Dynamic Packet Length Control in Wireless Sensor Networks

Wei Dong, *Member, IEEE,* Chun Chen, *Member, IEEE,* Xue Liu, *Member, IEEE,* Yuan He, *Member, IEEE,* Yunhao Liu, *Senior Member, IEEE,* Jiajun Bu, *Member, IEEE,* and Xianghua Xu, *Member, IEEE*

***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. 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 achieves the best performance compared with previous works.**

***Index Terms*—Packet length optimization, link estimation, aggregation, fragmentation, wireless sensor networks.**

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].

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 the header overhead by making packet large, and the need to reduce packet error rates (PER) in the noisy channel by using small packet length [2].

Although there have been several studies on packet length optimizations in the literature [2]–[6], existing approaches usually require that a set of parameters to be carefully tuned such that it can better match the level of dynamics seen by any particular data trace. However, any fixed set of parameters will not adapt to the changing conditions since one parameter set does not fit all conditions. Furthermore, the update process would require user intervention, further data collection and reprogramming the parameters. This is precisely what we want to avoid in our case, and one of the strengths of using dynamic packet length optimization scheme.

We design and implement DPLC based on TinyOS 2.1. The current implementation of DPLC on TelosB motes is lightweight. We evaluate DPLC in a testbed consisting of 20 TelosB nodes, running the CTP protocol [7], and compare its performance with a simple aggregation scheme and AIDA [8]. Results show that DPLC achieves the best performance in terms of transmission overhead and energy efficiency.

The contributions of our work are highlighted as follows.

- We design and implement a *dynamic* packet length optimization scheme in the context of WSNs. We incorporate an *accurate* link estimation method that captures wireless characteristics.
- We 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.
- We evaluate DPLC extensively. We demonstrate the feasibility of dynamic packet length optimization in WSNs, and show its performance improvement by integrating it into CTP [7], a widely used data collection protocol.

Compared with our conference paper [9], this journal version has the following extensions. (1) We provide a complete description on how to derive the metric in Section IV-B. (2) We analyze DPLC's energy consumption and convergence in Section IV. (3) We examine the accuracy of our link estimation method. We present more detailed comparison results in Section VIII.

The rest of this paper is structured as follows. Section II discusses related work. Section III describes the experimental observations that motivate our design. Section IV presents the design of DPLC. Section V presents an analysis the energy consumption and the convergence rate of DPLC. Section VI introduces the implementation details. Section VII shows the simulation results. Section VIII shows the evaluation results. Finally, Section IX concludes this paper.

Manuscript received August 1, 2012; revised February 6, August 4, and December 15, 2013; accepted December 15, 2013. The associate editor coordinating the review of this paper and approving it for publication was B. Liang.

A preliminary version of this work was published in IEEE INFOCOM 2010.

This work is supported by the National Science Foundation of China Grants No. 61202402, 61070155, 61170213, 61373181, 61370087, the National Key Technology R&D Program (2012BAI34B01), the Research Fund for the Doctoral Program of Higher Education of China (20120101120179), the Open Research Fund of Zhejiang Provincial Key Lab of Data Storage and Transmission Technology, Hangzhou Dianzi University (No. 201303), the Fundamental Research Funds for the Central Universities (2012QNA5007), and Demonstration of Digital Medical Service and Technology in Destined Region.

W. Dong, C. Chen, and J. Bu are with the Zhejiang Key Lab of Service Robot, College of Computer Science, Zhejiang University, China (e-mail: {dongw, chenc, bjj}@zju.edu.cn).

X. Liu is with the School of Computer Science, McGill University (e-mail: xueliu@cs.mcgill.ca).

Y. He and Y. Liu are with the School of Software and Tsinghua National Lab for Information Science and Technology (TNLIST), Tsinghua University, China (e-mail: {he, yunhao}@greenorbs.com).

X. Xu is with the Zhejiang Provincial Key Lab of Data Storage and Transmission Technology, School of Computer Science, Hangzhou Dianzi University, China (e-mail: xhxu@hdu.edu.cn).

Digital Object Identifier 10.1109/TWC.2014.012414.121106

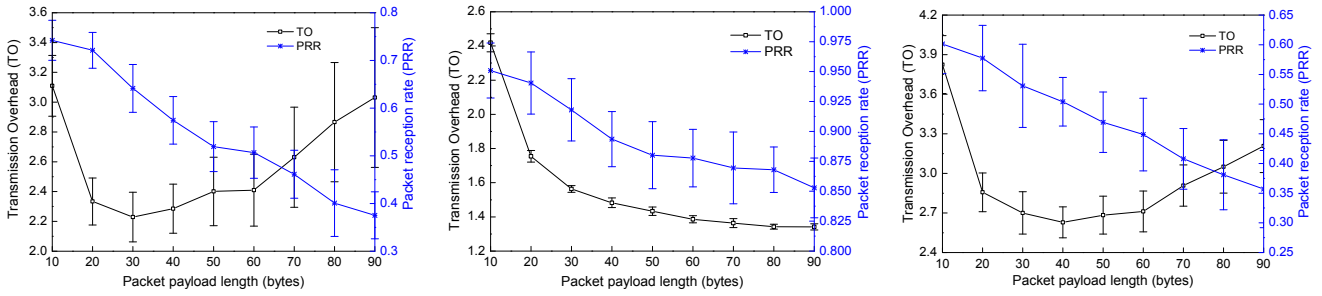


Fig. 1: Impact of packet payload length on normalized transmission overhead (TO) and packet reception rate (PRR). (a) Experiment 1: power level=3, distance=8m. (b) Experiment 2: power level=4, distance=8m. (c) Experiment 3: power level=4, distance=35m.

