message pool is limited, and thus the decisions should be made in a timely fashion.

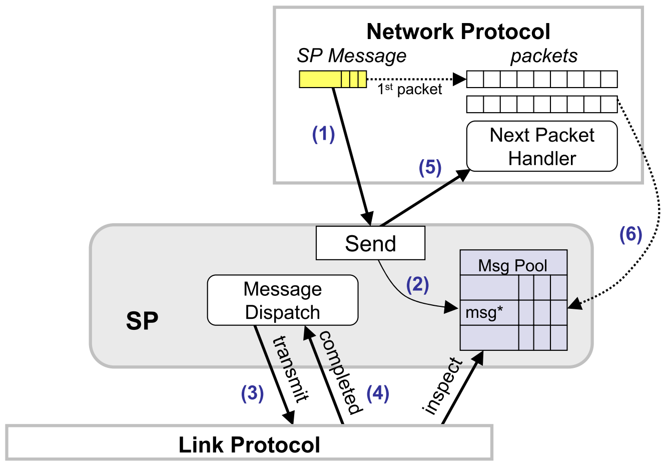


Figure 2: The SP send process stores an SP message in the pool, schedules it for transmission, and then requests the next packet in the SP message. Note that all message and packet storage is created by the network service advocating a "pay for what you use" policy.

The message pool contains references to messages, which can be accessed out of order. The messages are not stored in the SP layer; they are either stored in the network or the link layers. A message may consists of multiple packets. Similar to lazy task creation [32], packets are only handed off to SP when resources and the underlying communication medium become available.

Each message pool entry contains a reference to the next packet to be sent, the number of the remaining packets in the message, and notification when the next packet should be materialized. The concept of specifying the number of packets that a service intends to send instead of the actual packets is called *message futures*. Message futures only require that the next packet in a message be realized and passed to SP when requested. SP only asks for messages serially, and only after the current packet has been transmitted. By not initally specifying all of the packets in a sequence, services may allocate less message buffers and reduce memory and processing resource usage by only filling buffers when needed. Message futures are useful for a wide range of network protocols. Code propagation may only load the next code page when needed. Applications may store sensor data in flash that is retrieved and transmitted during rendezvous periods with a node's parent.

Figure 2 shows the operation of sending a message using SP. An SP message is submitted to SP (1), whose pointer is added to SP's message pool (2). SP decides when it is appropriate to send the first packet based on batching and link protocol inspection of the neighbor table (3). After the transmission completes (4), SP requests the next packet in the SP message (5). The SP message pointer is updated to point to the next packet buffer (6).

The message pool allows network and data link protocols to pass message information to each other. The send interface presents a packet buffer to SP along with simple indicators of latency sensitivity and need for reliability on the transmission. SP facilitates bi-directional exchange of control information through this control and feedback information. A network protocol can indicate that a particular message entry is latency critical by setting an *urgent* bit, which informs the link layer to treat it as high priority or to send it soon even if extra energy is expended to do so. The level of effort that should be expended transferring a message is indicated by a *reliability* bit, which informs the link layer to acknowledge and retransmit a message for a predefined number of times. The control mechanism provides guidance to lower layers which will attempt to optimize for power and channel efficiency.

After transmission, lower layers send feedback to the network protocol to adjust its behavior. For instance, if a message requested reliable transmission, SP notifies the network protocol if the desired level of reliability has been achieved. The link layer could inform the network protocols that the transmission rate is too high to sustain by setting a *congestion* bit. In another example, consider a typical sensor network application that generates traffic at regular intervals. Even at low duty cycle, this traffic pattern can result in high contention if sensor samples are highly correlated. One solution to this problem is to stagger the data transmissions across neighboring nodes. SP support staggering by sending *phase shift* feedback indicating when such a shift would be beneficial. SP provides a delta time recommendation for future traffic to prevent correlated behavior in sensornet applications. For example, the nodes may take samples at time correlated instances and then transmit them. Phase recommends to the network layer that if it sent its message at a different time, it may achieve less congestion or greater power savings. If SP detects congestion, SP queries the link protocol for a suitable phase shift. The phase shift is added to the delay incurred between submission of the message to the pool and the actual transmission. Phase shift recommendations are determined by the semantics of the link protocol; two mechanisms are to count the number of backoff events on a CSMA link or count the number of reserved slots in a TDMA protocol.

2.4 Discussion

Our current design of SP emphasizes minimalism: SP includes only the set of features that we considered absolutely necessary to develop applications in a sensor network environment. Primarily, these features follow from the need to balance the application requirements, on one hand, and to efficiently use the available resources, on the other hand. Given the extreme scarcity of resources in a sensor network, achieving efficient resource utilization is not only desirable, but *necessary*. This is the main reason for which the SP interface is necessarily more complex than the IP interface (the corresponding "narrow waist" of the Internet).

SP provides three functionalities that are only partially supported, or not supported at all by IP: (1) allows the link layer to provide congestion indication and schedule hints (*e.g.,phase shift* feedback) to the network protocols, (2) allows a network protocol to request an urgent and/or reliable service, and (3) enables network protocols and the link layer to share the link information.

Network protocols use the congestion indication and schedule hints received from the link layer to schedule its future transmissions in order to optimize resource utilization. For example, upon receiving a congestion indication, the network protocol can slow down to reduce the probability of message loss, or aggregate traffic to reduce the number of transmissions. Note that unlike IP where the congestion is signaled end-to-end, with SP the congestion is signaled at every hop. Architecturally, this design decision is justified by the fact that, unlike an IP router whose main role is to forward packets, a typical node in a sensor network runs also application code, which can process data locally, if needed.

In order to optimize for the energy usage, the data link layer needs to have knowledge about the application's delay and reliability requirements. SP allows network protocols to provide this information by associating a priority and a reliability bit with each message (see Section 3 for details). Whenever these bits are not set, SP in conjunction with the link layer aggressively optimize for energy usage by batching packets whenever possible.

```
interface SPSend {
  command result_t send(sp_message_t* msg);
  event void sendDone(sp_message_t* msg, result_t success);
  event TOS_MsgPtr nextSend(sp_message_t* msg, uint8_t pos);
  command result_t changed(sp_message_t* msg);
  command result_t cancel(sp_message_t* msg);
}
```

Figure 3: The SPSend interface.

Finally, allowing network protocols and the link layer to share link information eliminates the need for each network protocol to maintain its own neighbor table. The advantage of sharing this information in the context of resource management is twofold: it reduces the storage requirements, and avoids redundant measurements to estimate the link quality.

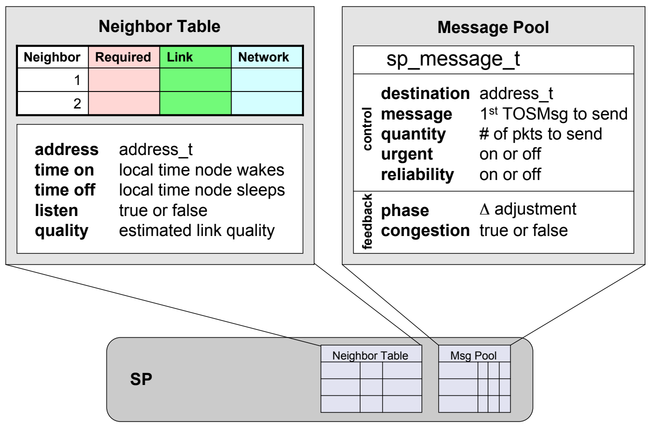


Figure 4: SP provides neighbor table and message pool structures. The required entries of SP's neighbor table are shown on the left, while the structure of SP messages is shown on the right.

```
interface SPNeighbor
{
  // Query and iterate through neighbors
  command sp_neighbor_t* query(uint16_t address);
  command sp_neighbor_t* get(uint8_t i);
  command uint8_t max();

  // Adding, admitting, updating and removing neighbors
  command result_t insert(sp_neighbor_t* neighbor);
  command result_t remove(sp_neighbor_t* neighbor);
  event void update(sp_neighbor_t* neighbor);
  event result_t admit(sp_neighbor_t* neighbor);

  // Expiry (update with new timing) and Eviction
  event void expired(sp_neighbor_t* neighbor);
  event void evicted(sp_neighbor_t* neighbor);

  // Adjust n's link quality based on message m
  command result_t adjust(sp_neighbor_t* n, TOS_MsgPtr m);

  // Listen to the specified neighbor on next wakeup
  command result_t listen(sp_neighbor_t* neighbor);

  // Try to find more neighbors
  command result_t find();
  command result_t findDone();
}
```

Figure 5: The SPNeighbor interface.

