LIPS: **Li**nk **P**rediction as a **S**ervice for Adaptive Data Aggregation in Wireless Sensor Networks

Xiao Ma, Seddik Djouadi, and Qing Cao
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
Email: {xma4, djouadi, cao}@utk.edu

Abstract—**A central component of the design of wireless sensor networks is reliable and efficient transmission of data from source to destination. However, this remains a challenging problem because of the dynamic nature of wireless links, such as interference, diffusion, and path fading. When the link quality worsens, packets will get lost even with retransmissions and acknowledgments when internal queues become full. For example, in a well-known study to monitor volcano behavior [19], the measured data yield of nodes ranges from 20% to 80%. To address this challenge brought by unreliable links, in this paper, we propose the idea of LIPS, or Link Prediction as a Service. Specifically, we argue that it is beneficial for the applications to be designed as adaptive from the start, by taking into account the *future* link quality estimates based on past measurements. In particular, we present a novel state-space based approach for link quality prediction, and demonstrate that it is possible to integrate this model into higher layer data aggregation protocols to improve their performance.**

I. Introduction

One key component of the emerging networked embedded systems, such as wireless sensor networks, is efficient and reliable data collection and aggregation. This task is complicated by various factors, and one most notable factor is the extremely unreliable nature of wireless links through which data are collected, which causes several problems such as congestion and packet loss. Inappropriate selection and use of wireless links will cause tremendous energy cost, shortened system lifetime, and degraded performance.

So far, the basic structure for collection is usually considered as a multi-hop tree topology [8]: each node is connected to the root through multiple hops, forming a tree structure. Routing protocols establish the routing tree based on the wireless link quality to reduce the end-to-end path cost, therefore, decreasing the cost of sending a packet to the root.

However, even though enormous efforts have been invested to choosing the best links to deliver packets, reports from field are far from satisfactory. In a well known study to monitor volcano activities, the data yield is estimated to be only between 20% and 80% [19]. Other experiments yield similar results [11].

We observe that data losses are inevitable as long as the size of internal buffers of intermediate nodes are limited in the presence of poor link quality. This is because the sender could easily fill up the buffer given that there is not sufficient time to transmit all packets reliably. One of the key reasons for this problem is the lack of feedback of link layer information to the upper layer applications and protocols. To illustrate this, consider the following scenario: in a sensor network developed to gather real-time information of passing vehicles (e.g., the VigilNet project [9]), whenever the target appears, the traffic volume surges, causing internal packet queues to grow dramatically. At the same time, the simultaneous transmissions along the tree cause interference, decreasing the link quality of pre-established trees. On the other hand, the application tries to guarantee reliable packet delivery by retransmissions, only to cause cascading effects that further reduce available bandwidth, a problem that finally leads to packet losses when internal queues become full.

While packet losses may not be a big problem for applications that are tolerant to them due to the redundancy in sensor data, for those applications that require high fidelity of data records, the problem will be severe. For example, for a smart camera sensor network that transmits one single image using multiple encoded packets, losing any packet will lead to failures in reconstructing the complete original pictures. Another problem with lost packets is wasted energy. Consider a path of N hops where a packet got lost in the Nth hop. The energy spent on transmission and retransmission of the these $N-1$ hops will be wasted. Therefore, losing packets that have traveled for long distances is especially cost-wasting. Because of these reasons, lost packets pose severe challenges for the cost effectiveness of wireless sensor networks.

In this paper, we propose a novel idea: we use the state space model to predict link quality, and provide these estimates as a system level service to application developers. This idea is based on the premise that to achieve the best performance, the application layer behavior should be aware of the networking layer conditions, e.g., in the collection protocol, and adjusts its behavior accordingly, to achieve balanced performance with link quality. The resulting integrated framework is what we call LIPS, or ***Li**nk **P**redictions as a **S**ervice*, that represents an integrated solution that has three novel contributions.