II. 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 [10]. Siew et al. discuss optimal packet length under the Rayleigh fading channel model [3]. Recently, WSNs have attracted a great deal of research attention [11]–[13]. Sankarasubramaniam et al. extend prior work into WSNs, where an optimal packet length framework based on energy efficiency is proposed [6]. The above works usually require that a set of parameters to be carefully tuned such that it can better match the level of dynamics seen by any particular data trace. However, any fixed set of parameters will not adapt to the changing conditions. DPLC can automatically adapt to network dynamics.

Dynamic packet length adaptation has been investigated in 802.11-based wireless systems [2], [5], [14], [15]. Lettieri et al. study the effect of variable packet length on several metrics, and implement a dynamic adaptation scheme on custom Linux OS [2]. Jelenković et al. propose a dynamic fragmentation algorithm that adaptively matches channel failure characteristics [5]. The work of [2] uses a simple independent bit error model for packet length adaptation. The work of [5] requires the sender to measure the channel availability period for adaptation. 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 [16]–[18]. We incorporate the recent technique proposed in [18] to passively monitor packet receptions. The work of [18] requires L2 ACKs. We further extend the work in [18] by proposing the Aggregated ACK (AggAck) mechanism which mitigates ACK overhead, and redundant data retransmissions in asymmetric links.

Jamieson et al. propose PPR [19] to exploit correct bits in partial packets to improve the performance of wireless communications. However, obtaining such information needs PHY-layer modifications. Such requirements make it hard to generalize PPR to many hardware platforms, such as TelosB and Mica nodes. In DPLC, we use the hardware-independent packet reception ratio (PRR) to measure the link quality.

Adaptive packet aggregation has been studied in [8], in which an application-independent L2.5 framework is proposed to maximize channel utilization. Our work differs from [8] in two major ways. First, the work of [8] adapts the degree of aggregation (DoA) by monitoring MAC delays, which are highly dependent on the MAC layer. Second, the work of [8] only deals with packet aggregation. In contrast, our work proposes a unified framework that integrates both aggregation and fragmentation.

III. Motivation

In this section, we perform experiments to motivate dynamic packet length adaptation.

To see the impact of variable packet length on the system performance in WSNs, we conduct three experiments with two TelosB motes, communicating in a quiet indoor environment. In experiment 1, the transmit power level is 3 and the communication distance is 8m. In experiment 2, the transmit power level is 4 and the communication distance is 8m. In experiment 3, the transmit power level is 4 and the communication distance is 35m. The packet transmission interval is 128ms. We measure the packet reception rate (PRR) and the normalized transmission overhead per useful received byte (denoted as TO) for each packet payload length. A low TO value indicates a high goodput and a high energy efficiency provided that the transmission time and the transmission energy consumption are approximately proportional to the packet length. We ran each experiment 30 times, and Figure 1 shows the mean value and the standard deviation of PRR and TO. We can see that with the packet payload length increases from 10 bytes to 90 bytes, the observed PRR decreases. The packet lengths that minimize the TO value are different for different experiment settings.

While the above experiments show that packet length and PRR show correlation when the transmit power level is relatively low with respect to the communication distance, the correlation also exists when there is heavy interference. For example, in [20], the authors design a scheme called WISE to maximize throughput efficiency of ZigBee in presence of WiFi traffic.

We see from the above experiment results that *a fixed set of parameters (e.g., using a fixed packet length) will not adapt to the changing conditions and dynamic packet length adaptation is beneficial in WSNs*.

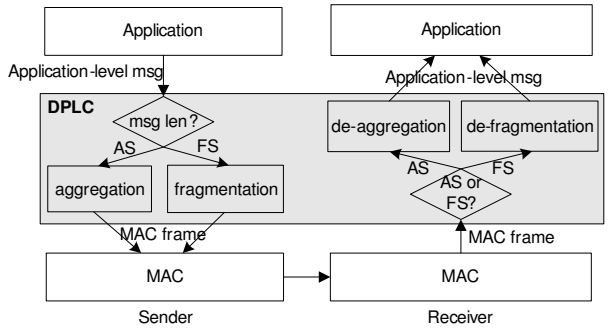


Fig. 2: DPLC overview.

IV. Design

In this section, we present DPLC's design. 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 incorporate an accurate link estimation method that can capture physical channel conditions.
- 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 this section. The remaining two design goals, i.e., ease of programming and lightweight for implementation, will be elaborated in Section VI-A and Section VI-B respectively.

A. Overview

Figure 2 shows an overview of DPLC. 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 appropriate 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), DPLC transmits it out 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 been received or the receive buffer is full in FS), the DPLC module at the receiver notifies the upper layer for further handling.

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 [7]. 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 denote AS_0 with AggAck afterwards).

(ii) FS is useful for bulk data transmission, e.g., Flush [21]. 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 denote FS with AggAck afterwards).

B. Metrics for Dynamic Adaptation

We use the metric of transmission overhead per useful byte (TO) as the metric. It is defined as the number of overall transmitted bytes divided by the number of received useful bytes. Generally speaking, as the transmission time and the energy consumption are approximately proportional to the number of transmitted bytes, a low TO value indicates a high goodput and a high energy efficiency. The goal of our dynamic adaptation scheme is hence to minimize the TO value.

For a path $1 \rightarrow k+1$, the normalized path transmission overhead $TO_{1\rightarrow k+1}$ is the number of total transmitted bytes at nodes $1, \ldots, k$ divided by the number of received useful bytes at node $k+1$.

We use $dr_{i\rightarrow i+1}$ to denote the data delivery ratio over the link $i \rightarrow i+1$. The data delivery ratio differs from link PRR in that link-layer retransmissions can improve the data delivery ratio. It relates to PRR and the link-layer retransmission threshold m as follows,

$$dr_{i\rightarrow i+1} = 1 - (1 - p_{i\rightarrow i+1})^{m+1} \tag{1}$$

When the threshold of retransmissions is 0 (i.e., there is exactly one transmission), dr equals to p, i.e., the packet delivery ratio equals to the link PRR. Note that both dr and p are functions of the packet length. We omit it here for simplicity of notation.