3. IMPLEMENTATION

To evaluate the feasibility of our approach and to make the proposal concrete, we implemented our SP abstraction in TinyOS. We discuss how SP is implemented, including the neighbor table, message pool, and minimal set of commands and events to build network protocols. The implementation is quite lean and completely event driven. Three types of events trigger SP to act: message receptions and other link protocol events, network layer commands, and internal timer events.

Network protocols issue send requests to SP using the `SPSend` interface, shown in Figure 3, which takes a single parameter, a pointer to an `sp_message_t` structure. Figure 4 shows a partial version of the `sp_message_t` structure (it has an additional 7 bytes of state, such as the time the message was submitted to SP, used for internal bookkeeping and metadata). The send command places a reference to the SP message in the message pool and SP schedules it for transmission. SP implements the reliability and urgent control bits. Urgency is treated as a priority mechanism—SP services urgent requests before others—but also is used to override the default power management schedule. Extra energy may be invested to wake up the destination in order to transfer the message quicker. Messages that are not marked as urgent are held in the message pool. If the neighbor has a known wakeup period or if other traffic for that neighbor is received, the SP attempts to send the pending data. Otherwise, after the message has been waiting in the pool for longer than a specified timeout, SP tries to send the message more aggressively in the same manner as urgent messages (but with less overall priority). If reliability is set, SP will use acknowledgments, retransmissions, or whatever mechanisms the underlying link provides to deliver the message.

When SP has completed transmission of the message, it signals the `sendDone` event informing the associated network protocol. If SP does not succeed in acknowledging delivery of a message marked reliable, it resets the reliability field to false to give feedback to the higher layer. Congestion tells the network layer whether the link layer observed a congested channel when the message was sent. Congestion allows saturation conscious protocols (such as floods) to react accordingly or reduce their message generation rate. Note that the threshold for setting the congestion bit may be different (and is higher in our implementation) than the threshold for providing a phase shift to network protocols.

The `SPSend` interface allows the network protocol to use message futures by setting the number of packets that are ready to follow the current one. If this count is non-zero, SP can signal the *nextSend* event to cause the network protocol to materialize the next packet. It may be generated from application data, data in EEPROM, or from a higher level buffer. Without imposing a large amount of RAM pressure, this allows the link layer to burst packets when it has opportunity to do so, or to schedule packets into upcoming slots. Alternatively, network protocols can verify if reliably transmission was achieved and choose to retransmit or change the SP message's parameters.

Network protocols can modify the status of on-going message streams using the `change` and `cancel` commands. After a message is submitted, the network protocol may cancel the outgoing message. Activities at the network layer may cause the contents of the message to change. For example, a routing protocol may receive a new route beacon that causes its parent to change. The network protocol may reset the destination of its pending messages and notify SP using the `change` command. Cancel removes a message from the pending message pool. It is particularly important for suppression in many dissemination protocols.

SP provides the `SPNeighbor` table interface presented in Figure 5 to allow maintenance of the neighbor table by the link and network protocols. Figure 4 shows the default TinyOS SP neighbor table. It provides five pieces of information: address, time-on, time-off, listen, and link quality.

Time-on and *time-off* fields indicate when the neighbor can receive messages in terms of local node time so that link layers can minimize idle listening. Network protocols can incorporate link constraints in generating communication schedules using timing information. For a fully active CSMA-based MAC, time on is always "now" and time off is always "never." In contrast, for a TDMA-based MAC, the values correspond to the next time slot and its duration. Our SP implementation favors sleeping over listening in order to reduce the node's duty cycle. If a message is pending for a neighbor, SP will wake up and send it according to the neighbor's time-on in the table. If no message is pending, SP will remain asleep, regardless of when that neighbor is awake. Therefore, the time-on and time-off fields are indicators of when the remote node is listening. In the opposite direction, if a neighbor's *listen* bit is set, SP will explicitly wake up and receive from that neighbor. This allows bi-directional low power communication. An example use of the listen bit is dissemination in a tree topology. Nodes periodically send to their parent during their parent's active period. Parents can broadcast data down the tree to their children during their active period. If the listen bit is not set, children will not wake up and receive from their parent if they do not have any pending messages to send to the parent. The listen bit also permits network protocols to snoop on traffic from other nodes.

SP uses a cooperative scheme to manage neighbor table membership and does not enforce policy on the neighbor table contents. When a protocol requests that a neighbor be added to the table, it calls the `insert` command. SP queries all other protocols with the `admit` event—if any of the protocols indicate an interest then the neighbor is added to the table. The `admit` event allows protocols to determine which action to take, including which entry to evict. Since SP generally does not have enough information to determine neighbor schedules on its own, it signals both network and link protocols on neighbor expiration to attain wakeup schedules using the `expired` event. For example, if a TDMA MAC or a higher level scheduling protocol knows when the neighbor will next be awake, it can update the entry correspondingly. To remove an entry, protocols call `remove` which nullifies the entry and signals an `evicted` event to link and network protocols.

Protocols may scan the neighbor table using the iteration commands found in the `SPNeighbor` interface. Protocols query the `max` command to query the maximum number of neighbors in the table and then use `get` to retrieve the neighbor. After receiving a message, protocols may request that the link adjust a neighbor's link quality entry through the `adjust` command by passing the neighbor table entry and received message. The link uses its link estimator to update the neighbor entry.

If the entries in the neighbor table are sparse, protocols may request that SP find new neighbors. The underlying `find` command may be implemented in numerous ways—either through active probing, passive scanning, or enabling channel sampling.

Our implementation includes a special reserved broadcast neighbor entry. The broadcast entry is used for the link and network protocol to inform SP of times when it is safe to send to the broadcast address. The shared broadcast entry allows network wide synchronized wakeup and local cell broadcast slots to be implemented with the same framework.

Protocols may add columns to the neighbor table for additional neighbor state. In our implementation, columns may be redefined by manual editing of the table entry structure at system build time. (Alternatively, the newly emerging nesC feature of attributes [11] may simplify this process.) The `update` event allows protocols to enter initial data into non-standard neighbor entries upon admission to the table, or after updated by other protocols. Neighbor data is updated through the `insert` command—if the neighbor is already in the table, its values are simply refreshed and an `update` event is signaled.

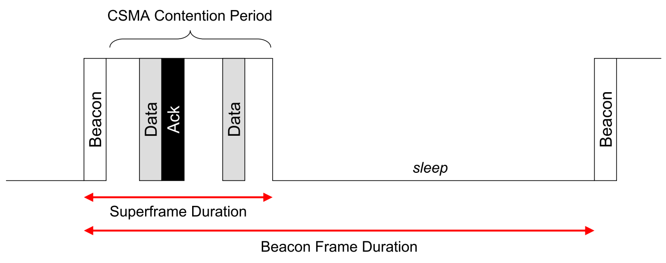


Figure 6: An IEEE 802.15.4 superframe consists of a MAC beacon message followed by a CSMA contention period for other traffic. The duty cycle is bounded by superframe to beacon frame ratio.