First, LIPS presents a state-space based link prediction to select the best paths. Currently, most approaches for estimating link quality are based on metrics such as packet reception rate (PRR), link quality indicator (LQI), and received signal strength indicator (RSSI) [1], [5]. However, existing approaches have been shown to be limited in their ability to predict the future. This is because using historical data implicitly makes the assumption that future measurements may stay similar, an assumption that is frequently invalidated by the frequent variations of wireless links. Therefore, in this paper, we tackle this problem by trying to predict the expected link quality using the state space model, which has been known for its exceptional prediction power. We further integrate such predictions as the foundation for upper-layer protocol adaptations.

Second, in response to link quality changes, LIPS presents a queue management architecture based on modifying the OS kernel to support elastic applications. Specifically, we observe the following trade-off: if the link quality becomes worse, the queues of intermediate nodes will increase due to increased number of retransmissions. Therefore, we argue that the length of the queues, and especially their changing trends, reflects the link quality and provides additional information to applications. To this end, we provide a suite of APIs for the user applications to manage the queue operations.

Finally, we demonstrate one case study where the application layer reduces their data rate and performs more aggressive data aggregation. This case study demonstrates that applications can indeed use our APIs to adjust themselves to the link layer realities, and validates the feasibility of our approach.

To the best of our knowledge, this is the first integrated framework that aims to improve application layer data collection services through a co-design of link layer prediction, queue management, and API support. Furthermore, our use of state space models to predict link quality is the first as we know. Finally, the overall design is effective based on our preliminary experimental results. To help explain our results, we have used a pilot application called `surge` to demonstrate its effectiveness.

The rest of this paper is organized as follows. Section II presents the state space model for parameter estimation. Section III presents the elastic queue management. Section IV describes the programming interface available for the application layer. Section V presents the implementation details and evaluation results. Section VI reviews the state of the art. Section VII summarizes this paper.

II. State Space Model and Parameters Estimation

A. State Space Model

In this section, we describe our state space model for predicting RSSI and PRR readings. The reason that we choose RSSI and PRR instead of LQI is based on reported results from the literature that RSSI is less vulnerable to sudden changes in signal strength, and therefore, is a better estimate of packet reception ratio (PRR) [3]. State space model has been widely used in a lot of areas. It consists of two equations. One is called state (or system) equation and the other is called measurement (or output) equation. A state equation can be written as a stochastic difference equation (SDE):

$$x(k+1) = A(k)x(k) + B(k)w(k) \tag{1}$$

where $x(k+1) \in \mathbb{R}^{n\times 1}$ ($\mathbb{R}^{n\times 1}$ denotes a real vector in dimension $n \times 1$) is the state variable of the time series $\{x(k)\}_k$ determined by the previous state $x(k)$ and the noise term $w(k) \in \mathbb{R}^{m\times 1}$ introduced at each k; $A(k) \in \mathbb{R}^{n\times n}$ and $B(k) \in \mathbb{R}^{n\times m}$ are coefficients that affect the amplitudes of $x(k)$ and $w(k)$ at each k.

A measurement equation can be written as:

$$y(k) = C(k)x(k) + D(k)v(k) \tag{2}$$

where $y(k) \in \mathbb{R}^{l\times 1}$ is the measurement generated by $x(k)$ and the noise term $v(k) \in \mathbb{R}^{m\times 1}$; $C(k) \in \mathbb{R}^{l\times n}$ and $D(k) \in \mathbb{R}^{l\times m}$ are the corresponding coefficients. Note that $x(k)$ characterizes the variety and the evolution of the series $\{x(k)\}_k$. With time increasing, $x(k)$ is involving through (1) and then affects $y(k)$ through (2). This property makes it very suitable to model the random measurements as the time variety of the measurements is transplanted to the states. Also note that the noise terms $w(k)$ and $v(k)$ can represent small perturbations or uncertainties introduced at each time which increase the flexibility of the model.

