DPLC: Dynamic Packet Length Control in Wireless Sensor Networks

Wei Dong[†], Xue Liu[⋆], Chun Chen[†], Yuan He[‡], Gong Chen[†], Yunhao Liu[‡], and Jiajun Bu[†]
[†]Zhejiang Key Lab. of Service Robot, College of Comp. Sci., Zhejiang University
[⋆]School of Comp. Sci., McGill University
[‡]Dept. of Comp. Sci. & Eng., Hong Kong University of Science and Technology
{dongw, chenc, chengong, bjj}@zju.edu.cn, xueliu@cs.mcgill.ca, {heyuan, liu}@cse.ust.hk

Abstract—Previous packet length optimizations for sensor networks often employ a fixed optimal length scheme, while in this study we present DPLC, a Dynamic Packet Length Control scheme. To make DPLC more efficient in terms of channel utilization, we incorporate a lightweight and accurate link estimation method that captures both physical channel conditions and interferences. We further provide two easy-to-use services, i.e., small message aggregation and large message fragmentation, to facilitate upper-layer application programming. The implementation of DPLC based on TinyOS 2.1 is lightweight, with respect to computation, memory, and header overhead. Our experiments using a real indoor testbed running CTP show that DPLC results in a 13% reduction in transmission overhead and a 41.8% reduction in energy consumption compared with the original protocol, and a 21% reduction in transmission overhead and a 15.1% reduction in energy consumption compared with simple aggregation schemes.

I. Introduction

A fundamental challenge in wireless networks is that radio links are subject to transmission power, fading, and interference, which degrade the data delivery performance. This challenge is exacerbated in wireless sensor networks (WSNs), where severe energy and resource constraints preclude the use of many sophisticated techniques that may be found in other wireless systems [1]. For example, (i) bit rate adaptation protocols [2] demand special hardware that is not available on general sensor nodes; (ii) effective forward error correction (FEC) requires the amount of redundant data transmitted to be tuned to match the link qualities [3], which is difficult to achieve in dynamic WSNs; (iii) other sophisticated coding schemes [4], [5] incur delays and computation overheads that are only suitable for specific applications.

In this paper, we consider a simple, cost-effective solution based on the technique of dynamic packet length control to improve the performance in these varying conditions. A trade-off exists between the desire to reduce header overhead by making packet large, and the need to reduce packet error rates (PER) in the noisy channel by using small packet length [6].

Although there have been several studies on packet length optimizations in the literature [6]–[11], existing approaches are not practically applicable to the sensor network scenario. A packet optimization scheme applicable in sensor networks must have the following important features.

(i) *Dynamic* packet length adaptation. Prior work in sensor networks uses fixed packet length optimization schemes [10], [11]. These schemes are not preferred due to the spatial-temporal diversity of link qualities in WSNs.

(ii) *Accurate* link estimation. The performance improvement of the packet length adaptation scheme is highly dependent on the link estimation accuracy. Prior work in wireless systems does not consider unique characteristics of WSNs, e.g., resource constraints of sensor nodes [6], [8], [9], thus leading to inaccurate link estimation in sensor networks.

(iii) *Easy* to use. To the best of our knowledge, no prior work addresses the application programmability issues of packet length adaptation scheme in sensor networks. Without substantial programming efforts, there is still a huge gap between theoretical optimality and practically achievable gains.

This paper presents DPLC, a Dynamic Packet Length Control scheme that overcomes the limitations of prior work mentioned above. Our work is motivated by a recently deployed sensor network system GreenOrbs, which aims to achieve large-scale and long-term surveillance in the forest [12], [13]. The temporal-spatial variance in wireless link qualities inspires us to use a dynamic packet length control scheme to improve the data delivery performance. First, DPLC utilizes a *dynamic* packet length adaptation scheme that suits the resource constraints of sensor nodes. DPLC adds only two bytes (at most) per MAC frame (hereafter, we use frame to refer to packet sent by the radio, message to refer to packet seen by application programmers, and packet to refer to both if there is no ambiguity). Second, DPLC is built upon an *accurate* link estimation method based on passive observations of packet receptions. Both physical channel conditions (due to channel fading, mobility, or power degradation) and interferences (from exposed and hidden terminals) can be captured. Third, DPLC includes two *easy-to-use* services, i.e., aggregation service (for small messages) and fragmentation service (for large messages), to facilitate upper-layer application programming.

We implement DPLC based on TinyOS 2.1, the standard sensor network OS in the literature [14]. The current implementation of DPLC on TelosB motes is lightweight. The computation overhead is at most 0.48 ms per transmitted or received frame, which is small compared to the packet transmission time plus MAC backoff time. The RAM and

ROM overhead are 0.37 KB and 4.5 KB respectively, which fit well in current sensor nodes. The header overhead is only two bytes (at most) per frame. In addition, we propose techniques to further optimize for various upper-layer usage models, e.g., we aggregate layer 2 (hereafter L2) ACKs to mitigate ACK overhead for both unreliable transmissions of small messages and reliable transmissions of large messages; we incorporate a lightweight and distributed scheme to further optimize end-to-end data delivery performance of a single flow.

We evaluate DPLC through both simulations and testbed experiments. The simulation results of TOSSIM [15] validate our design. Real indoor testbed experiments consisting of 20 TelosB motes, running the CTP protocol [16], show that DPLC results in a 13% reduction in transmission overhead and a 41.8% reduction in energy consumption compared to the original protocol, and a 21% reduction in transmission overhead and a 15.1% reduction in energy consumption compared to a simple aggregation scheme (that always transmits the largest packets supported by the radio).

The rest of this paper is structured as follows. Section II describes the experiment observations that motivate our design. Section III presents the design of DPLC. Section IV introduces the implementation details. Section V shows the evaluation results. Section VI discusses related work. Finally, Section VII concludes this paper.

II. Motivation

To better understand how a practical packet length adaptation scheme should be designed, we ran a series of experiments to provide insights that guide our design.