4. LINK PROTOCOLS

The novel constraints of sensornets—particularly, energy conservation—have led to many proposed protocols for media access control (MAC). These protocols fall into two basic classes: *slotted protocols* and *sampling protocols.* In slotted protocols, nodes divide time into discrete intervals (slots) and schedule whether the radio is in receive mode, transmit mode, or powered off in terms of these slots. Synchronizing slots with neighbors allows nodes to only power the radio on when needed, significantly reducing idle listening. Slotted protocols are often rigid; after they establish a schedule, a node can usually only communicate with other nodes on the same schedule. Short communication periods can lead to increased contention, plus synchronization maintenance costs both power and bandwidth. Slotted protocols include the TDMA family of protocols [12, 21, 39, 42], IEEE 802.15.4 [49], S-MAC [56], T-MAC [50] and TRAMA [37].

The second class, sampling protocols, take a different approach. Rather than coordinate time slots, nodes periodically wake up, and only start receiving data if they detect channel activity. Depending on the underlying physical layer, this detection can either be based on channel energy or successful symbol decoding. Periodic channel sampling allows a node to conserve energy by keeping its radio off most of the time. In contrast to slotted protocols, sampling protocols are very flexible: a node can communicate with any other node within its radio range. Flexibility comes with a cost, however. Unlike slotted protocols, which send regular data packets, sampling protocols must send long, expensive messages to wake up a neighbor. Examples of sampling protocols include Aloha with preamble sampling [6], B-MAC [35], WiseMAC [7].

To explore whether SP can be an effective abstraction for both of these classes of MAC protocol, we implemented it on an example protocol from each. For a slotted protocol, we chose IEEE 802.15.4 on the Telos [36] platform; for a sampling protocol, we chose the standard TinyOS mica2 networking stack (B-MAC on top of the Chipcon CC1000 radio [35]). We present each protocol and explain how the SP abstractions map to their capabilities.

4.1 Slotted Protocols

We used the IEEE 802.15.4 protocol (referred to as "15.4" hereafter) as our reference slotted protocol due to its widespread availability and use by Zigbee. 15.4 supports star and peer topologies; for this study we chose to only use the peer topology (every node is a "coordinator") since it maps more closely with existing sensor network protocols. Since each node acts as a coordinator, it periodically sends a beacon message with its schedule. Neighboring nodes receive and synchronize with that beacon. Figure 6 shows an example beacon period. In order to find beacons of neighboring nodes, 15.4 provides a `scan` command.

SP allows the 15.4 MAC to control its own beacon schedule. Each time a beacon is received, the timing information from that beacon is inserted into SP's neighbor table. When the beacon period expires, SP asks the link and network protocols to renew the expired entry—in this case, 15.4 updates the entry to the next expected beacon period for that neighbor. Each time a neighbor's beacon time arrives, SP checks if it has messages to send to that neighbor or if the `listen` bit is set. If either is true, SP instructs 15.4 to listen to the channel until the end of the beacon period. When the period expires, SP tells 15.4 that it has finished listening to the channel. This mechanism allows network protocols to listen during periods that are not associated with beacons and passes link wakeup information up the stack.

For broadcast messages, SP uses the broadcast neighbor table entry to determine if it can send to the broadcast address. If no information about the broadcast communication period has been recorded, our SP implementation sends the message using unicast by cycling through all the known neighbors. Alternative SP implementations could explicitly establish a broadcast slot.

If the neighbor table population is sparse, network protocols may request SP to find new neighbors. On 15.4, SP requests a beacon scan, which will find any neighbors with 15.4 beacons. Any protocol (network, link, or SP) may halt the neighbor scan.

Link estimation is performed by using the 15.4 LQI metric. The link quality indicator (LQI) is calculated from a correlation value that all 15.4 radios are required to provide to the MAC protocol. Any time that a service receives a message, it can ask SP to adjust the quality of a neighbor based on the received message. SP then asks 15.4 to compute the link quality which is updated in the neighbor table. When reliability is requested for a message, SP enables 15.4 link layer acknowledgments. If an acknowledgment fails, SP retries the message up to a threshold.

4.2 Channel Sampling Protocols

We used the default mica2 MAC protocol for our implementation of SP above a sampling protocol. The "Low Power Listening" (LPL) mechanism (part of B-MAC) and protocol hooks are described in [35]. Each node wakes up, samples the channel for activity, and returns to sleep. We added information to the MAC header in B-MAC to support synchronization and source addressing for neighbor state maintenance.

When a packet is sent with a long preamble, synchronization information is extracted and the neighbor is inserted with its LPL sampling schedule. If a message is destined for that neighbor, SP will wake up the radio prior to the sampling time of the remote host and transmit the data with a short preamble. The neighbor will wake up and sample the channel at its normal interval, detect activity, and receive. If the destination is an unknown neighbor or broadcast address, the packet will be sent with a long preamble to wake up all surrounding nodes.

Since SP maintains a message pool, it can "piggyback" data after receiving messages from neighboring nodes. If a neighbor sends a packet with a long preamble, other nodes that receive the long packet may transmit their packets with short preambles immediately following the long preamble packet. If a message is not urgent, SP takes advantage of piggybacking by waiting for others to send a long preamble. Collisions are mitigated by a small random backoff while piggybacking. If a single node has multiple packets to piggyback, our B-MAC implementation continues to transmit the preamble symbol, thereby holding the channel, while asking SP if there are additional packets to send.

Like 15.4, if reliability is requested, SP enables B-MAC's link layer acknowledgments and retries if a message send is unsuccessful. After a few unsuccessful tries, SP uses long preambles to try to communicate with the neighbor more aggressively. For broadcast packets, SP will either piggyback on a long preamble packet or send with a long preamble if the message is past due.

Two forms of neighbor estimation are included with our SP implementation. A basic RSSI link estimator provides coarse information about the link quality. Alternatively, a packet-error-rate estimator includes a sequence number only in messages with long preambles. The link quality is calculated from the fraction of received messages.

5. NETWORK PROTOCOLS

To determine whether SP is an effective abstraction to the network layer, we chose three representative protocols from the literature and implemented them in terms of SP: collection routing (MintRoute [53]), data dissemination (Trickle [26]), and data aggregation (Synopsis Diffusion [33]). In this section, we describe their implementation, and defer an evaluation of their performance to Section 6.

5.1 Collection Routing

MintRoute is a collection routing protocol that chooses a parent based on the expected number of transmissions (ETX) to the root of a collection tree [53]. Implementations commonly estimate ETX using additive link qualities, where zero represents a perfect link.[4] A parent's quality is its advertised value, sent periodically via route update beacons, plus the quality of the link between parent and child (lower values are better). MintRoute uses per-hop retransmissions to make additive quality an accurate measure (without retransmissions, ETX would be multiplicative due to the loss potential at each hop as the packet travels across the network).

Our implementation of MintRoute that uses SP sends two kinds of traffic, neither of which is marked urgent: route update beacons, which it broadcasts, and data messages, which it unicasts to the currently selected parent. Our implementation runs on both mica2 and Telos platforms without any code changes. The two types of traffic, route updates and data packets, encounter different handling techniques by the B-MAC and 15.4 link protocols.

MintRoute sends route beacons to the broadcast address. For the mica2, route beacons wake up remote nodes using long preambles. In contrast, our 15.4 implementation sends broadcast messages through a unicast round-robin emulation if no broadcast schedule is established. If the neighbor table has no good potential parents, MintRoute requests that SP `find` new neighbors using the `SPNeighbor` interface. On Telos nodes, the `find` command maps to 15.4's `scan` functionality to repopulate the neighbor table. On the Mica2 platform, SP simply continues using Low Power Listening to receive long messages.

MintRoute's multihop routing "engine" is responsible for data packet transmissions, for both local packets and forwarded packets. MintRoute maintains a send queue to buffer packets waiting to be sent. MintRoute uses the quantity field of its SP message set its queue size. Through message futures, SP bursts pending multihop messages when the destination is available. Data packets are sent with reliability turned on. They are retransmitted by the link and MintRoute is notified if the transmission was successful.

[4]The TinyOS CVS repository has one example implementation: `tos/lib/MintRoute/`.