Due to theses special characteristics, **we propose to use the state space model to track, model and predict the stochastic behaviors of RSSI/PRR**. Specifically, $y(k)$ in (2) is used to denote the value of RSSI/PRR at time k. For a better illustration, we state the procedure on how to predict the future measurements of RSSI/PRR as follows as well as in Fig. 1:

Prediction: *Given a batch of RSSI/PRR measurements $\{y(k)\}_{k=0}^{K}$, then the prediction of the future measurements can be divided into three steps:*

- *First, the measurements are characterized and governed by the state space model (3);*
- *Second, a parameter estimation algorithm (section II-B) is employed to compute the parameters (e.g. $A(k), B(k), C(k), D(k)$, and $x(0)$) in the model;*
- *Last, future measurements can be predicted by the explicitly by lemma 2.*

Before we go to the details of the above prediction procedure, we first introduce the following definition about multi-step-ahead prediction.

Definition 1: The N_p multi-step-ahead prediction of $y(k)$ is a prediction at a time instant $k + N_p$ making use of the measurements $y(l), l \leq k$. It is denoted by

$$\hat{y}(k + N_p|k, \theta)$$

where θ denotes the parameters of the model and will be described in detail in section II-B.

The state space predictor used to model RSSI/PRR measurements can be written as

$$\begin{aligned} \hat{x}(k+1) &= A\hat{x}(k) + Be(k) \quad \hat{x}(0) = x_0 \\ y(k) &= C\hat{x}(k) + e(k) \end{aligned} \tag{3}$$

where $y(k)$ is the RSSI/PRR measurement sampled; $\hat{x}(k)$ is the state variable characterizing the link property; A, B, and C are coefficients in the model; $x(0)$ denotes the initial state

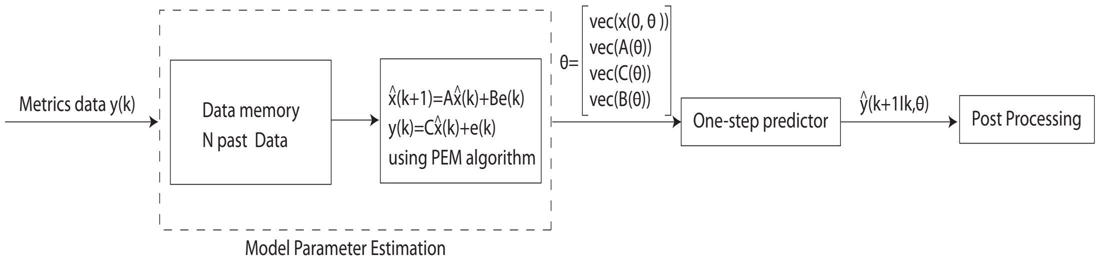


Fig. 1: The one-step-ahead prediction process of RSSI/PRR measurements. Block "Model Parameter Estimation" estimates the parameter θ and block "One-step-predictor" predicts the RSSI/PRR at next time through lemma 2. Once new measurements are provided, this process is repeated. The details of the algorithm in block 'Model Parameter Estimation' is given in table I.

condition; $e(k) = y(k) - C\hat{x}(k)$ is the error between the measured value and the predicted value $C\hat{x}(k)$ at k. A**,** B**,** C **and initial state condition** $\hat{x}(0)$ **in (3) are unknown and need to be estimated through the RSSI/PRR measurements data** $y(1), y(2), ..., y(k)$ **and then used to predict future measurements** $\hat{y}(k+N_p|k,\theta)$. Because RSSI/PRR measurements are scalars, thus in (3), $y(k) \in \mathbb{R}^{1\times 1}$, $e(k) \in \mathbb{R}^{1\times 1}$ and $B \in \mathbb{R}^{n\times 1}$.

In the next section, we are going to introduce the algorithm on how to estimate the parameters of the model and predict the future link quality information in detail.

B. *Prediction Error Minimization (PEM) Algorithm*

There are several parameter estimation algorithms, e.g. expectation and maximization [7], which yields the maximum likelihood parameter estimate. In this section, we introduce a parameter estimation algorithm – Prediction Error Minimization Algorithm [10]. In the following text, we will use x to denote $\hat{x}$ in (3) for the sake of convenience.

