Rethinking Link-Level Abstractions For Sensor Networks

Zhitao He, Adam Dunkels, Thiemo Voigt, Nicolas Tsiftes
{zhitao,adam,thiemo,nvt}@sics.se
Swedish Institute of Computer Science

***Abstract*—For designers of the communication stack of sensor nodes there is a constant tension between performance and modularity. To alleviate this tension, researchers have come up with a number of modular architectures. In this work we take a refreshed view of the design of an abstract link level service, an important component in the communication stack. We start with a critical review of one such service, the Sensornet Protocol (SP), and then we implement an SP-flavored link level service featuring a novel combination of ARQ and MAC. Experimental results of transmission delay and energy efficiency highlight a few subtle architectural design trade-offs we have encountered, namely semantics binding, implicit information sharing, and time scope initialization. These aspects have significant impact on software modularity in tiny sensor nodes.**

I. Introduction

The overall performance of a sensor network largely hinges on the efficient operation of the communication stack on each sensor node. Still an ample field of innovations itself, lower-level protocols such as MAC protocols are increasingly used as a standard system service. Nevertheless, the tension between critical performance metrics such as power efficiency and the need for software modularity have limited code reuse to a small proportion of the whole stack, and often requires further fitting and tweaking of the reused code. Therefore, researchers have come up with a number of modular architectures to address the need for an efficient abstraction of the lower layers.

One such software architecture explicitly targeting component reuse is the Sensornet Protocol (SP) proposed by Culler et al. [1] and realized by Polastre et al. [2]. The main abstraction of SP is a single-hop message sending and receiving service that encapsulates the actual datalink protocol and radio driver with a standard API. Essentially, SP divides the protocol stack into an upper part and a lower part by relaying messages across its upper and lower interfaces. This enables a standard way of designing new protocols or stacking existing protocols in a structured manner. Later developments that followed the same line of thought - architecture-based component reuse, include the modular network layer architecture (NLA) by Cheng et al. [3], the Chameleon architecture by Dunkels et al. [4], the MAC layer architecture (MLA) by Klues et al. [5], although with different focuses.

The purpose of our work is to take a refreshed look at the problem of defining a unifying link-level service, first provided by SP as a vision and with a showcase implementation, and later refined by NLA, Chameleon, MLA, etc. We first provide a critical review of the original SP abstraction to identify several key trade-offs, and complete the study with an implementation of a link service featuring an ARQ link retransmission protocol and a low power listening MAC protocol X-MAC [6]. Our analytical study and experimental results combine together to show the importance of a number of underestimated aspects that constantly undermine the elegance of components and layers, namely semantics binding, implicit information sharing, and time scope initialization. In additional to providing a solution, we also give a few afterthoughts on modular design.

The rest of this paper is structured as follows. In Section II we present a critical review of SP, in the light of comparative studies of a few other related services. In Section III we describe our implementation of a generic ARQ link service, to be followed in Section IV by a detailed elaboration of a few modularity-performance trade-offs we make to integrate X-MAC to the link service. Section V presents experimental evaluation of two different implementations in terms of packet delay and energy consumption. Section VI discusses the current trend of modular design and clarify the distinction between module and component; We conclude in Section VII by summarizing our findings.

II. Review of a unifying link abstraction layer

The SP layer essentially provides a single-hop message unicast and broadcast service with optional QoS control and feedback. This abstraction brings two main benefits: multiplexing of higher-layer services; decoupling of message submission and message transmission. These enable the next higher layer to see a continuous connection with any neighbor. Message sending involves two separate steps: a submission request and later a completion indication. Reception is triggered by an asynchronous event.

A. Expressivity

The expressivity of the SP API is similar to that of another logical link control protocol, the *acknowledged connectionless-mode* of IEEE 802.2 [7]. Table I lists the primitives of the network layer/SP interface, in parallel to IEEE 802.2's network layer/LLC layer interface. The similarity between the two is highlighted by the almost identical packet parameters used by their send primitives, shown in Table II. These parameters are passed out-of-band together with the data unit involved across the layer interface. The Rime stack in the Chameleon architecture [4] has a set of single-hop services using similar parameters.