A. Packet Length Optimization

To see the impact of variable packet length to the system performance in WSNs, we setup two TelosB motes which communicate variable packets at a distance of approximately 8 meters with transmission power level 3 in a quiet indoor environment. We measure the packet reception rate (PRR) and the transmission efficiency ($\mathcal{E}$, which is defined as the received useful bytes divided by the overall transmitted bytes) for each packet payload length. A high $\mathcal{E}$ value indicates a high goodput and a high energy efficiency provided that the transmission time and the transmission energy consumption are approximately linear to the packet length. We ran each experiment 30 times, and Figure 1 shows the mean value and the standard deviation of PRR and $\mathcal{E}$. We can see that with the packet payload length increases, PRR decreases while $\mathcal{E}$ reaches its maximum value at packet payload length 30 (bytes). Using the optimal packet length improves approximately 40% performance in terms of $\mathcal{E}$ compared to using the smallest packet length in this case.

We have also run other experiments at different distances and transmission powers. Results show that the optimal packet length varies in different conditions. Therefore, *packet length optimization is beneficial in WSNs*.

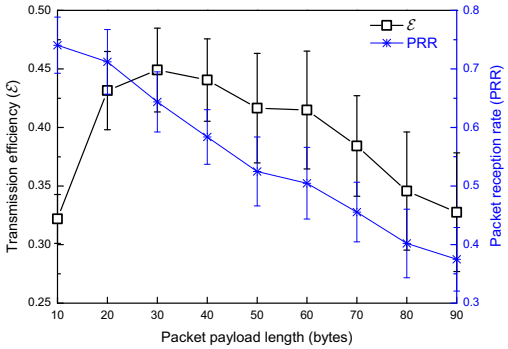


Fig. 1: Impact of packet payload length on transmission efficiency ($\mathcal{E}$) and packet reception rate (PRR).

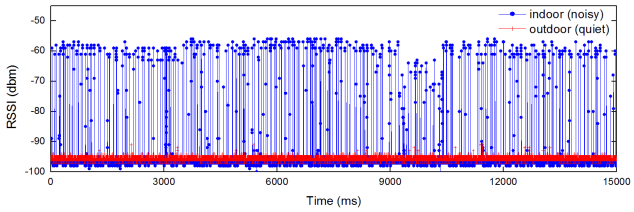


Fig. 2: Comparison of RSSI measurements indoor (noisy) and outdoor (quiet).

B. Dynamic Packet Length Adaptation

Wireless link qualities can be greatly affected by environmental factors at different locations [13]. To see how link diversity is, we setup one TelosB mote to measure the received signal strength at a resolution of 1 ms. The experiments are conducted both indoor (the environment is noisy because of 802.11 interferences) and outdoor (the environment is quiet), respectively. Figure 2 plots the RSSI (received signal strength indicator) value read from the CC2420 radio chip. We can see that the channel conditions vary drastically: in the noisy environment the RSSI value can be as high as -62 dbm while in the quiet environment the RSSI value can be as low as -96 dbm.

This indicates that *packet length adaption schemes must be dynamically adapted to physical channel conditions to deliver performance gains in WSNs with spatial-temporal diversity in link qualities*.

C. Link Estimation

A common approach to estimate link qualities is to directly use RSSI or LQI (link quality indicator) provided by the radio hardware [17]. To see how RSSI and LQI reflect link qualities, we setup three TelosB motes. Node 1 transmits normal TinyOS packets to node 2 at an interval of 128 ms with transmission power level 2. After 60 seconds, we introduce node 3 as a hidden terminal to node 1 that also transmits packets to node 2 at an interval of 128 ms. Figure 3 shows the results. In the measured PRR, we clearly see that the link quality degrades after 60 s. The RSSI and LQI value

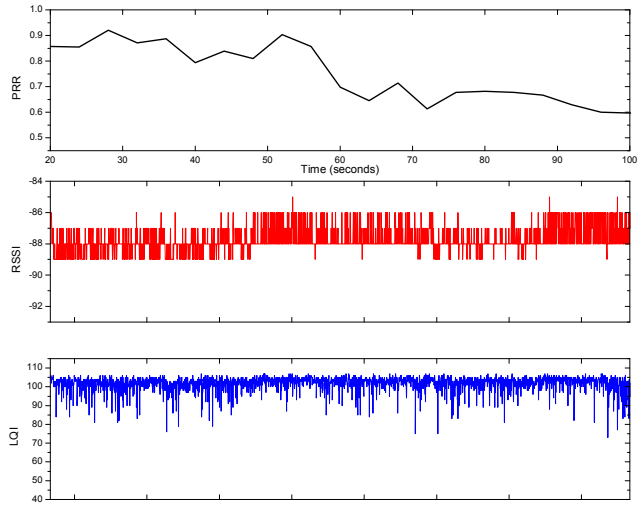


Fig. 3: Comparison of link estimation methods (PRR-based, RSSI-based, and LQI-based).

measured at node 2, however, do not see such a degradation. This indicates that directly measured RSSI and LQI are not enough for accurately estimating link qualities.

It is also worth mentioning that link estimation using proactive beaconing is also inappropriate in our scenario, because the data packet length is different from that of beacons, and it varies with link conditions. Therefore, *data-plane PRR statistics are needed in order to improve the accuracy of link quality estimations*.

III. Design

In this section, we present the design of DPLC, a dynamic packet length control scheme for WSNs. Below, we identify the major design goals.

- Dynamic adaptation. DPLC should provide a dynamic adaptation scheme to achieve performance improvements in dynamic, time-varying sensor networks.
- Accurate link estimation. DPLC should implement an accurate link estimation method that can capture both physical channel conditions (due to channel fading, mobility, or power degradation) and interferences (from exposed and hidden terminals).
- Ease of programming. DPLC should provide easy-to-use services to facilitate upper-layer application programming.
- Lightweight for implementation. DPLC should be lightweight for resource constrained sensor nodes.

We will see how DPLC achieves the first two design goals, i.e., dynamic adaptation and accurate link estimation in Section III-C. The remaining two design goals, i.e., ease of programming and lightweight for implementation, will be elaborated in Section IV-A and Section IV-B respectively.