We first need to parameterize the model (3). The aim of parametrization is to unify the parameters of the model into a vector variable to facilitate the parameter estimation. Assume that the entries of parameters A, B, C, and $x(0)$ depend on a parameter vector θ, then (3) can be written as:

$$\begin{aligned} x(k+1|k,\theta) &= A(\theta)x(k|k-1,\theta) + B(\theta)e(k) \\ y(k) &= C(\theta)x(k|k-1,\theta) + e(k) \end{aligned} \quad (4)$$

and $x(0)$ is parameterized by $x(0,\theta)$.

It is obvious that the dimensions of A, B, C, $x(k)$, $y(k)$ and $e(k)$ are $n\times n$, $n\times 1$, $1\times n$, $n\times 1$, 1×1 and 1×1, respectively. Then, the parameter vector θ including all the entries of matrices A, B and C as well as the initial state conditions $x(0)$ can be written as

$$\theta = \begin{bmatrix} vec(x(0,\theta)) \\ vec(A(\theta)) \\ vec(C(\theta)) \\ vec(B(\theta)) \end{bmatrix}$$

where $vec(M)$ is the operator vectorizing matrix M by stacking its columns. Thus, the dimension of θ is $q\times 1$, where $q = n + n\times n + 1\times n + n\times 1 = 3n + n^2$. We also provide an example to explain the parametrization below.

Example 1: Consider the model:

$$\begin{aligned} x(k+1) &= Ax(k) + Be(k), x(0) = x_0 \\ y(k) &= Cx(k) + e(k) \end{aligned} \quad (5)$$

where

$$A = \begin{bmatrix} 0.5 \ 1.2 \\ 1.1 \ 0.6 \end{bmatrix}, B = \begin{bmatrix} 0.7 \\ 0.9 \end{bmatrix}, C = [1\ 0], x_0 = [12]^T$$

If the model is parameterized with all entries of the parameter matrices, then the following parametric model is obtained:

$$\begin{aligned} x(k+1|k,\theta) &= \begin{bmatrix} \theta(3)\ \theta(4) \\ \theta(5)\ \theta(6) \end{bmatrix} x(k|k-1,\theta) \\ + \begin{bmatrix} \theta(9) \\ \theta(10) \end{bmatrix} e(k) &, \quad x(0,\theta) = [\theta(1)\ \theta(2)]^T \\ y(k) &= [\theta(7)\ \theta(8)]x(k|k-1,\theta) + e(k) \end{aligned} \quad (6)$$

where the parameter vector $\theta = [\theta(1), ..., \theta(10)]^T$.

After parametrization, the task becomes to estimate θ.

As it stands, PEM estimates the parameters by minimizing the prediction errors. In this paper, we will concentrate on the case $N_p = 1$ where one-step-ahead prediction is involved. Thus, given a finite number of the measurements N, PEM estimates θ by minimizing a least square cost function with respect to θ:

$$J_N(\theta) = \frac{1}{N}\sum_{k=0}^{N-1} \| y(k) - \hat{y}(k|k-1,\theta) \|_2^2 \quad (7)$$

where $\hat{y}(k|k-1,\theta)$ is one-step-ahead prediction. A more specific form of $J_N(\theta)$ is given in the following theorem.

Theorem 1: The functional $J_N(\theta)$ can be written as

$$J_N(\theta) = \frac{1}{N}\sum_{k=0}^{N-1} \| y(k) - \phi(k,\theta)[x(0,\theta), B(\theta)]^T \|_2^2$$

where the matrix $\phi(k,\theta)$ with dimension $1\times 2n$ is explicitly given as

$$\phi(k,\theta) = [C(\theta)(A(\theta) - B(\theta)C(\theta))^k$$
$$\sum_{\tau=0}^{k-1} y^T(\tau) \otimes C(\theta)(A(\theta) - B(\theta)C(\theta))^{k-1-\tau}]$$

where $\otimes$ is Kronecker product. That means

$$\hat{y}(k|k-1,\theta) = C(\theta)(A(\theta)-B(\theta)C(\theta))^k x(0,\theta) + \sum_{\tau=0}^{k-1} C(\theta)(A(\theta)-B(\theta)C(\theta))^{k-1-\tau}B(\theta)y(\tau) \quad (8)$$