TABLE I
NETWORK LAYER/DATALINK LAYER DATA TRANSFER INTERFACES OF SP AND IEEE 802.2 TYPE 3 OPERATION

Primitive	SP	802.2
Send packet	x	x
Receive packet	x	x
Modify submitted packet	x	
Cancel submitted packet	x	
Fetch next packet to send	x	
Retrieve packet from neighbor		x
Prepare packet to be retrieved		x

TABLE II
PACKET PARAMETERS PASSED BY THE SENDING PRIMITIVES OF SP AND IEEE 802.2 TYPE 3 OPERATION

Parameter	SP	802.2
Urgency (802.2 Priority)	x	x
Reliability (802.2 Service Class)	x	x
Maximum number of retransmissions	x	*constant*
Transmission status (success/failure, congestion, etc)	x	x

B. Complexity

Hidden behind these similar interfaces are however very different implementation complexity, based on different assumptions about traffic patterns and link mechanisms.

SP's implicit assumption about its lower layer is based on the existing types of sensor networks MAC protocols, which are characteristic of low duty cycles and interdependence between neighbors' wake/sleep schemes. On the other hand, average traffic load is supposed to be low compared with channel bandwidth. Therefore, a centralized message pool is used to provide sufficient buffering of unsent packets and necessary synchronization with neighbors. Message transmission is done after an online search among pooled messages, that is based on constraints of neighbor states and message priorities. Figure 1 illustrates this search policy of SP. There exists a potential performance problem with this, as the searching delay is proportional to the product of the sizes of the message pool and the neighbor table.

In IEEE 802.2, packet buffering is distributed among the sending components, essentially coexistent state machines, of individual logical links. This design avoids the latency of an online search at the expense of dedicated resource allocation and complex state maintenance, and requires relatively high quality links to operate efficiently.

Chameleon and MLA push the complexity of message buffering and scheduling down to a multi-message MAC layer, that has full control over queue management and state maintenance specific to the MAC protocol in use.

C. Message Encoding and Decoding

SP and Chameleon are examples of the two opposite ends of message encoding and decoding policy. SP assumes that a predefined message format, such as the default TinyOS message format, is global knowledge to all layers. Therefore SP simply acts as an internal router storing and forwarding messages across its upper and lower interfaces, relying on its

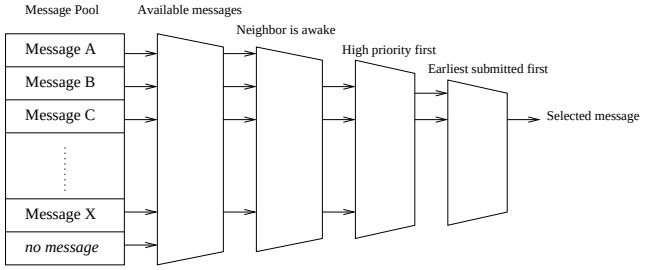

Fig. 1. The SP message pool's online search policy

next higher layer to perform encoding and decoding. TinyOS 2 [8] follows this cross-layer message format assumption. Chameleon, in contrast, assumes that higher layer protocols may benefit from a more abstract semantics, represented by individual packet attributes. To achieve this, packet encoding and decoding are performed by a dedicated attribute-header field transformation module below Chameleon's single-hop broadcast layer.

The dichotomy of encoding/decoding mechanisms between SP and Chameleon represents a subtle trade-off between flexibility and modularity. By drilling a hole across layers to leak packet header information, SP allows maximum flexibility but falls short of presenting a complete abstraction. It also has to duplicate some control information of the message header. Chameleon, on the other hand, seals off lower layer details by means of strict layering and late semantics binding. An additional benefit is the possibility of aggressive header compression. But strict layering increases the risk that an emergent low level feature will violate the predefined set of high level semantics.

III. IMPLEMENTATION OF A LINK SERVICE

We implement a single-hop message service in the Contiki operating system [9]. The message service consists of an abstraction layer above a stack of reliability and MAC mechanisms. The abstraction layer provides a common user API that is similar to the TinyOS 1.x-based SP [2] [10]. This service enable us to make independent evaluations unaffected by TinyOS artifacts.