Metric for Single-hop Transmission. For a flow traversing a link $i \rightarrow i+1$, we decide the transmitted packet length at node i in order to minimize the single hop metric which is,

$$TO_{i\rightarrow i+1}(l) = \frac{l + H + O}{l \cdot p(l)} \tag{2}$$

where l is the packet payload length (bytes) over the link, $p(l)$ is the PRR from i to $i+1$ given the packet payload length l, H is MAC header overhead, and O is the additional header overhead introduced by DPLC.

Metric for Multi-hop Transmission. For a flow traversing a path $1 \rightarrow k+1$, we decide the transmitted packet length at node k as follows.

The calculation is different because in this case node k is not the source node (the source node is $k-1$ hops away from node k). The normalized transmission overhead $TO_{1\rightarrow k+1}$ is

the sum of the transmission overhead over the link $k \to k+1$ and the transmission overhead over the path $1 \to k$ for node $k+1$ to receive 1 useful byte.

For receiving one useful byte at node $k+1$, the transmission overhead over the link $k \to k+1$ is $TO_{k \to k+1}$ according to the definition. For receiving one useful byte at node $k+1$, node k must receive $\frac{1}{dr_{k \to k+1}}$ bytes.

For receiving $\frac{1}{dr_{k \to k+1}}$ useful bytes at node k, the transmission overhead over the path $1 \to k$ is $\frac{1}{dr_{k \to k+1}} \cdot TO_{1 \to k}$. So the normalized transmission overhead over the entire path $1 \to k+1$ can be recursively calculated as follows,

$$TO_{1 \to k+1} = TO_{k \to k+1} + \frac{1}{dr_{k \to k+1}} \cdot TO_{1 \to k} \qquad (3)$$

The multi-hop metric can be explicitly expressed as a function of l as follows,

$$TO_{1 \to k+1}(l) = TO_{k \to k+1}(l) + \frac{1}{dr_{k \to k+1}(l)} \cdot TO_{1 \to k} \quad (4)$$

where l is the packet length over the link $k \to k+1$.

Distributed computation of the metric. For a single hop transmission over the link $i \to i+1$, node i needs to monitor the link PRR, $p(l)$ and decide the packet length that minimize the metric.

For a multi-hop path transmission over the path $1 \to k+1$, in order to decide the packet length over the link $k \to k+1$, node k needs to monitor (i) the link PRR, $p(l)$ (ii) the link delivery ratio, $dr(l)$ and (iii) the transmission overhead of the previous $k-1$ nodes, $TO_{1 \to k}$. The link PRR, $p(l)$ is monitored by node k, and $dr(l)$ can be derived from $p(l)$ and the link-layer retransmission threshold m. $TO_{1 \to k}$ is piggybacked in each data packet and is passed down from node $k-1$, i.e., the calculation of our metric at the current hop requires information at previous hops. It is worth noting that path changes will not affect the calculation since it only affects the *future* of packet transmission rather than the *history* of packet. The information at previous hops is piggybacked in a packet for the calculation of metric at the current hop.

It can be seen from Eq. (4) that when $dr_{k \to k+1}(l) = 1$, e.g., the link is perfect or the retransmission threshold is high enough, minimizing the path transmission overhead $TO_{1 \to k+1}(l)$ is equivalent to maximizing the link transmission overhead $TO_{k \to k+1}(l)$.

C. Description of DPLC

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 full, DPLC computes the metric and tries to increase (or decrease) the packet length by the granularity.

We use a gradient variable (g) to decide whether to increase the packet length or decrease the packet length. DPLC uses a bit vector to record each packet's receptions and use it to calculate the packet reception ratio. Initially, the gradient variable is set to 1 and DPLC stays in the `INIT` state. When the window is full, DPLC enters the `STEADY` state and computes the metric for the current packet length. Then DPLC enters the `TRY` state, increasing or decreasing the packet length. 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 transitions into the `STEADY` state. In this case, the gradient variable is unchanged. If the performance decreases, it restores the original packet length and reenter the `INIT` state. In this case, the gradient variable is inverted. DPLC also inverts the gradient when the packet length has already reached the minimum or maximum packet length.

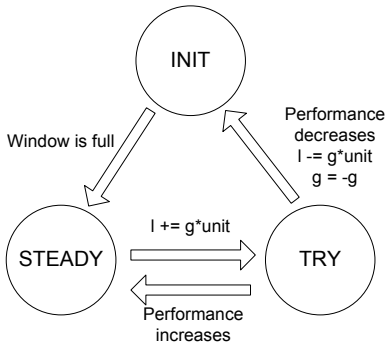


Fig. 3: DPLC state transition diagram. g denotes the gradient variable and unit denotes the granularity for adaptation.

D. The Aggregated ACK (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 ($n \geq 1$), 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 [22], this synchronous mechanism improves the ACK reliability significantly. The AggAck carries only one byte information, i.e., the number of received packets 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 current window have been received. To avoid sending a separate ACK request packet, we let the receiver 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 [23]. 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

Table 1: Notations

Notation	Meaning
n	The current node
C_n	The set of node n's child nodes; $c \in C_n$ denotes one child.
m	The retransmission threshold.
T_{on} (11ms)	Channel polling time in XMAC [25].
T_{off} (500ms)	Sleep interval
T_b	Time to transmit one byte
len	Packet length at the link from node n to its parent
len_c	Packet length at the link from node c to node n
$PRR(len)$	Packet reception ratio at the link from node n to its parent with packet payload length of len.
$PRR_c(len_c)$	Packet reception ratio at the link from node c to node n with packet payload length of len_c.
T_{txn}	Average time spent in transmission mode when transmitting a packet; $T_{txn} = T_b \cdot (H + O + len)$ where H is the header overhead and O is the overhead if DPLC is used.
T_{txc}	Average time spent in transmission mode when receiving a packet (e.g., replying an ACK);
T_{rxn}	Average time spent in receiving mode when transmitting a packet.
T_{rxc}	Average time spent in receiving mode when receiving a packet; $T_{txn} = T_b \cdot (H + O + len)$ where H is the header overhead and O is the overhead if DPLC is used.
U_n	Useful data rate from node n (bytes/s).
U_{txc}	Useful data from node c including all its downstream nodes (bytes/s).
F_{txn}	The rate at which node n transmits packets (# of packets/s).
F_{rxc}	The rate at which node n receives packets (# of packets/s).