Link estimation was a major component of early MintRoute implementations. As it was specific to MintRoute, this information could not be easily shared. Integration of the link estimator into MintRoute caused the TinyOS distribution to fork into numerous MintRoute implementations, each with a different integrated link estimator. Over time, these implementations diverged and are now very difficult to maintain [24]. Since SP provides link estimation information through its neighbor table, MintRoute only adds parent quality information (the parent's own ETX) and hop count to the default neighbor table fields. If MintRoute chooses a new parent, it updates outstanding SP messages with the new parent and calls SP's `changed` command to notify SP. If MintRoute does not hear three consecutive route update messages, it evicts that neighbor from the table.

MintRoute handles the `admit` and `evicted` events from SP. When another service tries to admit an entry to the table (such as 15.4 receiving a neighbor's beacon), MintRoute checks if the neighbor is a potential parent by looking at its ETX (received from route beacons). If the estimate is "good", above a threshold or better than existing neighbors, MintRoute allows SP to add the neighbor to the table. If a node is evicted, MintRoute handles the event and checks if SP evicted the current parent. If so, MintRoute chooses a new parent and inspects the neighbor table population. If the neighbor table become sparse, MintRoute calls the `find` command in the `SPNeighbor` interface to repopulate the list of potential parents in the SP neighbor table.

5.2 Dissemination

Trickle is an algorithm for suppression-based data dissemination [26]. Nodes periodically advertise what data they have, unless they have heard other nodes advertise the same data recently. Trickle scales the length of advertisement periods depending on whether a node has heard new data. By default, when a period ends, Trickle makes the next period double the size of the previous period, up to a maximum. When Trickle hears something new, it makes its period very small, so the node that has something new can hear that others need an update. Suppression allows Trickle to scale to dense networks, while scaling advertisement periods enables rapid dissemination with low overhead when all nodes have the current data.

The SP Trickle implementation is extremely simple; running on both mica2 and Telos platforms without any code modification. Trickle sends only broadcast messages, none of which are marked as urgent. Congestion and phase feedback are not considered as Trickle performs its own form of congestion control. However, the basic Trickle algorithm assumes that nodes can atomically and instantly broadcast to all of their neighbors. Delays between message submission and actual transmission can come from multiple sources: SP may delay the transmission of the message to optimize overall node behavior, CSMA-based networks may require backoffs for collision avoidance just like TDMA-based networks may need to wait until a broadcast slot arrives. During the time between message submission and transmission, advertisements may arrive that cause suppression to fail. Our implementation of Trickle uses the `cancel` command of the `SPSend` interface: if there is a broadcast pending when Trickle receives a suppressing message, it cancels the broadcast. SP's unifying interface allows Trickle to operate efficiently without any knowledge of the specific link protocol. Additionally, the link protocol need only provide a best effort broadcast mechanism since epidemic protocols are designed to efficiently handle unreliable broadcast.

Deluge [16] is a bulk data dissemination protocol built on top of Trickle; nodes make requests for data in response to Trickle advertisements of newer data. Due to Trickle's suppression mechanism, the advertisements effectively create clusters where requests elect nodes to transmit large chunks of data.

The SP Deluge implementation makes extensive use of message futures to keep resource usage to a minimum. As each message is transmitted by SP, Deluge pre-fetches the next message from external flash. Through the message futures mechanism, Deluge can realize large and quick data transfers in a power efficient manner while using minimal resources.

Deluge adds broadcast information to its advertisement messages that inform neighbors of transmission periods and recorded in SP's special broadcast neighbor entry. As described in [16], Hui designed Deluge to operate without any neighbor information to eliminate state and complexity. With SP providing a shared neighbor table, Deluge can now take advantage of any available link quality information with minimal added state and complexity. By limiting activity to neighbors with high quality links, contention and packet drops can be reduced. This simple optimization significantly improves propagation rate and energy consumption [17].

5.3 Aggregation

Synopsis Diffusion (SD) is a simple and space efficient approach for estimating whole-network aggregate data values, such as maximum, count, mean, mode, and median [33]. Synopsis diffusion computes estimates with order and duplicate insensitive (ODI) aggregates. This allows nodes to freely exchange aggregate values, safely taking advantage of opportunistic receptions. A user can achieve a desired synopsis accuracy by averaging over a series of independent estimates. SD differs from Trickle by sending estimates of an aggregate value towards a collection point whereas Trickle disseminates updates away from a collection point. The details of how the synopsis calculation at each node is computed is discussed in detail in [33].

SD requires a gradient to the collection point in order to expire old ODI values. SD, by itself, has no mechanism to create a gradient. Our implementation creates its own gradient by a simple hopcount added to all ODI messages. If SD runs in conjunction with another network service that creates a gradient, it uses the shared neighbor table to determine the direction of the collection point. Specifically, when running with MintRoute, SD queries the SP neighbor table and extracts MintRoute's neighbor hopcounts.

Every node periodically broadcasts a packet containing its current aggregate value. Using the SP send interface, it declares that the ODI message does not require reliability nor urgency. When a node hears an ODI value, it only aggregates if the source is further away from the collection point. Since each synopsis is a periodic calculation that is broadcast, the timing of the broadcast is not critical. Urgency is not required so SP and the link work together to batch ODI messages with other broadcast data in order to save power. SD calculates aggregates in a periodic manner local to the node's time, unlike Trickle, thus message timing is not critical for correct operation. Aggregates that are received by a higher level node (closer to the collection point) are collected into a single synopsis; so suppression is inherently built into the aggregation algorithm. Reliability is not necessary since only one node must receive a lower node's synopsis in order to aggregate it. The algorithm accounts for loss by averaging over many synopses received at the collection point.

6. RESULTS

In this section we evaluate the performance and complexity of running network protocols from Section 5 through SP. Our benchmarks validate that our SP design does not sacrifice communication

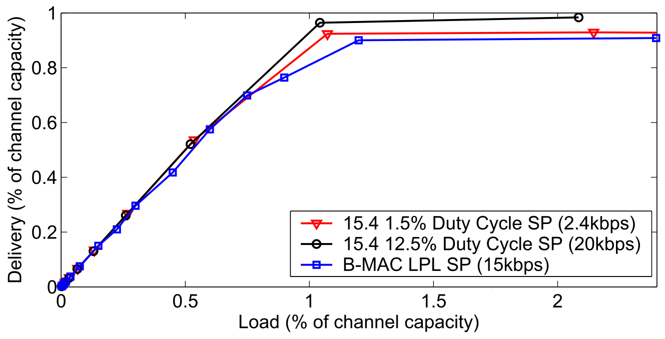


Figure 7: Network protocol offered load versus delivered load of SP running above 15.4 (at 1.5% and 12.5% duty cycles) and B-MAC (using low power listening).

or power efficiency in single and multihop protocols as compared with existing implementations despite presenting network protocols with a unified link abstraction.

For our tests, we used the mica2 B-MAC protocol[5] below our SP implementation as described in Section 4.2. On the Telos (Revision B) platform [36], we added a reduced functionality 802.15.4 protocol that performs neighbor synchronization, beaconing, acknowledgments, and coordination above the default CC2420 networking stack from TinyOS. We implemented the SP interface for the 15.4 stack as described in Section 4.1.

6.1 Single Hop Benchmarks

By using the SP abstraction instead of directly interfacing with the link protocol, network services should not lose performance or power efficiency. If either of these conditions are not met, protocol designers may be motivated circumvent the abstraction to achieve the best performance. To illustrate how SP handles message transmission and reception, we performed bandwidth and load studies on the mica2 and Telos platforms using SP.

We first tested SP's ability to submit pending messages efficiently to the link protocol for transmission. One transmitter was programmed to deliver messages at a given load to a single receiver in a one-hop network. Figure 7 shows the offered load and delivered load running SP on both 15.4 and B-MAC. In both cases, as the offered load increases, SP delivers the load to the receiver. As the load approaches the channel capacity, SP delivers data at almost 90% of the channel capacity across both links. As we vary the 15.4 duty cycle, SP is able to adapt and achieve good results regardless of the underlying link power management strategy.