A. Overview

The DPLC scheme works as follows. The application passes an application-level message for transmission. The DPLC module at the sender decides whether to use the aggregation service (AS, if the message length is small) or the fragmentation service (FS, if the message length is larger than the maximum packet length supported by the radio, i.e., 128 bytes for CC2420). The link estimator within DPLC dynamically estimates the optimal packet length for transmission. Based on this, the DPLC module at the sender decides how many messages should be aggregated (for AS), or how many frames the message should be fragmented into (for FS). When a frame is ready for transmission (enough messages have been aggregated or time is out in AS), we actually send it via the MAC layer. When the DPLC module at the receiver receives a MAC frame, it deaggregates or defragments the frame in order to obtain the original message. When the message is ready (all frames in the message have received or the receive buffer is full in FS), the DPLC module at the receiver notifies the upper layer for further handling.

As mentioned above, the DPLC scheme provides two services for upper-layer applications, i.e., the aggregation service (AS, for small messages) and the fragmentation service (FS, for large messages).

(i) AS is useful for small data collection, e.g., CTP [16]. AS provides three distinct mechanisms, i.e., reliable transmissions (AS_∞), unreliable transmissions with fixed number of retransmissions (AS_n, where $n \geq 1$ is the retransmission number), and unreliable transmissions (AS_0). Both AS_∞ and AS_n requires L2 ACKs provided by the link layer, because packets need to be retransmitted (at least once) when they are lost. For AS_0, we additionally provide a more efficient ACK scheme called AggAck that does not rely on L2 ACKs, and thus mitigate the ACK overhead (we use AS_0-L2 to denote AS_0 with L2 ACKs and AS_0-AA to represent the one with AggAck afterwards).

(ii) FS is useful for bulk data transmission, e.g., Flush [18]. FS provides reliable transmissions as a large message is usually very important for upper-layer applications. FS does not necessarily depend on L2 ACKs. As mentioned above, we additionally provide the AggAck mechanism to mitigate the ACK overhead, and more importantly, to deal with data packet retransmissions (we use FS-L2 to denote FS with L2 ACKs and FS-AA to represent the one with AggAck afterwards).

The link estimation method within DPLC dynamically estimates the link quality based on passive observations of packet receptions [19]. For each outgoing link, the sender keeps a sliding window in which each bit tracks whether the sent packet was ACKed. Based on this, it tunes the packet length for a given link. The link estimation method does not necessarily depend on L2 ACKs. As mentioned above, we additionally provide AggAck to provide such information. The advantages of AggAck are three-fold. First, for AS_0 and FS, it mitigates L2 ACK overhead for each transmitted packet. Second, for FS, it reduces redundant data packet retransmission in asymmetric links (in which the reverse link quality for ACK is poor). Finally, in the presence of asymmetric links, it provides an accurate estimation of directional links which is important to optimize multi-hop data delivery performance of a single flow.

B. Metrics for Dynamic Adaptation

For dynamic adaptation, we use the metric of transmission efficiency ($\mathcal{E}$) which is defined as the received useful bytes divided by the overall transmitted bytes. Generally, as the transmission time and the transmission energy consumption are approximately linear to the transmitted bytes, a high $\mathcal{E}$ value indicates a high goodput and a high energy efficiency. The goal of our dynamic adaptation scheme is hence to maximize the $\mathcal{E}$ value. The calculation of $\mathcal{E}$ is described below for different transmission patterns.

Metric for Single-hop Transmission. For a single-hop transmission link from n_i to n_{i+1}, the value of $\mathcal{E}_i$ equals to the single-hop transmission efficiency, denoted as ε_i. The value of ε_i equals to the received useful bytes (at n_{i+1}) divided by the transmitted bytes (at n_i). It is specified as follows,

$$\mathcal{E}_i = \varepsilon_i(l) = \frac{l \cdot p(l)}{l + H + O} \tag{1}$$

where l is the packet payload length (for MAC transmission), $p(l)$ is the PRR from n_i to n_{i+1} given packet payload length l, H is MAC header overhead, and O is the additional header overhead introduced by DPLC. The strategy for single-hop transmission is hence to monitor the $p(l)$ (detailed in Section III-C) and decide the packet length that maximizes this metric.

Metric for Multi-hop Transmission. For a multi-hop transmission from n_k to n_{k+1} (originated from n_1), the value of $\mathcal{E}_k$, equals to the received useful bytes (at n_{k+1}) divided by the *overall* transmitted bytes (from n_1 to n_k). It is specified as follows,

$$\mathcal{E}_k = \frac{1}{\varepsilon_k^{-1}(l) + \mathcal{E}_{k-1}^{-1}\frac{1}{dr_k(l)}} \tag{2}$$

where $\varepsilon_k^{-1}(l) = \frac{1}{\varepsilon_k(l)}$ is the normalized transmission overhead from n_k to n_{k+1} (i.e., the transmission overhead at n_k divided by the received useful bytes at n_{k+1}). $\mathcal{E}_{k-1}^{-1} = \frac{1}{\mathcal{E}_{k-1}}$ is the normalized transmission overhead from n_1 to n_k. $dr_k(l)$ is the data delivery rate from n_k at packet payload length l (it differs from PRR in that retransmissions can improve the data delivery rate). Note that n_k must receive as much as $\frac{1}{dr_k(l)}$ useful bytes to ensure that n_{k+1} receives 1 byte. We note that $\mathcal{E}_1 = \varepsilon_1$, which represents the single-hop case. We can see from Eq. (2) that both $dr_k(l)$ and $\mathcal{E}_{k-1}^{-1}$ need to be tracked in addition to $p(l)$. We first separate two cases to estimate $dr_k(l)$ in term of PRR, and then we introduce how to track $\mathcal{E}_{k-1}^{-1}$ accurately in Section III-E.