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.

V. Analysis and Extensions

Energy analysis and extensions. We can extend the energy analysis approach in [24] by considering the packet length under XMAC [25]. We use the same notations as [24] for better understanding. Table 1 lists some parameters for our analysis.

We analyze energy efficiency of node n in terms of the duty cycle, i.e., the fraction of time the radio spent in transmission mode (D_{txn}) and receiving mode (D_{rxn}). The duty cycle can be calculated as $D_{txn} + D_{rxn}$.

According to [24], node n's fraction of time spent in transmission mode can be computed as:

$$D_{txn} = F_{txn}T_{txn} + \sum_{c \in C_n} F_{rxc}T_{txc} \tag{5}$$

where F_{txn} and F_{rxc} are the rates at which node n transmits packets and receives packets respectively.

Node n's fraction of time spent in receiving mode due to actual communication is [24],

$$D_{rxc} = F_{txn}T_{rxn} + \sum_{c \in C_n} F_{rxc}T_{rxc} \tag{6}$$

Node n's fraction of time spent in receiving mode is [24],

$$D_{rxn} = D_{rxc} + (1 - D_{rxc})F_{cc} \tag{7}$$

where $F_{cc} = T_{on}/(T_{on} + T_{off})$.

The key difference is that we shall consider the impact of packet length in computing F_{txn} and F_{rxc}. Both variables depend on the packet length.

$$F_{txn}(len) = (N_{rtx}(len) + 1) \cdot (U_n + \sum_{c \in C_n} U_{txc}R_c(len_c))/len \tag{8}$$

where $N_{rtx}(len)$ is the expected number of retransmissions at the link from node n to its parent with packet payload length len, i.e., $N_{rtx}(len) = \min(1/PRR(len) - 1, m)$; R_c is reliability of link from child c to node n considering the retransmission threshold m, i.e., $R_c(len_c) = 1 - (1 - PRR_c(len_c))^{m+1}$; The term $(U_n + \sum_{c \in C_n} U_{txc}R_c(len_c))/len$ calculates the number of transmitted packets at the packet length of len.

$$F_{rxc}(len_c) = (N_{rtx,c}(len_c) + 1) \cdot U_{txc}/len_c \cdot P_{stc} \tag{9}$$

where $N_{rtx,c}(len_c)$ is the expected number of retransmissions at the link from node c to node n with packet payload length len_c, i.e., $N_{rtx,c}(len_c) = \min(1/PRR_c(len_c) - 1, m)$; P_{stc} is the probability of receiving at least one strobe, according to [24], $P_{stc} = 1 - (1 - PRR_{stc})^{(T_{on} - T_{st})/T_{it}}$ where T_{on}, T_{st} and T_{it} are all MAC specific parameters. It is worth noting that PRR_{stc} is independent on the packet length since the length of strobe packet is fixed in XMAC [25].

The metric to minimize energy consumption should minimize $(D_{txn} + D_{rtx})(len)$ which can be expressed as a function of len as follows (derived from Eq. 8 and Eq. 9):

$$\begin{aligned}(D_{txn} + D_{rtx})(len) &= F_{txn} \cdot T_{txn} + (1 - F_{cc}) \cdot F_{txn} \cdot T_{rxn} + C \\ &= U \cdot T_b \cdot \frac{H + O + len}{PRR(len) \cdot len} + \frac{U \cdot (1 - F_{cc}) \cdot T_{rxn}}{PRR(len) \cdot len} + C\end{aligned} \tag{10}$$

where F_{cc}, C and U are all independent on len. $U = U_n + \sum_{c \in C_n} U_{txc}R_c(len_c)$.

We can extend DPLC to use the new metric for minimizing the energy consumption. However, MAC specific parameters (such as F_{cc}, T_{rxn}) need to be known in advance.

Convergence analysis and extensions. DPLC requires some amount of data traffic to converge with respect to channel conditions. Consider a link A→B with link bit error rate (BER) b_1 during $[t_1, t)$ and link BER b_2 during $[t, t_2]$. Let w denotes the sliding window size and δ denotes the granularity for packet length adaptation in DPLC. Assume the optimal packet length under b_1 is $l_1 = k_1\delta$ and the optimal packet length under b_2 is $l_2 = k_2\delta$.

We can get the following results:

- If $l_1 < l_2$, DPLC requires $\sum_{k=k_1}^{k_2} k\delta w$ bytes of data traffic to converge.
- If $l_1 > l_2$, DPLC requires $\sum_{k=k_2}^{k_1} k\delta w$ bytes of data traffic to converge.

Proof: Without loss of generality, we consider $l_1 < l_2$. According to the operation of DPLC (see Section IV-C), it stays in the INIT state with packet length $l_1 = k_1\delta$ until w packets are transmitted, translating to a total of $k_1\delta \cdot w$ bytes traffic.

Then, DPLC tries to increase the packet length by δ step-by-step until the length reaches l_2. There are $k_2 - k_1$ steps. In each step, DPLC transmits w packets with packet length

```
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);
  command void setTimeout(uint32_t ms);
}
```

(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.

$l = k\delta$ ($k_1 < k \leq k_2$), translating to $k\delta \cdot w$ bytes traffic in each step.

Therefore, the total data traffic in the above convergence process equals to $\sum_{k=k_1}^{k_2} k\delta w$. ■

This result implies two facts. (1) The higher the traffic rate, the smaller the convergence time. (2) The smaller the change in link quality, the smaller the convergence time.