In order to directly compare the overhead of SP and evaluate the efficacy of our feedback mechanisms, we placed a number of nodes in a circle with a single receiver in the middle. We measured the delivered packet throughput at the receiver and varied the number of nodes in the cell. This test is a direct comparison between SP and the single hop bandwidth test presented in [35], verified by reproducing the data from [35], and is shown in Figure 8.

First, examine the difference in throughput between B-MAC and SP with no power management. As the number of nodes increases, B-MAC's performance decreases much more quickly than SP. We attribute this decrease due to the single-packet interface provided by B-MAC. Since SP is batching messages and then sending them in bulk when the channel becomes available, it can achieve more channel bandwidth than MAC protocols that operate on only a single packet at a time. While optimizations could be implemented

[5]We used `tinyos-1.x/contrib/ucb/tos/CC1000Pulse` as of April 1, 2005 as the source for B-MAC as per [35].

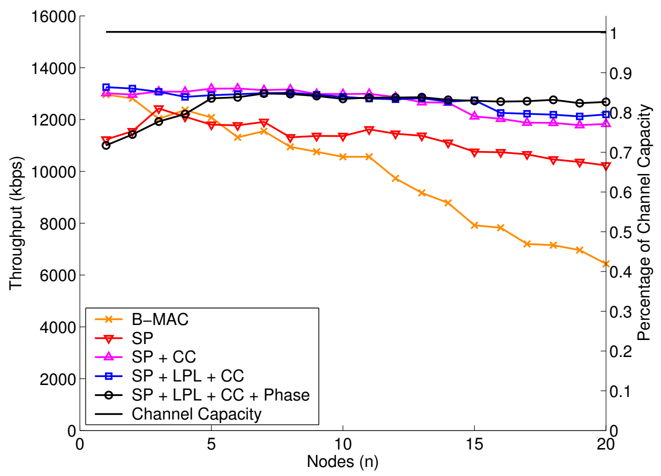


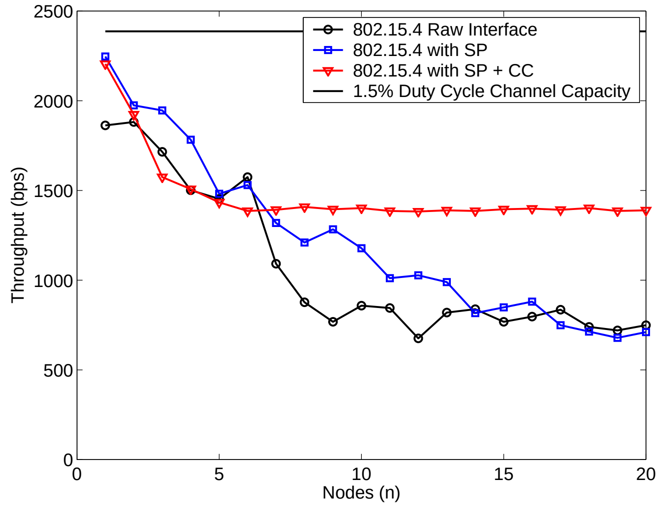
Figure 8: Delivered throughput using SP's control and feedback mechanism on the mica2. Each node transmits as quickly as possible. Congestion control (CC) and phase adjustment are implemented at the network layer using SP

in the monolithic approach, our SP architecture provides a separation of concerns allowing protocol designers to focus their effort on their protocol. This simplifies protocol implementation and does not require entire sets of protocols to be integrated together in order to co-exist within a system. Of more value is the observation that SP did not decrease performance while acting as an intermediary between the network and link protocols.

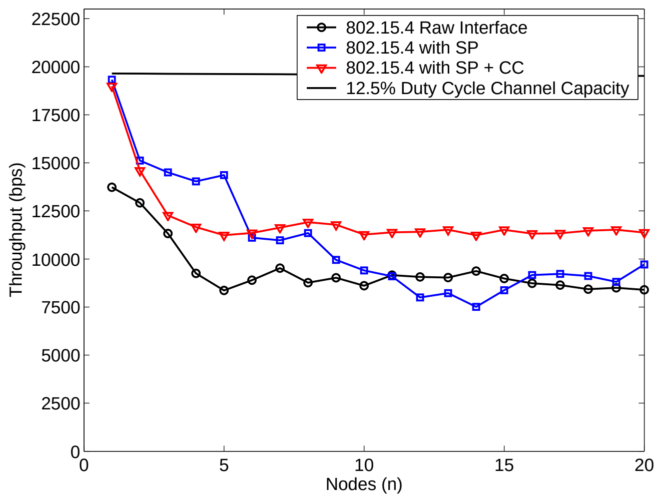
We examined the impact of "piggybacking" on low power operation. SP with B-MAC's LPL sends long preambles when the channel is idle. Other nodes may piggyback their data on the long preamble packet. As the load increases, most traffic is sent using short preambles. This allows SP with low power listening to ramp up to the full bandwidth of the channel, quickly transfer data, and return to sleep. SP with LPL performs identically (within 3% in all cases) to SP without LPL and therefore is omitted from Figure 8.

Finally, we tested the congestion and phase feedback of SP to build a congestion control protocol above SP. In our congestion control (CC) implementation, we used an additive increase, additive decrease (AIAD) scheme due to the low bandwidth of the channel. When the `quantity` field of the message became high or the congestion bit was set, we decreased the message generation rate. Likewise, if the congestion bit is not set and the `quantity` field of our SP message goes to zero, we increase our generation rate. The results show that effective congestion control schemes can be built independent of the underlying link protocol. When phase feedback is provided, we change the phase that the congestion control algorithm submits packets to SP.

We performed the same bandwidth measurements on the 15.4 MAC at duty cycles of 1.5% and 12.5%. Figure 9 shows our results at both duty cycles. With only a single node, SP's bandwidth is very close to the channel limit. The increased delivery of SP is caused by message futures—when the first packet is sent, clear channel estimation is disabled for the remainder of the packets in the message. We observe that as the number of nodes increases, delivered bandwidth drops significantly unlike in the mica2 tests. Because 15.4 radios are faster than microcontrollers, it takes approximately 1.8 ms to load the next packet into the radio's packet buffer plus 450 μs to switch from receive to transmit mode. During this time, other nodes may sense a clear channel and the result is a collision. Our results show that both the non-SP and SP implemen-



(a) IEEE 802.15.4 and SP at 1.5% duty cycle



(b) IEEE 802.15.4 and SP at 12.5% duty cycle

Figure 9: Delivered throughput of a single hop channel running 15.4 at 1.5% and 12.5% duty cycles under congestion. Each node transmits as quickly as possible–more nodes lead to more channel congestion.

Component	**RAM**	**Msgs**	**Flash**
mica2			
TinyOS Engine	34	784	840
TinyOS Neighbors	371	49	1924
TinyOS Total	405	833	2764
SP Engine	50	784	870
SP Neighbors	13	67	1104
SP Total	63	851	1974
Telos			
TinyOS Neighbors	400	44	1884
TinyOS Engine	34	704	848
TinyOS Total	434	748	2732
SP Engine	52	704	874
SP Neighbors	13	64	1244
SP Total	65	768	2118

Table 1: Comparison of the code and memory usage of MintRoute in TinyOS and MintRoute built above the SP abstraction. Engine relays multihop messages from the application to the link protocol. Neighbors performs neighbor management. RAM is the memory usage, Msgs is the amount of RAM used by message buffers, and Flash is the code size in bytes.

tation suffer from this effect. If the underlying MAC made better carrier sense decisions, SP would also benefit. In Figure 9(a), congestion control results in the highest throughput, especially in high contention.