- The case for reliable multi-hop transmissions. For reliable transmissions, $dr_i = 1$ $(1 \le i \le k)$. Hence,

$$\mathcal{E}_k = \frac{1}{\varepsilon_k^{-1}(l) + \sum_{i=1}^{k-1}\varepsilon_i^{-1}} \tag{3}$$

  Maximizing Eq. (3) is equivalent to maximizing Eq. (1) (i.e., we do not need to track $\mathcal{E}_{k-1}^{-1}$).
- The case for unreliable multi-hop transmissions with fixed number of retransmissions. For unreliable transmission with m number of retransmissions ($m \ge 0$), the data delivery rate relates to PRR as follows,

$$dr_i = 1 - (1 - p_i)^{m+1} \tag{4}$$

  where p_i is the PRR from n_i to n_{i+1} $(1 \le i \le k)$. Therefore, $dr_k(l) = 1 - (1 - p_k(l))^{m+1}$, i.e., dr_k can be estimated from the PRR observed at n_k. We still need to get the value of $\mathcal{E}_{k-1}^{-1}$, which will be described in Section III-E.

C. Description of DPLC

We incorporate the link estimation algorithm in [19] to dynamically control the packet length. The DPLC scheme has the following features. (i) It passively monitors packet receptions to dynamically adjust the packet length. For transmissions that enable L2 ACKs, this incurs no additional overhead. In addition, we provide the AggAck mechanism to mitigate L2 ACKs for unreliable aggregation service and reliable fragmentation service. (ii) It provides accurate link estimations that can capture both physical channel conditions (due to channel fading, mobility, or power degradation) and interferences (from exposed and hidden terminals). (iii) It is lightweight for implementation on resource-constrained sensor nodes. e.g., for each outgoing MAC frame, it adds only two bytes (at most) header overhead.

DPLC individually tunes the packet length on each outgoing link. A link is initially set to transmit at its default granularity (which equals to the message payload length for AS and 10 bytes for FS in our current implementation). DPLC monitors all packet receptions by keeping a sliding window of size w. When the window is filling, DPLC stays in the `INIT` state. When the window is full, DPLC computes the metric as described in Section III-B. Then it enters into the `TRY` state, increasing or decreasing the packet length by the granularity. Whether to increase or decrease the packet length depends on whether a gradient variable is positive or negative (we initially set the gradient variable to be positive). When $\alpha \cdot w = \frac{2}{3}w$ packet receptions are monitored, DPLC compares the metrics to see whether the `TRY` state improves the performance. If the performance increases, it keeps the state unchanged, and waits until it transits into the `STEADY` state. If the performance decreases, it restores the original packet length and reenter into the `INIT` state.

The gradient variable described above decides whether to increase the packet length or decrease the packet length. When the gradient is positive, DPLC tries to increase the packet length while vice versa. DPLC inverts the gradient when it sees the performance decreases or it has already reached the minimum or maximum packet length.

D. The AggAck Mechanism

We additionally provide the AggAck mechanism to mitigate the ACK overhead for unreliable aggregation service (AS_0) and reliable fragmentation service (FS). For AS_∞ and AS_n ($n \ge 1$), L2 ACK is required because data packets need to be retransmitted when they are lost.

(i) For AS_0, we use a sender-initiated AggAck mechanism, i.e., the sender requests for an ACK at the end of a sliding window. The request is piggybacked in the data packet, and we keep on requesting until an ACK is received. We can do this because if the transmission is unreliable we can proceed to send the next data packet (and piggyback the ACK request) after the previous packet is sent out. We do not change the packet length until an ACK is received. On receiving an ACK request, the receiver sends out the ACK without MAC layer carrier sense assessment (CCA). As indicated in [20], this synchronous mechanism improves the ACK reliability significantly. The AggAck carries only one byte information, i.e., the received packet number in the current window, which is used by the sender to compute the PRR.

(ii) For FS, we use a receiver-initiated AggAck mechanism. The reason is that we can not keep on piggybacking the ACK request in the next data packet when the window is full and we are not sure whether all packets in the window have been received. To avoid sending a separate ACK request packet, we use the timing information at the receiver side to automatically send an AggAck if no data packets have been received in a short timeout. Because FS is reliable, the AggAck carries a bitmap indicating which packets in the current window are lost. This mechanism is similar to the NACK mechanism employed in the Deluge protocol [3]. FS can also use L2 ACKs. However, we prefer to use the AggAck mechanism. The reason is twofold. First, as mentioned above, the AggAck mechanism mitigates the ACK overhead compared to L2 ACKs which are transmitted after every data receptions. Second, in the presence of asymmetric links, L2 ACKs can be lost. In this case, the data packets will be redundantly retransmitted.

E. Handling Multi-hop Traffic

As described in Section III-B, we need to obtain $\mathcal{E}_{k-1}^{-1}$ for metric calculation at n_k in the case of unreliable multi-hop transmissions. For a single flow originated from n_1 to n_{k+1}, we piggyback $\mathcal{E}_i$ calculated at n_i to data packets which are sent to the next hop. n_{i+1} can thus calculate $\mathcal{E}_{i+1}$. In this manner, the $\mathcal{E}$ value at each node can be calculated hop by hop.

It is worth noting that in this case accurate estimation of PRR is important as an inaccurate PRR value will affect the calculation of $\mathcal{E}$ at downstream nodes, which again impacts the choice of optimal packet length at these links. This indicates that in this case the AggAck mechanism is favored over L2 ACKs because of the existence of asymmetric links (in this case PRR estimation based on L2 ACKs will be inaccurate). It is also worth noting that in the single-hop case, however, asymmetric links will not impact the choice of optimal packet length because the ACK loss rate is irrelevant to the packet length (i.e., the metric $\mathcal{E}_i' = \varepsilon_i \cdot PRR_{\text{ack}}$ is equivalent to the metric described by Eq. (1)).

Optimizing the performance for multi-flows in the strict sense is quite complicated that we believe it exceeds the capabilities of current sensor nodes. This is, however, not a major concern for the following two reasons. First, one can have the sink schedule the transfers from each node one at a time, in a round-robin fashion, as suggested in [18]. Second, in practice, as in the CTP data collection protocol [16], optimizing single hop transmissions (using the metric described by Eq. (1)) leads to near-optimal performance because of high data collection reliability (i.e., $dr \approx 1$, hence the metric described by Eq. (2) is approximately equivalent to the metric described by Eq. (1)).

```
interface DMSend {
  command error_t send(am_addr_t addr,
                       void* msg,
                       uint16_t len);
  event void sendDone(void* msg, error_t err);
  command void* getPayload();
  command void setLinkAck(bool ack);
  command void setRetries(uint8_t num);
  command void flush(am_addr_t addr);
  command void setMaxLen(uint8_t len);
  command void setMinLen(uint8_t len);
}
```

(a) The `DMSend` interface

```
interface DMReceive {
  event void* receive(void* msg, uint8_t len,
                      uint16_t pktlen);
  command message_t* getAM(message_t* msg);
}
```

(b) The `DMReceive` interface

Fig. 4: Two interfaces provided by DPLC: the `DMSend` interface and the `DMReceive` interface.