DPLC's convergence time can be improved. For example, Error Estimating Code [26] can be employed to estimate the link bit error rate (BER). With this information, DPLC can predict the link PRR at different packet lengths. Hence, the optimal packet length can be directly derived by predicting PRR as $(1 - BER)^{H+O+l}$ without the iteration process described in Section IV-C. However, additional header overhead and computational overhead are required for BER estimation.

VI. Implementation

A. Programming Interface

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

Using dynamic packets introduces several challenges in the protocol design. In the following paragraphs, we describe the drawbacks introduced by dynamic packet adaptation and the corresponding approaches incorporated by DPLC to alleviate the negative impacts.

(1) The increase of message buffer sizes. As DPLC needs to send and receive variable sized packets, the MAC layer needs to reserve one sending buffer and one receiving buffer at the maximum size supported by the radio (e.g., 128 bytes for CC2420). For this purpose, we use a different structure type called max_message_t for defining the sending and receiving buffers so that arbitrary sized packets can be sent or received. We do not use the default structure type message_t and change the macro TOSH_DATA_LENGTH directly because it will unnecessarily increase the size of every message_t defined in the application logic. The increase is $(128 - 36) \times 2 = 184$ bytes because we enlarge two buffers from 36 bytes (default) to 128 bytes.

(2) Different type for message passing (between different layers). The main difference between DPLC and the TinyOS AM interface is that we use void* instead of message_t* to point to a message structure. Such a difference stems from the fact that the application needs to pass to the DPLC layer a message buffer for sending and the message buffer can be larger than the size of message_t (e.g., for FS). This overhead depends on how many times the application program invokes the send/receive operations.

(3) Timing. For AS, sending delays and receiving delays may be introduced because a sufficient number of packets need to be aggregated at the MAC layer before actually sent out. To prevent indefinite delays, we provide another command in the DMSend interface, i.e., setTimeout(uint32_t ms). With this command, application programmers can specify a maximum delay that the application can tolerate before the message is actually sent out. A message whose in-queue delay reaches the specified timeout value will be sent out without further aggregation.

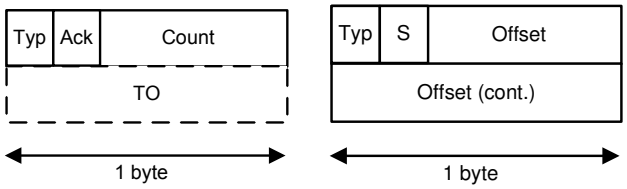
Generally speaking, application programming using DMSend is not much different from that using AMSend. DPLC handles all specific details about aggregation/deaggregation, fragmentation/defragmentation such that upper-layer applications need not care about how to use the appropriate packet length in the MAC layer. Without such a scheme, the message length will need to be manually changed, 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 Section IV-C) (1) The extra overhead of DMSend.send is only about 30 μs before calling AMSend.send. (2) The extra overhead of DMReceive.receive is about 170 μs due to deaggregation or defragmentation. (3) The extra overhead of the decision making algorithm is at most 480 μs (at sendDone or AggAck is received). The frame transmission time in TinyOS consists of the following components: (i) copying the message from memory to CC2420 FIFO buffer which takes about 3–4ms. (ii) expected MAC backoff time which takes about 5ms because the maximum initial backoff time in TinyOS is about 10ms (iii) message transmission time which takes about 1.6ms (for a 40-byte packet). Our algorithm adds an additional 0.48ms, which increases the frame transmission time by 0.48/(3+5+1.6)=5%.

Memory Overhead. DPLC incurs memory overhead on RAM (data) and ROM (program). (1) DPLC needs about 256



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

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

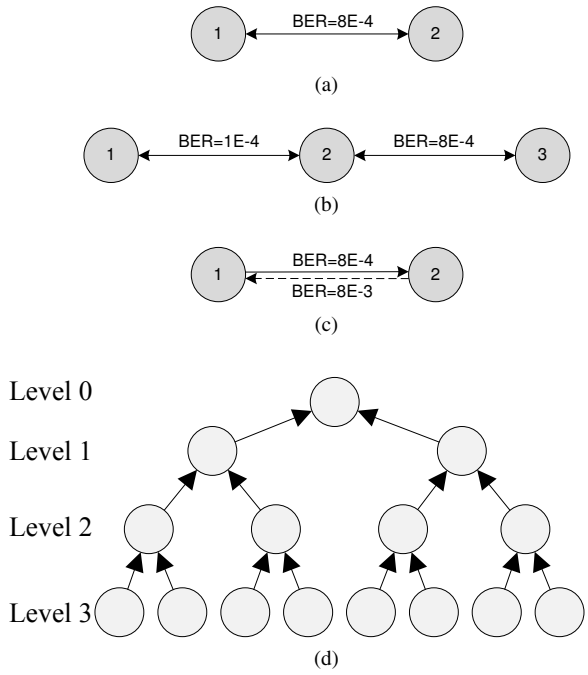


Fig. 6: Topologies for validating DPLC's design. (a): single hop (symmetric link with median link quality). (b): two hops (symmetric links in which the quality of the first link is high and the quality of the second link is median). (c): single hop (asymmetric link in which the reverse link quality is low). (d): a tree topology of 4 levels.

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. (2) 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 VI-A. We have found that the original benchmark consumes 11,528 bytes ROM while the modified version consumes 16,052 bytes ROM. This indicates the DPLC module consumes approximately 4.5 KB ROM, which is acceptable compared to 48 KB ROM in TelosB.

Header Overhead. We add small header overhead to both data packets (at most 2 bytes) and AggAck packets. Figure 5 shows the data packet overhead in DPLC. Please refer to our conference paper [9] for a detailed analysis.

VII. Simulations

The major goals of the simulations described in this section is to demonstrate DPLC's achievable improvements in terms of the transmission overhead per useful byte (TO).