6.2 Multihop Benchmarks

We measured the performance of the network protocols described in Section 5 to determine the overhead imposed by SP. These tests allowed us to verify with real sensor networks that the primitives provided are sufficient for sensor network protocols.

First, we studied the efficacy of our single MintRoute implementation running on both mica2 and Telos platforms. MintRoute using SP is smaller in code size on both platforms shown in Table 1. Since SP separates link estimation and basic neighbor table functions from network protocols, the code for neighbor management is smaller and simpler than its TinyOS counterpart. MintRoute's administration of the neighbor table is the brain behind its operation, and thus where the majority of code is located. The Engine is responsible for sending and forwarding multihop messages. The slightly larger code size is the result of setting fields in `sp_message_t`, not added complexity.

	Min	Median	Average	Max
Duty Cycle	0.031	0.045	0.044	0.047
Delivery	94.1%	96.6%	97.4%	100.0%
Retrans/Pkt	0	0.057	0.059	0.095
Parent Changes	0	1	1.58	5
Parent Evictions	0	0	0	0

Table 2: 15.4 MintRoute statistics on 29 nodes in a 3 hop network over an 8 hour period.

We deployed MintRoute for 8 hours on Telos and mica2 platforms using the same settings as in [35]—a 3 minute application data generation rate and 5 minute route beacon interval. We examined a few metrics to determine if separating concerns led to link-independent optimizations. Table 2 the data collected from running MintRoute above SP on the 15.4 MAC protocol (Telos). MintRoute established reliable delivery and stable routes across a 3 hop network evident by the high success rate. We can conclude that MintRoute effectively managed the neighbor table since there were very few parent changes and evictions. Small parent changes combined with high packet success rate is proof that the link estimation provided relevant values to make good routing decisions independent of the underlying link.

We deployed the same MintRoute implementation above SP on the mica2 platform. Data from the mica2 test led to the same overall results—there were a minimal number of parent changes (at most three) and only one eviction. One additional metric is network power consumption. Since B-MAC can send both long and short packets consuming an order of magnitude different energy, we can evaluate the effectiveness of message futures on system lifetime. Results in [35] show a maximum duty cycle of 2.5% without SP. With SP, our nodes achieved a median duty cycle of 1.1%, maximum of 1.5% for nodes closest to the root, and minimum of 0.5% for leaf nodes. The lower duty cycle is due to message futures and piggybacking. Almost twice as many packets were sent with short

Component	RAM	Msgs	Flash
mica2			
Trickle	7	57	573
Synopsis Diffusion	19	57	*880
telos			
Trickle	10	64	578
Synopsis Diffusion	19	64	642

* An additional 400 bytes of flash are used for 64-bit C libraries

Table 3: Measured code size and memory usage for the Trickle and Synopsis Diffusion SP implementations. RAM is the memory usage, Msgs is the amount of RAM used by message buffers, and Flash is the code size in bytes.

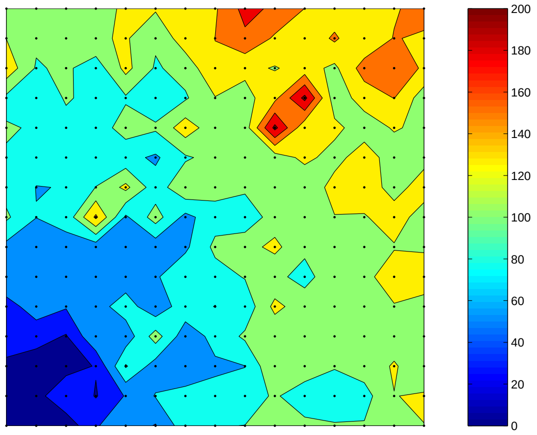


Figure 10: Trickle propagation behavior on 15.4. New data is injected at the lower left corner node with time measured in seconds.

preambles than with long preambles. The large amount of piggybacking saved significant energy and increased the overall network lifetime.

To evaluate Trickle, we ran it on Telos nodes. Trickle successfully disseminated advertisements to all node; however, our deployments only yielded a few hops and were not sufficient for a full analysis. For additional verification, we analyzed Trickle in TOSSIM [23]. We emulated the CC2420 transceiver, including wakeup and transmission times, internal state of the RF IC, and register and memory contents. The RF propogation model is based on empirical data published in [36]. The emulation allowed us to run our Telos SP code and the TinyOS CC2420 link protocol in simulation without any changes.

We simulated Trickle on a grid of 225 nodes to mirror the simulations in [26]. Figure 10 shows the dissemination speed of Trickle when a new advertisement is injected from the bottom left corner of the network. Trickle successfully completes its dissemination despite our 15.4 implementation only providing a round-robin unicast mechanism for messages destined for the broadcast address. Not only does Trickle work well above a foreign link, it also gains the power management capabilities provided by the link. Our Trickle SP implementation is the first time Trickle has been realized and evaluated on a slotted link protocol. Table 3 shows the code size of Trickle, requiring only minimal state and implementation.

To test interoperability between multiple network protocols, we evaluated the power consumption of running MintRoute, Synopsis Diffusion (SD), and MintRoute running with SD. As discussed in Section 5, SD benefits from the gradient created by other protocols. Instead of testing the neighbor sharing, we show that the message pool greatly reduces overall power consumption.

Protocol	Type	Neighbors	Msg Pool
802.11	CSMA/RTS	provide/use	-
SIFT,CSMA/p*	CSMA/CA	-	-
B-MAC	CSMA/LPL	-	very helpful
WiseMAC	np-CSMA	provide/use	helpful
S-MAC,T-MAC	Slotted CSMA	provide/use	helpful
TDMA/802.15.4	TDMA	provide	helpful
TRAMA	TDMA	provide	critical

Table 4: Interaction between SP and link protocols

We tested protocol interoperability on the mica2 over a multihop network with a "density" (or connectivity) of approximately 5 nodes per hop. We ran both protocols separately and then finally within the same system on SP. Our results in Figure 11 show an excerpt of packet receptions during each test. White blocks are MintRoute packets while gray blocks are SD packets. Long packets are represented by a long block while short packets are a short block. The small ticks are the times that the root node sampled the channel for activity.

The results show that each protocol independently must send long packets in order to wake up neighboring nodes. However, when the services are run side-by-side, SP's message pool batches the broadcast messages together to reduce power consumption. Furthermore, neighbor nodes that hear the broadcast piggyback their pending broadcast messages after the current node completes. Analysis of the packet receptions reveals that running the two protocols together above SP result in a 35% power savings over running each of the protocols independently–a 35% power savings can result in a 54% longer node lifetime! The presence of the message pool allowed batching of common messages leading to significant power savings; we made no changes to either network protocol in order to run this test.

7. RELATED PROTOCOLS

In previous sections, we illustrate our SP design through implementation and performance benchmarks on the TinyOS platform. Three different network protocols showed the benefits of pushing the unifying abstraction closer to the data link layer. In this section, we review related protocols above and below SP, and discuss how these protocols can use SP.

7.1 Link Layer Protocols

Researchers have proposed may link protocols for wireless sensor networks. To show the generality of our SP abstraction, we investigated recent publications that are representative of current protocols. Table 4 summarizes proposed link protocols and how they can potentially interact with SP. For example, classic CSMA protocols, such as SIFT or CSMA/p* [20, 45], use different carrier sense and backoff techniques to resolve multiple access on the channel. They do not require neighbor information in general, nor do they optimize transmission according to information in message pool. When the link does not maintain neighbors, SP relies on network protocols to populate the table. For example, MintRoute uses information from route beacons to add entries to the neighbor table if they do not exist. Other CSMA protocols, like WiseMAC [7], maintain schedules to reduce idle listening. WiseMAC uses a similar optimization to that presented in Subsection 4.2 where nodes learn their neighbor's low power listening schedule. Although many CSMA protocols do not maintain neighbor information, they benefit from the message pool and message futures that enable higher throughput by aggregating transmissions.