Proof: Follow the proof of Theorem 8.1 on page 262∼263 in [18] and set input $u(k)$ equal to 0. ■

In order to compute θ, we employs Gauss-Newton method [18] to numerically minimize the cost function. Define the error vector $E_N(\theta) = [\epsilon(0,\theta), \epsilon(1,\theta), ..., \epsilon(N-1,\theta)]^T$ where $\epsilon(k,\theta) = y(k) - \hat{y}(k|k-1,\theta)$. Note the difference between $\epsilon(k,\theta)$ and $e(k)$ is that the former is a function of θ. Then the cost function $J_N(\theta)$ can be written as

$$\begin{aligned} J_N(\theta) &= \frac{1}{N}\sum_{k=0}^{N-1} \parallel y(k) - \hat{y}(k|k-1,\theta) \parallel_2^2 \\ &= \frac{1}{N} E_N^T(\theta) E_N(\theta) \end{aligned} \quad (9)$$

Also define the derivative of $E_N(\theta)$ with the notation

$$\Psi_N(\theta) = \frac{\partial E_N(\theta)}{\partial \theta^T}$$

Then the Jacobian and Hessian of $J_N(\theta)$ (first derivative and second derivative of $J_N(\theta)$) can be expressed as [18]

$$J'_N(\theta) = \frac{\partial J_N(\theta)}{\partial \theta} = \frac{2}{N}\Psi_N^T(\theta)E_N(\theta) \quad (10)$$

$$\begin{aligned} J"_N(\theta) &= \frac{\partial^2 J_N(\theta)}{\partial\theta\partial\theta^T} \\ &= \frac{2}{N}\frac{\partial^2 E_N^T(\theta)}{\partial\theta^T\theta}(I_p \otimes E_N(\theta)) \\ &+ \frac{2}{N}\Psi_N^T(\theta)\Psi_N(\theta) \end{aligned} \quad (11)$$

where I_p is $p \times p$ identity matrix.

The Gauss-Newton approximates the Hessian by the matrix $H_N(\theta) = \frac{2}{N}\Psi_N^T(\theta)\Psi_N(\theta)$, where the first term is neglected and thus saves a lot of computation cost. When $H_N(\theta)$ is invertible, Gauss-Newton updates θ by

$$\theta_{i+1} = \theta^i - H_N(\theta_i)^{-1} J'_N(\theta_i) \quad (12)$$

where the index i denotes the ith iteration. The derivation of the update equation can be found in any optimization book, e.g. [18]. Once the cost function $J_N(\theta_i)$ reaches a tolerable threshold, the iteration can be stopped and $\theta = \theta_i$.

Remark: The matrix $H_N(\theta)$ may be singular. One possible way to solve this is through regularization, where a penalty term is added to the cost function to address the singularity. Instead of minimizing $J_N(\theta)$, the problem becomes $min_\theta J_N(\theta) + \lambda \parallel \theta \parallel_2^2$ and the update equation (12) is rewritten as

$$\theta_{i+1} = \theta_i - (H_N(\theta_i) + \lambda I_p)^{-1} J'_N(\theta_i) \quad (13)$$

where $\lambda > 0$ and $H_N(\theta^i) + \lambda I_p$ is made non-singularity.

Note that Gauss-Newton is one of the optimization algorithms to minimize a function, other algorithms such as *steepest descent method*, and *gradient projection* are also applicable to our problem.

In our paper, for convenience, we use $n = 1$ dimension state space to model PRR/RSSI. Next, we are going to derive the explicit equations of $\Psi_N(\theta)$ in this case. For the case when $n > 1$, the equations can be derived similarly.