We modify TOSSIM (version 2) to use the bit error model, i.e., $PRR = (1 - BER)^{L+H+O}$, where $H = 13$ for TinyOS, and $O = 2$ for DPLC. Figure 6 shows the topologies for validation. Basically, we validate DPLC in both single-hop topology and multi-hop topology with varying link qualities (e.g., BER=1×10^{-4} or BER=8×10^{-4}). We run each experiment 5 times in each topology.

For the single hop case, we setup two nodes communicating at time interval of 200 ms. For the multihop case, we setup three nodes in which the first node transfers a single flow of data to the last node at time interval of 200 ms. In this case, the second node serves as a forwarding node. In the tree topology, the level-0 node collects data packets from all other nodes.

We use exhaustive search to find the optimal packet length in order to validate the DPLC algorithm. The exhaustive search procedure simply iterates over each possible packet length. In each iteration, it calculates the TO metric defined in Section IV-B based on the already known simulation parameters. The search procedure finally returns the packet length that minimizes the TO metric. Since 802.15.4 has a maximum packet length of 128 bytes, the search space is small.

DPLC is an iterative algorithm that increases or decreases the packet length based on the TO metric. It will converge to the packet length with which neither an increase nor a decrease will improve the performance with respect to the TO metric. DPLC does not use the exhaustive search procedure since some parameters (e.g., bit error rate, network topology) are difficult to know a priori in practice.

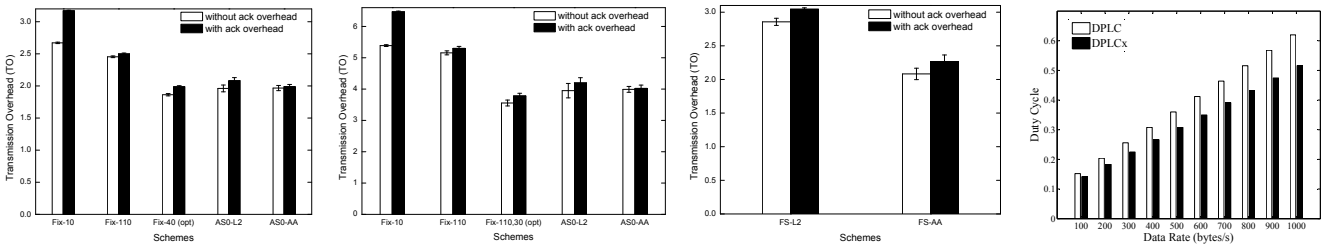
Figures 7a, 7b show the performance of AS_0 in topology (a) and topology (b) respectively. We measure the normalized transmission overhead (TO) of each scheme in two ways: TO without ACK overhead (in which the ACK overhead is not accounted for), and TO with ACK overhead (in which the ACK overhead is also accounted for). We can see that in both cases, our DPLC algorithm achieves satisfactory performance. Compared with the fixed optimal scheme (which always uses the optimal packet length found by the exhaustive search procedure), the additional overhead of DPLC keeps within 10%. We can also see that the AggAck mechanism mitigates the ACK overhead, leading to about 5% improvement.

Figure 7c shows the performance of FS in topology (c). We can see that in asymmetric links, the performance of FS-L2 degrades due to redundant data packet retransmissions. On the other hand, the AggAck mechanism achieves a much better performance because in this case only the data packets that are actually lost need to be retransmitted.

We evaluate the energy efficiency of DPLCx, the extended version of DPLC for X-MAC, in terms of radio duty cycles compared to the original DPLC. Figure 7d shows how the radio duty cycle varies with increasing data rate for X-MAC in the tree topology in Figure 6d. We can see that the radio duty cycle can further be reduced by 6% to 17% if MAC parameters are taken into account.

VIII. Testbed Experiments

We implement DPLC based on TinyOS 2.1 and integrate it into CTP [7]. CTP is a data collection protocol that dynamically selects the best route to the sink according to a hybrid link estimation algorithm [7].



(a) AS_0 in topo-(a) (b) AS_0 in topo-(b) (c) FS in topo-(c) (d) DPLC and DPLCx in topo-(d)

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)".

We evaluate DPLC in testbed experiments consisting of 20 TelosB motes running the CTP protocol [7] in order to show that DPLC can be utilized to improve the performance of existing protocols.

Prior fixed optimization solutions heavily depend on the accurate selection of multiple factors such as the distance between nodes, path loss exponent, shadow fading component, and etc. If those factors are accurately measured, fixed optimization framework can outperform dynamic approaches. The fact is that they are difficult to obtain in many real-world deployments. For this reason, we select a dynamic adaptation approach AIDA [8] for comparison.

We first evaluate the accuracy of DPLC's link estimation method in Section VIII-A. Then we conduct a comparative study on the performances of different schemes integrated with CTP in Section VIII-B.

- CTP denotes the original CTP protocol using a default packet length of 23 bytes.
- CTP-max denotes the original CTP protocol with a simple aggregation scheme that always transmits the largest packets.
- CTP-AIDA denotes the CTP protocol with a more intelligent aggregation scheme proposed in [8].
- CTP-DPLC denotes the CTP protocol integrated with DPLC.

We use the `TestNetwork` benchmark in TinyOS. The transmission power is configured to 3 and the retransmission threshold is 4. Each `TestNetwork` node sends a packet periodically to the sink at an interval of 20 seconds.

In CTP, the topology is dynamically formed. A node sends packets to its parent which is selected according to the path-ETX metric estimated by existing link estimators. We looked into the trace, and find the hops of the network in the `tth` field of each packet. The network has 3-hop nodes. The topology, however, is time-varying. For most of the time, node 20 delivers packet to node 1 (sink) via node 8 (i.e., two hops).

We test each scheme for at least 2 hours (we do not analyze data in the first 30min warmup time) with and without the use of lower power listening (LPL, in which the sleep interval is set to 500ms). We collect more than 24,000 packets for each of the experiments. After a total of 12 hours, we analyze the following performance metrics.