IV. Implementation

We implement DPLC based on TinyOS 2.1. In this section, we describe DPLC's implementation details. First, we introduce the programming interface of DPLC to show that it is easy to use for upper-layer application programming (Section IV-A). Second, we evaluate DPLC's implementation overhead to show that it is indeed lightweight for current sensor nodes (Section IV-B).

A. Programming Interface

The DPLC module provides two interfaces for application programming, i.e., `DMSend` and `DMReceive` (see Figure 4). They are very similar to the TinyOS `AMSend` and `Receive` interfaces.

(i) We provide the `DMSend` interface for message transmission. The `send` command is used to transmit both a small message (needs aggregation) and a large message (needs fragmentation). Whether to aggregate or fragment depends on the message length. The `sendDone` event is signaled when the message is actually sent out by the radio. For AS, the `getPayload` command is used to get the payload field of the internal `max_message_t` structure (which is used to send packets of variable length). The `setLinkAck` command is

used to enable or disable L2 ACKs. The `setRetries` command is used to set the retry number for retransmission. The `flush` command is used to flush any buffered packets to the MAC layer. The `setMaxLen` and `setMinLen` commands are used to set the maximum and minimum packet payload length for transmission.

(ii) We provide the `DMReceive` interface to receive a message. The `receive` event will be signaled which a message has been received. For FS, the application allocates a pre-defined message buffer for receiving. When either a message is received in its entirety or the receiving buffer is full, the `receive` event will be signaled. The third parameter, `pktlen`, indicates the total packet length (which may exceed the MAC frame length). The `getAM` command return a TinyOS compatible `message_t` structure for retrieving the metadata, such as RSSI readings and LQI readings, etc.

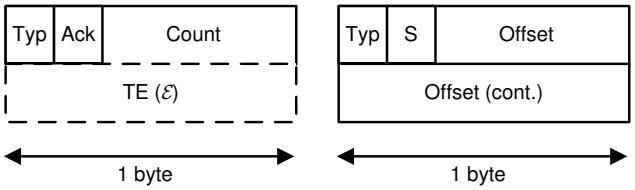
Generally, application programming using `DMSend` is not much different from that using `AMSend`. DPLC handles all specific details about aggregation/deaggregation, fragmentation/defragmentation such that upper-layer applications need not care about how to use the optimal packet length in the MAC layer. Without such a scheme, the message length will need to be manually changed every time the optimal packet length changes, leading to time consuming modifications and cumbersome design. By isolating packet length adaptation decisions into L2.5, DPLC reduces such cross-layer dependencies, thus leading to a lower cost to application programming.

B. Overhead

This section analyzes DPLC's implementation overhead in terms of computation overhead, memory overhead, and header overhead, respectively.

Computation Overhead. DPLC incurs computation overhead mainly in `send`, `receive`, and decision making algorithm (described in III-C) (i) The extra overhead of `DMSend.send` is only about 30 μs before calling `AMSend.send`. (ii) The extra overhead of `DMReceive.receive` is about 170 μs due to deaggregation or defragmentation. (iii) The extra overhead of the decision making algorithm is at most 480 μs (at `sendDone` or AggAck is received). Overall, we think that this overhead is acceptable. First, data rate in most sensor network application is low. Therefore, throughput is not a major concern. Second, a maximum delay of 0.48 ms per MAC frame is small compared to the frame transmission time (which is about 1.6 ms for a normal 40-byte TinyOS packet) plus the backoff time (which is about 5 ms in average for an initial transmission in TinyOS).

Memory Overhead. DPLC incurs memory overhead on RAM (data) and ROM (program). (i) DPLC needs about 256 bytes for a sending message buffer and receiving message buffer. It needs (w/8+6) for each outgoing link. With $w = 24$ and neighbor number equals to 8, this consumes 72 bytes. Additionally, DPLC needs 47 bytes local variables. Overall, this adds up to 375 bytes. This overhead is small compared to 10 KB RAM in TelosB. (ii) To evaluate DPLC's ROM overhead, we compare the `RadioCountToLeds` benchmark using the default AM message and the same benchmark using interfaces described in Section IV-A. We have found that the original benchmark consumes 11528 bytes ROM while the modified version consumes 16052 bytes ROM. This indicates the DPLC module consumes approximately 4.5 KB ROM, which is acceptable compared to 48 KB ROM in TelosB.



(a) DPLC header for AS (b) DPLC header for FS

Fig. 5: DPLC data packet headers for AS and FS.

Header Overhead. We add minimal header overhead to both data packets (at most 2 bytes) and AggAck packets. (i) Data packet overhead (see Figure 5). For AS, each frame adds the following header overhead. The `Typ` field (1 bit) is used to indicate that it is for AS. The `Ack` field (1 bit) is used to indicate whether an AggAck is requested. The `Count` filed (6 bits) is used to indicate how many messages are aggregated in this frame. The `TE` field (8 bits) is used to carry the $\mathcal{E}$ value of the previous hop. Note that it is only used for optimization in multi-hop data delivery of a single flow. For FS, each frame also adds the following header overhead. The `Typ` field (1 bit) is used to indicate that it is for FS. The `S` field (1 bit) is used to indicate whether it is the start frame of a (large) packet. The `Offset` field (14 bits) is used to indicate the offset of the current received frame within the (large) packet (used to reassemble the packet at the receiver). (ii) AggAck packet overhead. For AS, the AggAck payload length is only 1 byte, which is the received packet number in the current window. For FS, the AggAck payload length is w/8 (which equals to 3 bytes in our current implementation). Each bit indicates whether the corresponding data packet has been received, and needs to be retransmitted.