A. Message API

We show our message API in Figure 2, which consists of a initialization function, a send function and a receive callback. The main differences between our API and its TinyOS 1.x counterpart are: instead of obliging the user to bind the message payload and control parameters explicitly into an *SP message* before submission, we treat them as function parameters and bind them internally later, thus relieving the user from the use of error-prone opaque compound data structures; we bind the split-phase send request and the send completion indication into a concise function call by leveraging the convenience of callback; we do not implement any message modification or cancellation primitive.

```
sp_init(datalink_protocol);

sp_send(data, length, destination, urgent, reliable,
        send_completion_callback(data, length, destination,
                                 acknowledged, congested, sending_delay));

sp_set_receiver_callback(receive_callback(data, len, source, destination
                                          signal_strength));
```

Fig. 2. Message API. The asynchronous interface represents a message as a payload associated with a set of parameters. Asynchronous events including the send completion confirmation and the receive indication are handled by respective user callbacks. This C function-based interface enforces much stronger type checking than generic event handlers that process pointers to compound data structures, an error-prone practice.

B. Message Sending and Receiving

The *sp_send* primitive does three things: 1. It encodes the actual packet by calling the link protocol's *new_packet* function; 2. It reserves an entry in a central message pool to store the packet, its associated parameters, and a time stamp; 3. It performs the search algorithm that actually looks up the message pool to decide whether any message is ready to be transmitted by the link protocol. In case of multiple valid candidates, it selects the best one, i.e. the earliest submitted, high priority message. Because we desire to have stable links that are agnostic to bursty submissions and radio duty-cycling, the search algorithm needs to be called on two other occasions: when a message transmission is done; or when the radio wakes up from sleep mode.

Message reception is handled by a user callback invoked by the link service. An incoming packet is presented to the next higher layer as a payload with a number of parameters decoded by the underlying link protocol.

The message sending and receiving mechanisms of our link service embodies a late binding/early unbinding principle of protocol semantics. The goal is to fully leverage the abstracting power of layering by pushing syntactic details to lower layers.

C. Neighbor Management Service

The neighbor table of SP is accessible to other layers via a common interface, to enable cross-layer collaboration such as link estimation and routing. This falls into the shared database category of typical cross-layer implementations classified by Srivastava and Motani [11].

We implement a similar mechanism in our link service, with the following enhancements. Firstly, we keep track of the time a neighbor entry was last updated, as a hint for neighbor freshness that can be used by various protocols such as routing or topology control. Secondly, we reserve three generic link quality metrics to be supplied by the PHY layer, the link layer, and the routing layer respectively. This is based on similar observations about multi-layer link estimation made by Fonseca et al. [12]. But instead of encoding the multiple values of link quality into a bit format, we leave the definition of these metrics to the respective protocol entities, again favoring the late-binding principle. Thirdly, we allow the user to extend protocol logic by using callbacks for customized operations on any specific neighbor. For example, the user may adjust a neighbor's link quality upon packet reception, by hooking the adjustment routine to the *neighbor update* primitive.

D. A Link-Level ARQ Protocol

We choose an acknowledgment-based automatic retransmission protocol, a commonly used reliability mechanism for wireless networks, to be the foundation of our link-level service. The module also defines a 6-byte packet header consisting of source and destination addresses, a packet type (either 'data' or 'acknowledgment'), and a sequence number. Encoding of outgoing messages and decoding of incoming packets are performed by the module as well. Retransmission is enabled by the reliability flag associated with the message being transmitted, and the retransmission counter is fed back as a congestion indicator upon sending completion. The implementation is intended to be general enough to run on different MAC protocols and radio platforms.

IV. Integrating X-MAC To the Link Service

The message pool, the neighbor table, the ARQ protocol, together with a radio driver, constitute a complete link-level service. Our interest, however, lies in the adaptability of this architecture. More specifically, we want to evaluate the expressivity of the user API, the performance of the buffer management and transmission scheduling mechanisms within the typical bandwidth and energy constraints of sensor networks. Therefore we add one of the state-of-the-art MAC protocols, X-MAC [6], to the stack.