- Reliability of data collection in terms of packet delivery ratio.

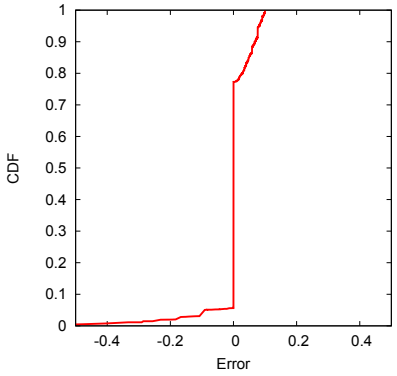


Fig. 8: Accuracy of link estimation.

- Transmission overhead in terms of the number of total transmitted bytes.
- Energy efficiency in terms of the radio duty cycle.

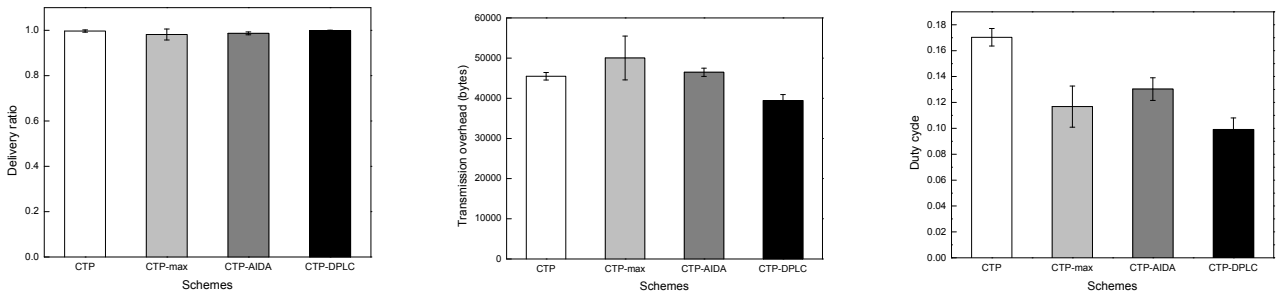
A. Accuracy of link estimation

We evaluate the accuracy of the link estimation method in terms of absolute error. Each node transmits 200 packets in turn. All other nodes record each packet's reception. The estimated link qualities are compared against the ground truth values. Figure 8 shows the CDF of absolute error (i.e., estimated value - real value). We can see that the errors keep within 10% for 95% links, indicating that the link estimation method yields accurate results.

B. Performance comparison

Reliability. Figure 9a shows the collection reliability in term of data delivery ratio. The data delivery ratio is the number of packets received at the sink node divided by the number of generated packets. Note that we use a maximum link-layer retransmission threshold of 4 in this experiment. This means if a packet transmission fails, the sender would retry at most 4 times before it gives up. With link-layer retransmission, the packet delivery ratio keeps high, e.g., above 95%. We can see that the CTP-max scheme is less stable than the other schemes. The reason is due to the fact that larger packets are more suspectable to wireless loss. We can see that CTP-max slightly reduces the reliability while CTP-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 TO value is equivalent to



(a) Delivery ratio (b) Transmission overhead (c) Duty cycle

Fig. 9: Performance comparisons.

optimizing the transmission overhead. We would like to see how CTP-DPLC reduces the transmission overhead. Figure 9b shows the transmission overhead of 19 nodes except the sink node. We indeed see that CTP-DPLC reduces the transmission overhead of the original CTP. This is because CTP-DPLC aggregates application-level messages to larger MAC-level packets. Hence CTP-DPLC's relative header overhead compared to the payload (percentage) is smaller than that of the original CTP. We also see that CTP-DPLC reduces the transmission overhead compared to both CTP-max and CTP-AIDA. We find that the performance of AIDA is relatively low because our experiment has a low data rate. Hence the MAC level contention is not severe, resulting in less aggressive aggregation in AIDA. In summary, CTP-DPLC results in 13%, 21%, 17.7% reduction in the transmission overhead compared to CTP, CTP-max, CTP-AIDA, respectively.

Energy Efficiency. In order to see the energy efficiency of different schemes, we test each scheme with default LPL in TinyOS. With LPL, the energy consumption mainly depends on the radio-on time because the transmitting mode and receiving mode consume roughly the same energy. For example, the TelosB node consumes 19.5mA in the transmitting mode and 21.8mA in the receiving mode [27]. Figure 9c shows the duty cycles for different schemes. We see that DPLC leads to the lowest duty cycle. We improve the energy efficiency of CTP because of two reasons. 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. CTP-AIDA also has a higher duty cycle than CTP-DPLC because of less aggressive aggregation. In summary, CTP-DPLC results in 41.8%, 15.1%, 31% reduction in terms of duty cycles compared to CTP, CTP-max, CTP-AIDA, respectively. Our current metric achieves the smallest energy consumption as it has the smallest radio-on time.

IX. Conclusion

This paper presents DPLC, a Dynamic Packet Length Control scheme. DPLC incorporates a lightweight and accurate link estimation method that captures physical channel conditions. Moreover, DPLC provides 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. The implementation is lightweight with respect to computation, memory, and header overhead. Our experiments using a real indoor testbed running CTP show that DPLC achieves the best performance compared with previous works.

References

[1] H. Dubois-Ferrière, D. Estrin, and M. Vetterli, "Packet combining in sensor networks," in *Proc. 2005 ACM SenSys.*

[2] P. Lettierri and M. B. Srivastava, "Adaptive frame length control for improving wireless link throughput, range, and energy efficiency," in *Proc. 1998 IEEE INFOCOM.*

[3] C. K. Siew and D. J. Goodman, "Packet data transmission over mobile radio channels," *IEEE Trans. Veh. Technol.*, vol. 38, no. 2, pp. 95–101, 1989.