V. Evaluation

We evaluate DPLC through both simulation in TOSSIM [15] and real indoor testbed experiments consisting of 20 TelosB motes running the CTP protocol [16]. TOSSIM simulation results validate DPLC's design in a controlled manner while testbed experiments show how DPLC can be utilized to improve performance of existing sensor network protocols and applications.

A. Validation

The major goals of the experiments described in this section is to demonstrate DPLC's achievable improvements in terms of the $\mathcal{E}$ value, and to validate some important designs within DPLC (e.g., the AggAck mechanism).

For these simulations, we modify TOSSIM (version 2) to use the bit error model, i.e., $PRR = (1-BER)^{L+H+O}$, where

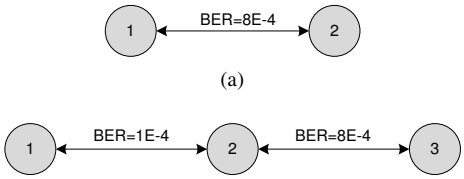
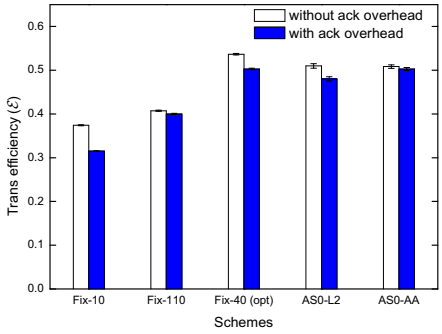
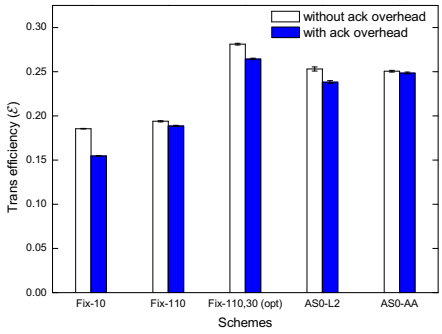


Fig. 6: Two topologies for validating DPLC's design. (a): single hop (symmetric link with median link quality). (b): two hops (symmetric links in which the quality of the first link is high and the quality of the second link is median).



(a) Evaluation of AS_0 in topology (a)



(b) Evaluation of AS_0 in topology (b)

Fig. 7: TOSSIM simulation results. The notation "Fix-<len>" refers to the scheme using a fixed packet length of <len>. In Figure 7b, the notation "Fix-<len1,len2>" refers to the scheme using a fixed packet length of <len1> in the first hop and a fixed packet length of <len2> in the second hop. The optimal scheme is labeled "(opt)".

$H = 13$ for TinyOS, and $O = 1$ or 2 for DPLC. Figure 6 shows the topologies for validation. Basically, we validate DPLC in both single-hop case and multi-hop case with varying link qualities (e.g., BER=1×10^{-4} or BER=8×10^{-4}). We run each experiment 5 times in each topology.

Figures 7a, 7b show the performance of AS_0 in topology (a) and topology (b) respectively. In Figure 7a, the notation "Fix-<len>" refers to the scheme using a fixed packet length of <len>. In Figure 7b, the notation "Fix-<len1,len2>" refers to the scheme using a fixed packet length of <len1> in the first hop and a fixed packet length of <len2> in the second hop. In both figures, the optimal scheme is labeled "(opt)".

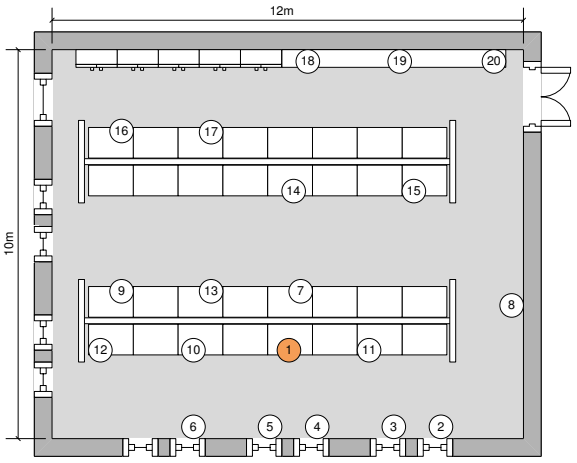


Fig. 8: Indoor testbed deployment where node 1 is the sink.

We measure the transmission efficiency of each scheme in two ways: the transmission efficiency without ACK overhead (in which the ACK overhead is not accounted for), and the transmission efficiency with ACK overhead (in which the ACK overhead is also accounted for). We can see that in both cases, DPLC achieves more than 90% performance compared to the optimal scheme. We can also see that the AggAck mechanism mitigates the ACK overhead, leading to about 5% improvement considering the ACK overhead.