X-MAC can be viewed as an enhanced version of the low power listening B-MAC [13] with packetized preambles, or a hand-shaking channel reservation protocol using multiple RTS packets to trade for reduced idle-listening time. Apart for its well known power-saving properties as a MAC protocol, we choose X-MAC for its lack of explicit acknowledgments, which makes a link-level ARQ protocol a perfect complementary mechanism to it.

The obvious way to achieve maximum modularity is to stack the ARQ layer above an independent MAC layer. The latter duty-cycles the radio and exchanges strobe packets before delivering an encoded packet. This apparently clean slate implementation, nevertheless, must rely on a minimum set of shared knowledge: source and destination's addresses in link-level formats are needed to encode and decode RTS and CTS packets, a broadcast flag or a predefined broadcast address disables hand-shaking and enables persistent strobing.

Experimental evaluations reveal further subtle layer interdependencies, whose details will be provided in Section V, that undermine the performance of this baseline implementation, and prompt us to explore improvements.

One dependency lies in layer-specific timing configurations, where a mismatch between the ARQ retransmission timeout and the X-MAC cycle period lead to either unnecessary early retransmissions that waste bandwidth and energy, or necessary but belated retransmissions that increase latency. We can see

that the problem of time scope matching between adjacent layers exists in higher layers too, e.g., end-to-end retransmissions must depend on estimation of single-hop latency. We propose a bottom-up initialization procedure that allows a lower layer protocol to pass time configuration to the next higher layer to successively establish a consistent and layer-optimal hierarchy of time scopes. This also means that existing component-based architectures such as MLA that uses a top-down configuration approach are inadequate. Another type of information that needs to be passed bottom-up through the stack to ensure consistency is the maximum transmission unit (MTU) of the radio driver that is normally constrained by hardware or software buffer sizes.

ARQ acknowledgments, if sent over X-MAC as normal packets, not only roughly double the bandwidth usage as they require the same strobing procedure, but also cause the packet delivery latency to be dependent on the phase difference between specific sender-receiver pairs, as shown in Figure 3. These effects can be remedied by modifications to timing. We may reset a sender's X-MAC schedule whenever a transmission is completed, so that an immediate subsequent acknowledgment from the receiver will be processed in a timely manner. This optimization doesn't require any changes to the ARQ layer, but only takes advantage of the knowledge about its existence. Such an assumption about a next higher-layer mechanism undermines the MAC layer's neutrality. Another more aggressive optimization is to replace the ARQ acknowledgment with MAC acknowledgment that is sent immediately after packet reception, thus eliminating the latency and energy overhead of extra strobing all together. This optimization, however, requires a major code revision as generation and detection of acknowledgment packets are now moved from the ARQ layer to the MAC layer. Figure 3 illustrates the three acknowledgment schemes we have discussed above: normal link acknowledgment, quick link acknowledgment, and MAC acknowledgment.