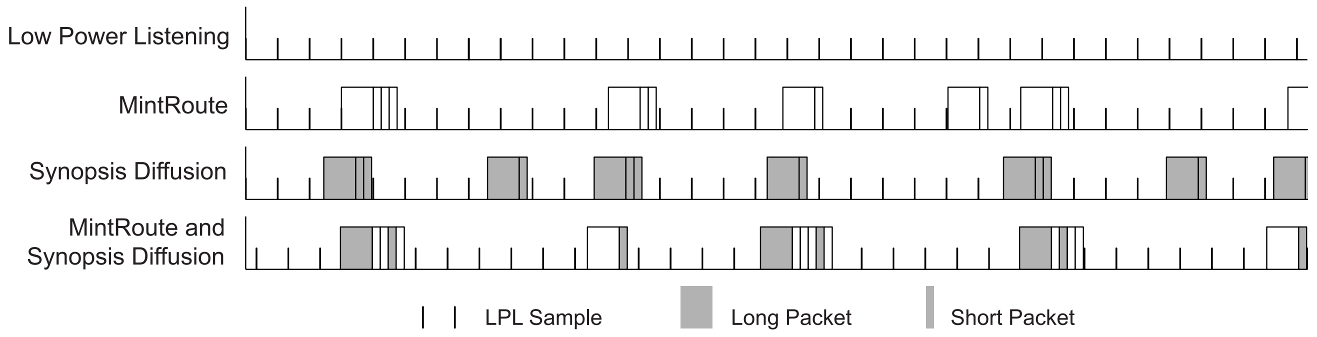


Figure 11: Multiple network protocols running above SP can cause the overall system to save power compared to running the protocols independently. This 2 minute excerpt of each protocol running on the mica2. The top graph shows an idle node while the bottom graph shows the protocols piggybacking on each other even though they are oblivious to the other's presence. MintRoute packets are shown in white, while synopsis diffusion packets are gray.

On the other end of spectrum, it is mandatory for many TDMA-type schemes, such as S-MAC [56], T-MAC [50] and TRAMA [37], to maintain a list of neighbors' schedules. Some of them can optionally optimize their schedules through the use of message futures. Our proposed design of SP can serve as a good link layer abstraction to accommodate these link protocols without losing their primary features and benefits.

7.2 Network Layer Protocols

Three different network protocols—collective routing, data dissemination and aggregation—perfomed well using our SP abstraction. SP is applicable to other network protocols. We tabulated the properties of several proposed network protocols in Table 5. Despite the large range of functionality, most benefit from shared neighbor state. A unified link abstraction not only provides a link-independent interface but also reduces state and computation overhead signficantly for maintaining system-wide neighborhood information. Even if neighbor information is provided by the link layer, other network protocols, such as topology control protocols [1, 14, 55], can enforce domain-specific policies on the neighbor population to achieve additional power savings or throughput. Many network protocols generate periodic traffic; the SP message pool structure allows the link protocol to optimize accordingly. Network protocols may inspect the pool and adjust their behavior. For example, AODV [34] can take advantage of message futures to delay expiration of a routing state if the same route will be used again in the near future. Extreme cases are PEDAMACS [10], FPS [15], and AppSleep [38] which require explicit information of the upper layer's bandwidth and traffic patterns to schedule transmissions. These protocols may maintain their slots through the neighbor table wakeup schedules, and even overwrite MAC layer schedules, to dictate when SP can safely send messages. For all of these protocols, SP provides a rich, yet concise, set of primitives for network services to achieve their expected functionality. In many cases, SP can reduce implementation overhead by simplifying access to commonly used structures such as the neighbor table.

8. CONCLUDING REMARKS

Culler et. al. claim that "the primary factor currently limiting progress in sensornets is not a specific technical challenge but instead is the lack of an overall sensor network architecture." [3] The unified link level abstraction embodied in SP can advance the research in sensor networks and provides a first step to building a larger architecture. By building upon SP's abstraction, rather than programming directly to the specific link, protocols can outlast technology generations. Already successors to 802.15.4 are in progress and many innovations in low power radio designs are emerging. These may provide more efficient forms of sampling or slotting. The evidence provided here of power-aware network protocols expressed in terms of SP being mapped efficiently to very different link-level power management mechanisms suggests that these same protocols are likely to map to future link technologies. Thus, good protocols can be long lasting and can be improved with time and experience. Second, such an abstraction encourages innovation in network protocol design because the designer can realize the protocol at a fairly high level, without concentrating on link specifics. By exposing sets of packets, exerting simple reliability and urgency controls, adapting to congestion and loss after concerted effort, and by cooperating in neighbor management and schedule formation, protocol optimizations can be realized on a variety of specific link layers. Although in a more monolithic design, a network protocol may squeeze every last ounce out of the link through an integrated approach, we find that in practice a lean layer that provides a separation of concerns may allow greater overall optimization as the developers can more easily focus on key aspects of the protocol design, rather than designing the entire system. We presented several cases where SP achieves better performance through the abstraction than the extisting monolithic implementations. Those existing protocols could be improved using the concepts presented by SP; however, we feel the optimizations are easier to perform with the abstraction in place. Finally, it becomes natural to think about optimizations arising from multiple coexisting protocols cooperating in how they use resources and share information.

The unified abstraction presented here is only a step towards an "overall sensor network architecture." There are significant issues that are within the scope of a link layer abstraction like SP that we have not addressed and there are important architectural issues that are beyond the scope of such an abstraction.

Many sensor network applications utilize time correlated time samples and thus require time synchronization. Example applications include structural analysis [54], shooter localization [40], and several middleware services like localization [30]. Time synchronization services [9, 31] are usually situated above the link layer; however, MAC-layer timestamping greatly improves precision [29] by providing microsecond resolution of the start-of-frame delimiter. Since timestamps must cross layers, a unifying link abstraction should provide a mechanism for conveying timestamps. There are a

Protocol	Function	Neighbor List	Message Pool
AODV, DSDV	routing	use	helpful
MintRoute	1-sink routing	provide/use/trim	queueing
PSFQ, RMST	reliable transport	use	helpful
CODA, Fusion	congestion control	maybe use	critical
TAG	query/aggregation	use	helpful
Synopsis Diffusion	aggregation	use for gradient	helpful
Trickle, Deluge	dissemination	helpful	helpful
AppSleep, FPS	scheduling	use extensively	critical
LEACH, GAF, SPAN	topology management	use/trim	-

Table 5: Network Protocols above SP and their use of SP's neighbor table and message pool. Features marked as helpful indicate where code complexity is reduced if used.

number of ways one could implement timestamping in SP—packet meta-data and event callbacks are two possible methods. The preferred method of exposing timestamps in the link abstraction remains an open question.

Arguments can be made for additional features in SP. For example, SP provides single-hop communication but does not explicitly address multicast. One can imagine that providing such a capability between the link and network layers might simplify both, much as cooperative neighbor management and scheduling in SP has done.

Although SP stands for "Sensor Protocol", it does not embody all of the traits commonly attributed to a conventional protocol. SP defines communication between two entities—in our case, SP facilitates communication between protocols within a system. In this way, SP provides a service together with a set of unifying interfaces. On the other hand, one aspect notably absent from SP is a wire format. As both SP and the proposed sensornet architecture mature, SP services and interfaces may promote a full protocol definition, similar to IP. However, SP currently operates as an architecture for composing protocols, a communications service, and a unified abstraction for building efficient sensor network systems.

Numerous network level architectural issues are beyond the scope of SP, although we hope the presence of such an abstraction will enable their resolution. For example, network level naming remains an open question. Several studies have suggested attribute-based naming, rather than node addresses, is particularly important in sensor networks [19, 28]. Others have observed that address-free protocols are important [3, 8] and the use of predicates for identifying participants in network-level communication. SP takes no position on these issues; it simply conveys opaque link level unique identifiers and a broadcast address between the link and network layers. In addition, SP does not address how sensor network patches and the services within them present themselves to the Internet. It simply provides a way to connect to gateway nodes, regardless of the specific link technology. SP does not dictate what network protocols ought to be present, how they are factored, or what they should do. Its design does pay special attention to allowing multiple network protocols to coexist and cooperate and anticipates that applications would specialize what protocols are present, but the abstraction would be valuable if a single widely used network layer emerged. Our SP design strictly provides mechanisms for network protocols to be built. We have strived to separate mechanism from policy by facilitating communication between link and network protocols.