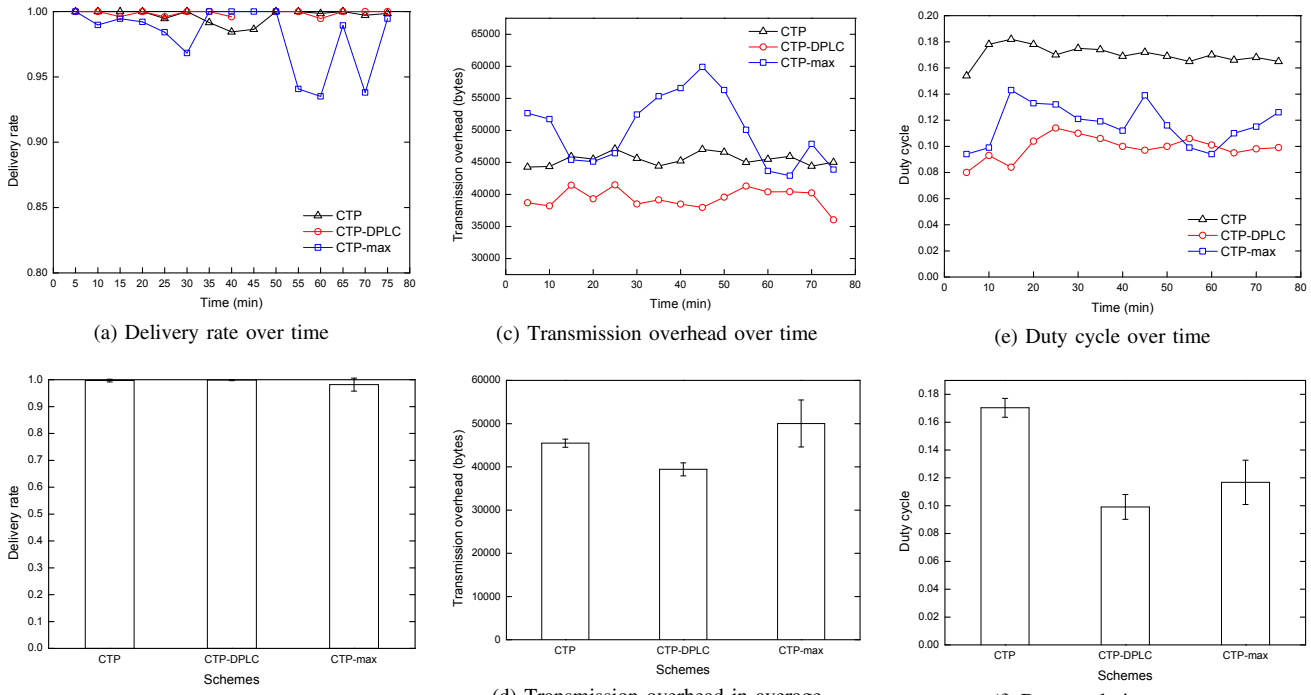
B. Testbed Results

We use a real indoor testbed consisting of 20 TelosB motes running the CTP protocol [16] to evaluate how DPLC can improve the performance of state-of-the-art sensor network protocols and applications. CTP is a data collection protocol that dynamically selects the best route to the sink according to a hybrid link estimation algorithm [16].

CTP needs to be slightly modified. The major modification is located at the data sending logic. We need to use `DMSend` instead of `AMSend`. The changes to CTP keep within approximately 100 lines of code.

We setup 20 TelosB motes in our lab in a Saturday and a Sunday without human interventions (as shown in Figure 8). We use the `TestNetwork` benchmark in TinyOS (transmission power is 3; retransmission count is 4), to evaluate three schemes, i.e., the original CTP, CTP using DPLC (CTP-DPLC), and CTP always using the maximum packet length (CTP-max). Each `TestNetwork` node sends a packet (of payload length 15) periodically to the sink at an interval of 20 seconds. We run each scheme at least 2 hours (we do not analyze data in the first 30 min warmup time) with and without the use of lower power listening (LPL, in which the sleep interval is set to 500 ms). After a total of 12 hours, we collect the statistics to evaluate how our scheme impacts (i) reliability of data collection, (ii) transmission overhead, (iii) energy efficiency.

Reliability. Figure 9a shows the collection reliability in term of data delivery rate varied with time. We can see that the



(a) Delivery rate over time

(b) Delivery rate in average

(c) Transmission overhead over time

(d) Transmission overhead in average

(e) Duty cycle over time

(f) Duty cycle in average

Fig. 9: Indoor testbed results.

CTP-max scheme is less stable than the other two schemes. The reason is due to the fact that larger packet are more suspectable to wireless loss. Figure 9b shows the average collection reliability in term of data delivery rate for a duration of 75 min. We can see that CTP-max slightly reduces the reliability while DPLC remains the high reliability of the original CTP.

Transmission Overhead. With data collection reliability almost the same (i.e., with almost the same amount of received useful bytes), optimizing the $\mathcal{E}$ value translates to optimizing the transmission overhead. We would like to see how DPLC reduces the transmission overhead. Figure 9c shows the transmission overhead of 19 nodes every 5 min for a total of 75 min. We indeed see that CTP-DPLC reduces the transmission overhead of the original CTP due to smaller header overhead. We also see that CTP-DPLC reduces the transmission overhead of CTP-max which looks quite unstable over time. Figure 9d shows the average transmission overhead in 5 min for a duration of 75 min. We see that CTP-DPLC reduces 13% transmission overhead compared to CTP, and reduces 21% transmission overhead compared to CTP-max.

Energy Efficiency. In order to see the energy efficiency of different schemes, we test each scheme with default LPL in TinyOS. Figure 9e shows the duty cycle every 5 min for a total duration of 75 min. We see that DPLC leads to the lowest duty cycle. The reason why we improve the energy efficiency of CTP is easy to understand: first, we reduce the transmission overhead; second, we also reduce the MAC delay because of packet aggregation. CTP-max has a higher duty cycle than CTP-DPLC for most of the time because of more retransmissions. Figure 9f shows the average duty cycle. We can see that CTP-DPLC reduces 41.8% duty cycle compared to CTP, and reduces 15.1% duty cycle compared to CTP-max.

VI. Related Work

Packet length optimization has been studied extensively in the literature. Spragins et al. discuss optimizing packet length for a variety of factors, e.g., error in the channel, buffer size, and ARQ schemes [21]. Siew et al. discuss optimal packet length under the Raleigh fading channel model [7]. Recently, WSNs have attracted a great deal of research attentions [22]–[24]. Sankarasubramaniam et al. extend prior work into WSNs, where an optimal packet length framework based on energy efficiency is proposed [10]. Recently, Vuran et al. propose a cross-layer packet length optimization framework in which effects of multi-hop routing and the broadcast nature are captured [11]. The major difference of our work compared to the abovementioned work is that we allow dynamic adaptation while all the above work derives a fixed optimal packet length which is not suitable for dynamic, time-varying WSNs.

Dynamic packet length adaptation has been investigated in 802.11-based wireless systems. Lettieri et al. study the effect of variable packet length on several metrics, and implement a dynamic adaptation scheme on custom Linux OS [6]. Jelenković et al. propose a dynamic fragmentation algorithm that adaptively matches channel failure characteristics [9]. The work of [6] uses a simple independent bit error model for packet length adaptation. The optimal packet length is calculated based on an estimated BER which is hard to estimate accurately in practice. The work of [9] requires the sender to

measure the channel availability period for adaptation. This method suffers from hidden terminal problems. Our work differs from the above work in that we employ a lightweight and accurate link estimation method that is essential to a packet adaptation scheme.