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

[5] P. R. Jelenković and J. Tan, "Dynamic packet fragmentation for wireless channels with failures," in *Proc. 2008 ACM MobiHoc.*

[6] Y. Sankarasubramaniam, I. F. Akyildiz, and S. W. Mclaughlin, "Energy efficiency based packet size optimization in wireless sensor networks," in *Proc. 2003 IEEE International Workshop Sensor Netw. Protocols Applications.*

[7] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. 2009 ACM SenSys.*

[8] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "AIDA: adaptive application-independent data aggregation in wireless sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 2, pp. 426–457, 2004.

[9] W. Dong, X. Liu, C. Chen, Y. He, G. Chen, Y. Liu, and J. Bu, "DPLC: dynamic packet length control in wireless sensor networks," in *Proc. 2010 IEEE INFOCOM.*

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

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Netw.*, vol. 38, pp. 393–422, 2002.

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

[13] 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 Trans. Netw.*, vol. 17, no. 3, pp. 843–856, 2009.

[14] F. Zheng and J. Nelson, "Adaptive design for the packet length of IEEE 802.11n networks," in *Proc. 2008 IEEE ICC.*

[15] M. N. Krishnan, E. Haghani, and A. Zakhor, "Packet length adaptation in WLANs with hidden nodes and time-varying channels," in *Proc. 2011 IEEE GlobeCom.*

[16] H. Zhang, A. Arora, and P. Sinha, "Learn on the fly: data-driven link estimation and routing in sensor network backbones," in *Proc. 2006 IEEE INFOCOM.*

[17] K. Srinivasan and P. Levis, "RSSI is under appreciated," in *Proc. 2006 EmNets.*

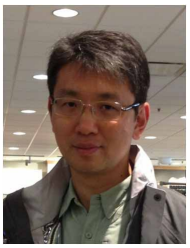
[18] G. Hackmann, O. Chipara, and C. Lu, "Robust topology control for indoor wireless sensor networks," in *Proc. 2008 ACM SenSys.*
[19] K. Jamieson and H. Balakrishnan, "PPR: partial packet recovery for wireless networks," in *Proc. 2007 ACM SIGCOMM.*
[20] J. Huang, G. Xing, G. Zhou, and R. Zhou, "Beyond co-existence: exploiting WiFi white space for ZigBee performance assurance," in *Proc. 2010 IEEE ICNP.*
[21] 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 *Proc. 2007 ACM SenSys.*
[22] L. Sang, A. Arora, and H. Zhang, "On exploiting asymmetric wireless links via one-way estimation," in *Proc. 2007 ACM MobiHoc.*
[23] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. 2004 ACM SenSys.*
[24] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTunes: runtime parameter adaptation for low-power MAC protocols," in *Proc. 2012 ACM/IEEE IPSN.*
[25] M. Buettner, G. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proc. 2006 ACM SenSys.*
[26] B. Chen, Z. Zhou, Y. Zhao, and H. Yu, "Efficient error estimating coding: feasibility and applications," in *Proc. 2010 ACM SIGCOMM.*
[27] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proc. 2005 ACM/IEEE IPSN.*

Wei Dong received his BS and Ph.D. degrees from the College of Computer Science at Zhejiang University in 2005 and 2011, respectively. He is currently an associate professor in the College of Computer Science in Zhejiang University. His research interests include networked embedded systems and wireless sensor networks. He is a member of the IEEE.

Chun Chen received his Bachelor of Mathematics degree from Xiamen University, China, in 1981, and his M.S. and Ph.D. degrees in Computer Science from Zhejiang University, China, in 1984 and 1990 respectively. He is a professor in College of Computer Science, and the Director of Institute of Computer Software at Zhejiang University. His research interests include embedded system, image processing, computer vision, and CAD/CAM.

Xue Liu received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2006. He received his B.S. degree in Mathematics and M.S. degree in Automatic Control both from Tsinghua University, China. He is currently an associate professor in the School of Computer Science at McGill University in Canada. His research interests are in real-time and embedded systems, cyber-physical systems, server and data center performance modeling and management, and software reliability. He worked briefly in the Hewlett-Packard Labs and IBM T. J. Watson Research Center. His work on software reliability received the IEEE Transactions on Industrial Informatics Best Paper Award in 2008. He is a member of both the IEEE and the ACM.

Yuan He received his BE degree in University of Science and Technology of China, his ME degree in Institute of Software, Chinese Academy of Sciences, and his PhD degree in Hong Kong University of Science and Technology. He is now an assistant professor in the School of Software of Tsinghua University and the Vice Director of the IOT-Tech Center, Tsinghua National Lab for Information Science and Technology. His research interests include sensor networks, peer-to-peer computing, and pervasive computing.



Yunhao Liu received the BS degree in automation from Tsinghua University, China, in 1995, the MS and PhD degrees in computer science and engineering from Michigan State University, in 2003 and 2004, respectively. He is now Cheung Kong Professor at Tsinghua University. His research interests include RFID and sensor network, the Internet, and pervasive computing.

Jiajun Bu received the B.S. and Ph.D. degrees in Computer Science from Zhejiang University, China, in 1995 and 2000, respectively. He is a professor in College of Computer Science and the deputy dean of School of Software Technology at Zhejiang University. His research interests include embedded system, mobile multimedia, and data mining. He is a member of the IEEE and the ACM.

Xianghua Xu is now a professor in the School of Computer Science at Hangzhou Dianzi Unviersity, China. He received his B.Eng. in Computer Science from Hangzhou Institute of Electronic Engineering, China, and his Ph.D. degree in Computer Science from Zhejiang University, China. His research interests include wireless networks, parallel and distributed computing. His recent research has been supported by Natural Science Fundation of China. He is the member of the IEEE, ACM, the senior member of CCF (China Computer Federation). He is also the member of CCF Technical Committee of Service Computing and CCF Technical Committee of High Performance Computing. He is a recipient of the "Best Paper Award" of IISWC'2012.