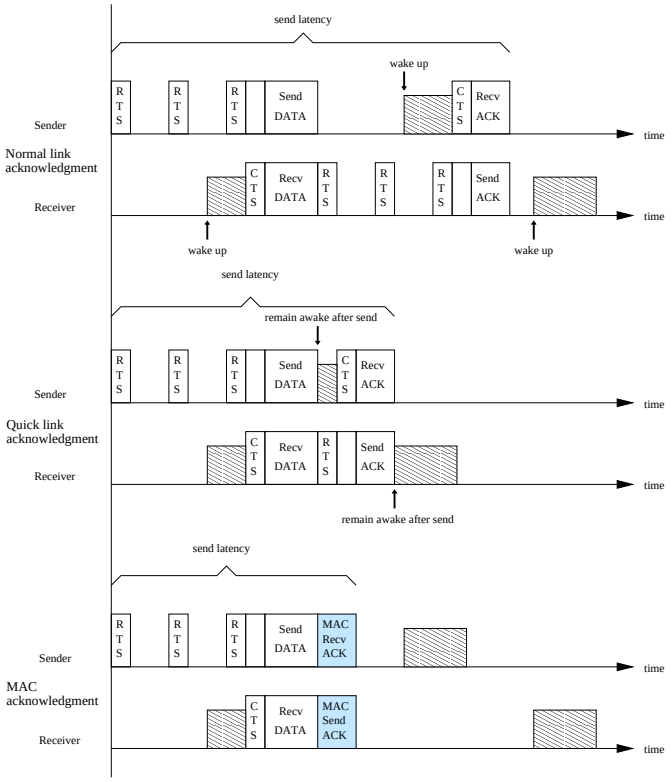

Fig. 3. Send latency can be reduced by optimizing the acknowledgment scheme. In quick link acknowledgment, X-MAC resets its power cycle after a transmission, so that the sender remains awake for any immediate feedback from the receiver. MAC acknowledgment lets X-MAC generate and detect acknowledgment packets immediately after a packet delivery.

V. Evaluation

We evaluate several key aspects of our link service: packet delivery latency, bandwidth efficiency, and code and data memory usage. The experiments are conducted on the TelosB platform [14], with various combinations of acknowledgment schemes and X-MAC duty cycles.

A. One-to-One Latency and Bandwidth Efficiency

First we evaluate the performance of the link service without congestion. The experimental set-up consists of a sender and a receiver running a unicast traffic generation program using the same link service configurations. A test run is started by the receiver broadcasting its address to the sender, who stores the receiver address in its neighbor table. Then the sender starts to generate 100 unicast messages, each 50 bytes in size, to the receiver at random time, so that any phase-coupling between submission and transmission is eliminated.

Packet delivery latency consists of three components: submission, transmission, and acknowledgment. The parameterized central message pool enables us to conveniently perform timestamping of the submission and acknowledgment events associated with each message, whereby the message latency is the derived from the difference between the two timestamps. The send completion callback sums up the message latencies and performs averaging at the end of the run. Figure 4 shows the average raw bit rates with the three different acknowledgment schemes and X-MAC duty cycles, from 2% to 25%. The normal link acknowledgment scheme incurs the largest latency due to excessive X-MAC strobing for sending the acknowledgment, whereas the quick link acknowledgment and MAC acknowledgment nearly double the performance, benefiting from reduced strobing.

Because radio communication usually dominates a node's power consumption profile, bandwidth efficiency is a good indicator of energy efficiency. We use the average radio duty-cycle, i.e. the amount of time when the radio is listening or transmitting divided by the total run time, as our bandwidth efficiency indicator. We conduct the time measurements by enabling the software-based energy estimation utility of Contiki [15]. Figure 5 shows the radio duty-cycles of the different acknowledgment schemes under various X-MAC duty-cycles. Note that unlike in the latency measurements where the average message latency is independent of the traffic load as long as no congestion occurs, the bandwidth

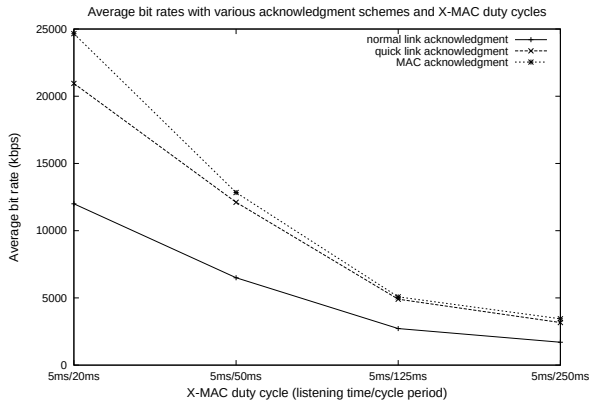

Fig. 4. Average raw bit rates with different acknowledgment schemes under various X-MAC duty cycles. The quick link acknowledgment scheme and the MAC acknowledgment scheme both reduce latency by almost half.

efficiency is an artifact of the relative traffic load over available bandwidth. The measurements show that both the improved acknowledgment schemes offer significant energy savings over the normal acknowledgment scheme.