SP does not address how security is integrated into the sensor network architecture. Since SP acts as a communication mechanism between link and network protocols, we envision that link security may be implemented orthogonally from network security with SP sandwiched in the middle. The packet's data payload is opaque to SP, and thus the contents are not important for SP's operation. The Zigbee security model follows this policy—it supports independent security at link, network, and application layers.

One might ask what are the inherent limits of our approach. Will the urgency and reliability hints be sufficient for "effector networks" to provide distributed control of various actuators? Will it extend to other wireless links, such as those that use frequency hopping and other forms of diversity? Ultimately, time and continued innovation prove out an architecture. As a step toward those larger determinations, we have shown a novel kind of translucent layer that allows power-aware network protocols to operate very effectively on widely varying link-level power management schemes. We have shown multiple network protocols gaining benefit from their coexistence over a unified link level abstraction. We have seen that such a "narrow waist" can be formed without sacrificing performance and even that the separation of concerns can lend itself to optimizations that, while theoretically possible, had not appeared in monolithic approaches. We may conclude that in-network processing, rather than end-to-end communication is not inconsistent with the presence of unifying abstraction. Indeed, such a lower layer waist can be made concrete and demonstrated to be effective.

Acknowledgments

This work was supported by the National Science Foundation under grant # 0435454 ("NeTS-NOSS"), DARPA grant F33615-01-C1895 ("NEST"), California MICRO, and other NSF support.

9. REFERENCES

[1] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Proceedings of the Sixth ACM Conference on Mobile Communications and Networking*, July 2001.

[2] Chipcon Corporation. CC1000 low power FSK transciever. `http://www.chipcon.com/`, Apr. 2002.

[3] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: Lowering the waistline. In *Proceedings of the International Workshop on Hot Topics in Operating Systems (HotOS)*, 2005.

[4] J. Ding, K. Sivalingam, R. Kashyapa, and L. J. Chuan. A multi-layered architecture and protocols for large-scale wireless sensor networks. In *Proceedings of the IEEE Vehicular Technology Conference*, Oct. 2003.

[5] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *Proceedings of The Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, Apr. 2005.

[6] A. El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *Proceedings of IEEE International Conference on Communications*, Apr. 2002.

[7] A. El-Hoiyi, J.-D. Decotignie, and J. Hernandez. Low power MAC protocols for infrastructure wireless sensor networks. In *Proceedings of the Fifth European Wireless Conference*, Feb. 2004.

[8] J. Elson and D. Estrin. An address-free architecture for dynamic sensor networks. Technical Report 00-724, University of Southern California, Computer Science Department, Jan. 2000.

[9] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, 2002.

[10] S. C. Ergen. PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. Master's thesis, University of California at Berkeley, Dec. 2002.

[11] D. Gay. nesC: A programming language for deeply networked systems. `http://nescc.sf.net`, Mar. 2005.

[12] D. J. Goodman, R. A. Vrdenzuela, K. T. Gayliard, and B. Ramarmrrthi. Packet reservation multiple access for local wireless communications. *IEEE Transactions on Communications*, 37, Aug. 1989.

[13] M. Hamilton, M. Allen, D. Estrin, J. Rottenberry, P. Rundel, M. Srivastava, and S. Soatto. Extensible sensing system: An advanced network design for microclimate sensing. `http://www.cens.ucla.edu`, June 2003.

[14] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of the Hawaiian International Conference on Systems Science*, Jan. 2000.

[15] B. Hohlt, L. Doherty, and E. Brewer. Flexible power scheduling for sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, Apr. 2004.

[16] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[17] J. W. Hui, A. Newberger, and G. Tolle. Data dissemination with geometric structure. `http://www.cs.berkeley.edu/~jwhui/research/deluge/cs262/cs262b-report.pdf`, May 2004.

[18] Infineon Technologies AG. TDA525x Series ASK/FSK Transceiver Family. `http://www.infineon.com/wireless`, July 2002.

[19] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks*, Aug. 2000.

[20] K. Jamieson, Y. Tay, and H. Balakrishnan. Sift: A MAC protocol for event-driven wireless sensor networks. Technical Report MIT-LCS-TR-894, MIT, May 2003.

[21] M. J. Karol, Z. Liu, and K. Y. Eng. An efficient demand-assignment multiple access protocol for wireless packet (atm) networks. *ACM/Baltzer Wireless Nefworks*, 1(3):267–279, 1995.

[22] J. Kulik, W. Rabiner, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th ACM/IEEE Mobicom Conference*, Aug. 1999.

[23] P. Levis, N. Lee, M. Welsh, , and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2003.

[24] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The emergence of networking abstractions and techniques in TinyOS. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[25] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An operating system for wireless sensor networks. In *Ambient Intelligence*. Springer-Verlag, 2005.

[26] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, Mar. 2004.

[27] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[28] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of SIGMOD*, June 2003.

[29] M. Maróti, B. Kusý, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[30] W. Merrill, L. Girod, J. Elson, K. Sohrabi, F. Newberg, and W. Kaiser. Autonomous position location in distributed, embedded, wireless systems. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, 2002.

[31] D. L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[32] E. Mohr, D. Kranz, and R. Halstead. Lazy task creation: a technique for increasing the granularity of parallel programs. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, 1990.

[33] S. Nath, P. B. Gibbons, Z. Anderson, and S. Seshan. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[34] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[35] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[36] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of The Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, Apr. 2005.

[37] V. Rajendran, K. Obraczka, and J. Gracia-Luna-Aceves. Energy efficient, collision-free medium access control for wireless sensor networks. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, Nov. 2003.

[38] N. Ramanathan, M. Yarvis, J. Chhabra, N. Kushalnagar, L. Krishnamurthy, and D. Estrin. A stream-oriented power management protocol for low duty cycle sensor network applications. In *Proceedings of the Second IEEE Workshop on Embedded Networked Sensors*, May 2005.

[39] D. Raychaudhuri and N. D. Witson. Atm-based transport architecture for multi-services wireless personal communication networks. *IEEE Journal on Selected Areas in Communications*, 12:1401–1414, Oct. 1994.

[40] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusý, and A. Nádas. Sensor network-based countersniper system. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[41] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Proceedings of the ACM/IEEE Conference on Mobile Computing and Networking*, Oct. 1998.

[42] K. M. Sivathrgarn, M. B. Srivastav, P. Agmwrd, and J. C. Chen. Low-power access protocols based on schedrding for wireless and mobile atm networks. In *Proceedings of the IEEE International Conference on Universal Personal Communications*, Oct. 1997.

[43] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[44] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, 2004.

[45] Y. Tay, K. Jamieson, and H. Balakrishnan. Collision-minimizing CSMA and its applications to wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, Aug. 2004.

[46] The Institute of Electrical and Electronics Engineers, Inc. Part 2: Logical Link Control, May 1998.

[47] The Institute of Electrical and Electronics Engineers, Inc. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.

[48] The Institute of Electrical and Electronics Engineers, Inc. Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs), June 2002.

[49] The Institute of Electrical and Electronics Engineers, Inc. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), Oct. 2003.

[50] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, Nov. 2003.

[51] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, Sept. 2002.

[52] A. Woo and D. E. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the seventh annual international conference on mobile computing and networking*, July 2001.

[53] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of multihop routing in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, Nov. 2003.

[54] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[55] Y. Xu, J. S. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Mobile Computing and Networking*, pages 70–84, 2001.

[56] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *In Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, June 2002.

[57] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, Nov. 2003.