Link estimation in WSNs has been extensively investigated in the literature [17], [19], [25]. We incorporate the recent technique proposed in [19] to passively monitor packet receptions. The work of [19] requires L2 ACKs. We further extend the work in [19] by proposing the AggAck mechanism which mitigates ACK overhead, and redundant data retransmissions in asymmetric links.

Adaptive packet aggregation has been studied in [26], in which an application-independent L2.5 framework is proposed to maximize channel utilization. Our work differs from [26] in three major ways. First, the work of [26] adapts the degree of aggregation by monitoring MAC delays, which are highly dependent on the MAC layer. For example, it may not work correctly with TDMA or LPL in which MAC delays do not reflect real channel conditions. Second, the work of [26] only deals with packet aggregation. In contrast, our work proposes a unified framework that integrates both aggregation and fragmentation. Third, we conduct real testbed experiments running the CTP protocol, which is missing in [26].

VII. Conclusion

This paper presents DPLC, a dynamic packet length control scheme for WSNs. We introduce a lightweight and accurate link estimation method that captures both physical channel conditions and interferences. Moreover, we provide two easy-to-use services, i.e., small message aggregation and large message fragmentation, to facilitate upper-layer application programming. We implement DPLC based on TinyOS 2.1. Real indoor testbed experiments running the CTP protocol show that DPLC results in a 13% reduction in transmission overhead and a 41.8% reduction in energy consumption compared to the original protocol, and a 21% reduction in transmission overhead and a 15.1% reduction in energy consumption compared to a simple aggregation scheme. We are now examining the effectiveness of this scheme in our ongoing project, GreenOrbs [12].

Acknowledgments

This work is supported by the National Basic Research Program of China (973 Program) under grant No. 2006CB303000, and in part by NSERC Discovery Grant 341823-07, NSERC Strategic Grant STPGP 364910-08 and FQRNT Grant 2010-NC-131844.

References

[1] H. Dubois-Ferrière, D. Estrin, and M. Vetterli, "Packet Combining in Sensor Networks," in *Proceedings of ACM SenSys*, 2005.

[2] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-Layer Wireless Bit Rate Adaptation," in *Proceedings of ACM SIGCOMM*, 2009.

[3] J. W. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," in *Proceedings of ACM SenSys*, 2004.

[4] M. Yang and Y. Yang, "A Hypergraph Approach to Linear Network Coding in Multicast Networks," *IEEE Transactions on Parallel and Distributed Systems*, 2009, to appear.

[5] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Data Gathering with Tunable Compression in Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 2, pp. 276–287, 2009.

[6] P. Lettieri and M. B. Srivastava, "Adaptive Frame Length Control for Improving Wireless Link Throughput, Range, and Energy Efficiency," in *Proceedings of IEEE INFOCOM*, 1998.

[7] C. K. Siew and D. J. Goodman, "Packet Data Transmission Over Mobile Radio Channels," *IEEE Transactions on Vehicular Technology*, vol. 38, no. 2, pp. 95–101, 1989.

[8] E. Modiano, "An adaptive algorithm for optimizing the packet size used in wireless ARQ protocols," *Wireless Networks*, vol. 5, no. 4, pp. 279–286, 1999.

[9] P. R. Jelenković and J. Tan, "Dynamic Packet Fragmentation for Wireless Channels with Failures," in *Proceedings of ACM MobiHoc*, 2008.

[10] Y. Sankarasubramaniam, I. F. Akyildiz, and S. W. Mclaughlin, "Energy Efficiency based Packet Size Optimization in Wireless Sensor Networks," in *Proceedings of IEEE Internal Workshop on Sensor Network Protocols and Applications*, 2003.

[11] M. C. Vuran and I. F. Akyildiz, "Cross-layer Packet Size Optimization for Wireless Terrestrial, Underwater, and Underground Sensor Networks," in *Proceedings of IEEE INFOCOM*, 2008.

[12] GreenOrbs, "Available: http://www.greenorbs.org."

[13] L. Mo, Y. He, Y. Liu, J. Zhao, S. Tang, X.-Y. Li, and G. Dai, "Canopy Closure Estimates with GreenOrbs: Sustainable Sensing in the Forest," in *Proceedings of ACM SenSys*, 2009.

[14] TinyOS, "Available: http://www.tinyos.net."

[15] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *SenSys*, 2003.

[16] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proceedings of ACM SenSys*, 2009.

[17] K. Srinivasan and P. Levis, "RSSI is Under Appreciated," in *Proceedings of EmNets*, 2006.

[18] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica, "Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks," in *Proceedings of ACM SenSys*, 2007.

[19] G. Hackmann, O. Chipara, and C. Lu, "Robust Topology Control for Indoor Wireless Sensor Networks," in *Proceedings of ACM SenSys*, 2008.

[20] L. Sang, A. Arora, and H. Zhang, "On Exploiting Asymmetric Wireless Links via One-way Estimation," in *Proceedings of ACM MobiHoc*, 2007.

[21] J. D. Spragins, J. L. Hammond, and K. Pawlikowski, *Telecommunications: Protocols and Design*. Addison Wesley Publishing Company, 1991.

[22] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.

[23] X. Liu, Q. Wang, W. He, M. Caccamo, and L. Sha, "Optimal real-time sampling rate assignment for wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 2, pp. 263–295, 2006.

[24] S. Ganeriwal, I. Tsigkogiannis, H. Shim, V. Tsiatsis, M. B. Srivastava, and D. Ganesan, "Estimating clock uncertainty for efficient duty-cycling in sensor networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 843–856, 2009.

[25] H. Zhang, A. Arora, and P. Sinha, "Learn on the Fly: Data-driven Link Estimation and Routing in Sensor Network Backbones," in *Proceedings of IEEE INFOCOM*, 2006.

[26] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "AIDA: Adaptive Application-Independent Data Aggregation in Wireless Sensor Networks," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 426–457, 2004.