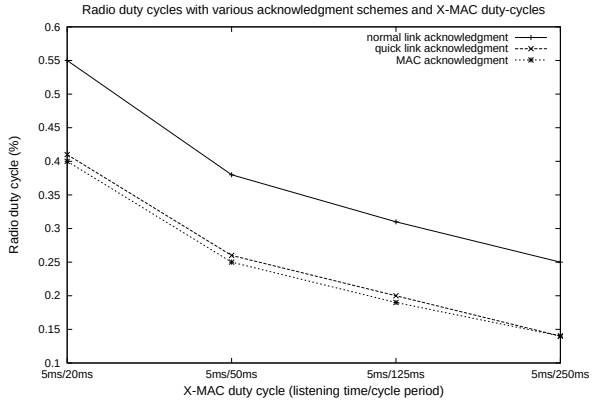

Fig. 5. Radio duty cycles with different acknowledgment schemes and various X-MAC duty cycles. The reduction in radio duty cycle when using either of the improved acknowledgment schemes can contribute to significant energy savings, regardless of X-MAC duty cycles.

B. Many-to-One Latency and Bandwidth Efficiency

We repeat the latency and bandwidth efficiency tests using multiple senders, which introduces packet collisions that are remedied by the ARQ mechanism. We use a 10% X-MAC duty-cycle and increase the number of senders from 1 node to 5 nodes successively. The average sending rate per node is once every 4 seconds. Figure 6 and Figure 7 show the latencies and radio duty cycles associated with the two improved acknowledgment schemes. We can see that the MAC acknowledgment are slightly more efficient than the quick link acknowledgment, but the difference becomes smaller as congestion increases.

C. Code and Data Memory Usage

Table III shows the code and static data memory usage of each module of our link level service. The code footprint of

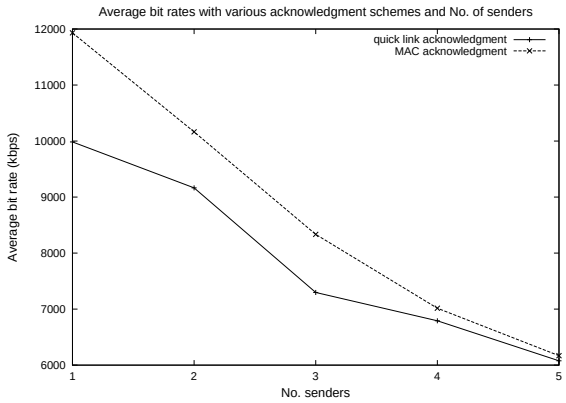

Fig. 6. Average raw bit rates with multiple senders and 10% X-MAC duty-cycle. The MAC acknowledgment is slightly faster than the quick link acknowledgment.

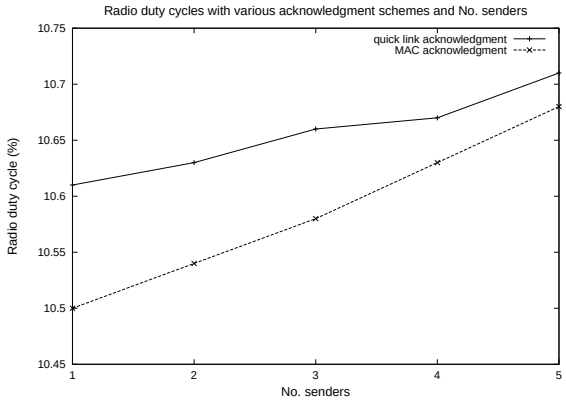

Fig. 7. Radio duty cycles with multiple senders and 10% X-MAC duty-cycle. Both acknowledgment schemes achieve bandwidth efficiency close to the 10% X-MAC duty-cycle.

the abstraction layer and the neighbor table utility amounts to 1406 bytes, significantly smaller than the approximately 4kB used by SP in TinyOS 1.x [16]. The total code size of the service amounts to 5300 bytes. Note that a large proportion of the static data memory is taken up by message buffers, whose sizes are configurable.

VI. Discussions

Some developers of SP and NLA have acknowledged in a recent reflection that an overall sensor node software architecture is 'too much, too soon' [17]. Particularly, the vision of a dominant 'narrow-waist' service enforcing API conformance

TABLE III
CODE AND DATA MEMORY USAGE OF MODULES OF THE LINK SERVICE IN BYTES

Module	Code	Static Data
Abstraction layer	742	126
Neighbor table	664	160
ARQ layer	858	686
X-MAC layer with quick acknowledgment	848	284
CC2420 radio driver	2188	26

of new protocols has given way to a more pragmatic approach of consolidating existing lower level mechanisms. We regard both Chameleon and MLA to reflect this trend toward finer grained components. In light of this, a more conscious distinction between collaborative modules and reusable components, as suggested by Messerschmitt [18], may help us to clarify the problem. Modules within a reference architecture are the result of decomposing a system into interoperable and complementary parts. When the system functionality changes, which is a common case for sensor networks, the decomposition may need to change also. Components specializing in more concrete functionality, on the other hand, are less dependent on system context and tend to survive architectural changes. The neighbor table and the ARQ layer of our link service are designed with such kind of component reuse in mind.

VII. Conclusions

Based on our study of some design principles and outstanding difficulties embodied in previous work, we have built a layered link-level message service for sensor networks. We have identified several critical design and implementation trade-offs that have significant influence on software modularity and protocol performance. Late semantics binding is conducive to better abstraction of protocol logic, so long as the expressivity of the abstraction is sufficient to represent lower level mechanisms. A minimal set of implicitly shared information between adjacent layers are desirable for modularity concerns, but needs to include necessary support for resolution of layer interdependencies such as time scope initialization. Expansion of inter-layer interactions is needed to meet performance goals, as we have done with the improved ARQ/MAC acknowledgment schemes.

References

[1] D. Culler and P. Dutta and Cheng T. E. and R. Fonseca and J. Hui and P. Levis 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.

[2] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A unifying link abstraction for wireless sensor networks." in *SenSys*, 2005.

[3] Cheng T. E. and R. Fonseca and S. Kim and D. Moon and A. Tavakoli and D. Culler and S. Shenker and I. Stoica, "A modular network layer for sensornets," in *Proceedings of OSDI*, Aug. 2006.

[4] A. Dunkels and F. Österlind and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys 2007)*, Sydney, Australia, Nov. 2007.

[5] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A component-based architecture for power-efficient media access control in wireless sensor networks," in *Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys 2007)*, Sydney, Australia, Nov. 2007.

[6] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, Boulder, Colorado, USA, 2006, pp. 307–320.

[7] "IEEE Standard for Information technology – Telecommunication and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 2: Logical Link Control," IEEE Computer Society, New York, NY, USA, 1998.

[8] P. Levis, D. Gay, V. Handziski, J. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk *et al.*, "T2: A second generation os for embedded sensor networks," Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universitat Berlin, 2005, Tech. Rep.

[9] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, Nov. 2004.

[10] J. Polastre, "A unifying link abstraction for wireless sensor networks," PhD Dissertation, Univerisity of California, Berkerly, 2005.

[11] V. Srivastava and M. Motani, "Cross-layer design: a survey and the road ahead," *Communications Magazine, IEEE*, vol. 43, no. 12, pp. 112–119, 2005.

[12] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four-bit wireless link estimation," in *ACM HotNets-VI*, 2007.

[13] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2004, pp. 95–107.

[14] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. IPSN/SPOTS'05*, Los Angeles, CA, USA, Apr. 2005.

[15] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based online energy estimation for sensor nodes," in *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, Jun. 2007.

[16] A. Tavakoli, J. Taneja, P. Dutta, D. Culler, S. Shenker, and I. Stoica, "Evaluation and Enhancement of a Unifying Link Abstraction for Sensornets," *Under submission*.

[17] A. Tavakoli, P. Dutta, J. Jeong, S. Kim, J. Ortiz, D. Culler, P. Levis, and S. Shenker, "A Modular Sensornet Architecture: Past, Present, and Future Directions," in *Proceedings of the International Workshop on Wireless Sensor Network Architecture (WWSNA '07)*, 2007.

[18] D. Messerschmitt, "Rethinking Components: From Hardware and Software to Systems," *Proceedings of the IEEE*, vol. 95, no. 7, pp. 1473–1496, 2007.