Lemma 1: If $n = 1$, then we have

$$\Psi_N(0,\theta) = [C(\theta),\ 0,\ x(0,\theta),\ 0]$$

$$\begin{aligned} \Psi_N(1,\theta) = [&C(\theta)(A(\theta)-B(\theta)C(\theta)),\ \ C(\theta)x(0,\theta), \\ &(A(\theta)-2B(\theta)C(\theta))x(0,\theta)+B(\theta)y(0), \\ &-C(\theta)^2x(0,\theta)+C(\theta)y(0)] \end{aligned}$$

And for $1 < k < N-1$, $\Psi_N(k,\theta)$ is computed in (14),

Proof: From the definition of $E_N(\theta)$ and $\Psi_N(k,\theta)$, we have

$$\Psi_N(k,\theta) = \frac{\partial E_N(k,\theta)}{\partial\theta^T}$$

$$= [\frac{\partial E_N(k,\theta)}{\partial x(0,\theta)}, \frac{\partial E_N(k,\theta)}{\partial A(\theta)}, \frac{\partial E_N(k,\theta)}{\partial C(\theta)}, \frac{\partial E_N(k,\theta)}{\partial B(\theta)}]$$

Compute each derivative term above, $\Psi_N(k,\theta)$ in the lemma can be obtained. ■

The next lemma provides a procedure to compute the multi-step-ahead prediction after the parameters have been estimated.

$$\Psi_N(k,\theta) = \begin{bmatrix} C(\theta)(A(\theta)-B(\theta)C(\theta))^k \\ kC(\theta)(A(\theta)-B(\theta)C(\theta))^{k-1}x(0,\theta) + \sum_{\tau=0}^{k-1}(k-\tau-1)C(\theta)(A(\theta)-B(\theta)C(\theta))^{k-\tau-2}B(\theta)y(\tau), \\ (A(\theta)-B(\theta)C(\theta))^k x(0,\theta) - kC(\theta)B(\theta)(A(\theta)-B(\theta)C(\theta))^{k-1}x(0,\theta) \\ +\sum_{\tau=0}^{k-1}((A(\theta)-B(\theta)C(\theta))^{k-\tau-1}B(\theta)y(\tau) - (k-\tau-1)C(\theta)B(\theta)(A(\theta)-B(\theta)C(\theta))^{k-\tau-2}B(\theta)y(\tau)), \\ -kC(\theta)^2(A(\theta)-B(\theta)C(\theta))^{k-1}x(0,\theta) + \sum_{\tau=0}^{k-1}(C(\theta)(A(\theta)-B(\theta)C(\theta))^{k-\tau-1}y(\tau) \\ -(k-\tau-1)C(\theta)^2(A(\theta)-B(\theta)C(\theta))^{k-\tau-2}B(\theta)y(\tau)) \end{bmatrix} \quad (14)$$

Lemma 2: Given the model structure (4) and the quantities $x(0,\theta)$, $A(\theta)$, $B(\theta)$, $C(\theta)$ and $\{y(l)\}_0^k$, then the one-step-ahead prediction is computed as:

$$\begin{aligned} x(k+1|k,\theta) &= (A(\theta)-B(\theta)C(\theta))^{k+1}x(0,\theta) + \\ &+ \sum_{\tau=0}^{k}(A(\theta)-B(\theta)C(\theta))^{k-\tau}B(\theta)y(\tau) \\ \hat{y}(k+1|k,\theta) &= C(\theta)x(k+1|k,\theta) \end{aligned} \tag{15}$$

and based on one-step-ahead prediction, the multi-step-ahead prediction where $N_p > 1$ can be computed as:

$$\begin{aligned} x(k+N_p|k,\theta) &= A(\theta)^k x(k+1|k,\theta) \\ \hat{y}(k+N_p|k,\theta) &= C(\theta)x(k+N_p|k,\theta) \end{aligned} \tag{16}$$

Proof: Follow Lemma 8.2 on page 260 in [18] and set input $u(k)$ equal to 0. ■

The details in the block 'Model Parameter Estimation' in Fig. 1 including PEM algorithm discussed above is summarized in the following table:

TABLE I: Model Parameter Estimation

Given N past measurements $\{y(k)\}_{k=0}^{N-1}$ and initial parameter estimate θ_0;
① Initial Estimation:
Compute $\{\hat{y}(k|k-1,\theta_0)\}_{k=0}^{N-1}$ where $\hat{y}(k|k-1,\theta_0)$ can be calculated by (8);
Then, compute the cost functional $J_N(\theta_0)$:
If $J_N(\theta_0) < \beta$, where β is a small positive constant, output $\theta = \theta_0$;
Else, goes to ②.
② Recursive Estimation:
Compute $\Psi_N(\theta_i)$, $E_N(\theta_i)$,
then compute $H_N(\theta_i) = \frac{2}{N}\Psi_N^T(\theta_i)\Psi_N(\theta_i)$;
Compute $J'_N(\theta_i)$ by (10);
Update θ_i by (12);
Then, compute the cost functional $J_N(\theta_i)$:
If $J_N(\theta_i) < \beta$, output $\theta = \theta_i$;
Else, $i = i+1$, then repeat ②.

III. Elastic Queue Management

Having described the mathematical foundation of LIPS, in this section, we describe the elastic queue management by modifying how the operating system handles incoming packets. We choose the LiteOS operating system [2], an in-house experimental operating system for our purpose. Given that the LiteOS system has not integrated any support for queueing, we modified its communication stack to integrate with dynamic memory management to implement queueing through a doubly linked list data structure.

Figure 2 shows the communication stack that serves as the foundation for our queue management model. In this figure, both the receiving (on the left) and the sending (on the right) operations are illustrated. When the sender intends

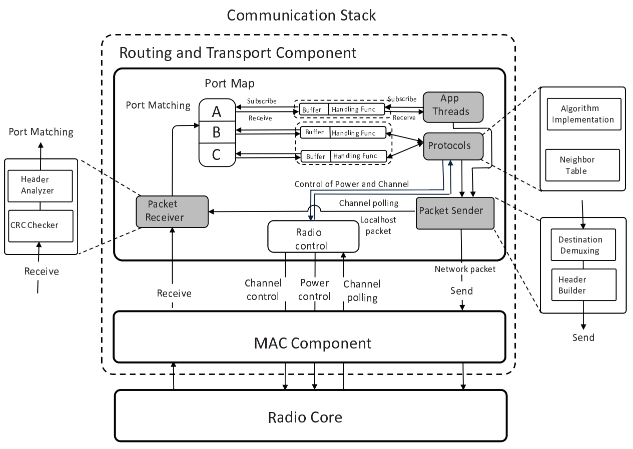


Fig. 2: The Architecture of Communication Stack

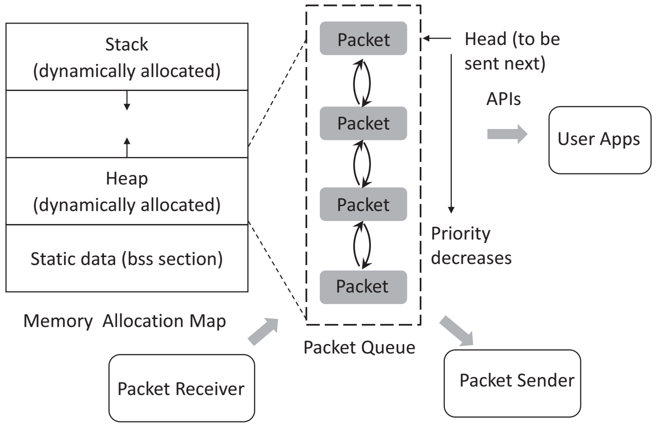


Fig. 3: The Design of the Queueing Component

to deliver packets, it puts the destination address as well as the port number for the destination node into the packet header. The packet is then delivered to the MAC component and broadcasted over the radio. When the packet is received by a neighbor, its CRC field is first checked for integrity. If this packet is sent to the current node, its port number is matched against each process that is listening to incoming packets. The thread that has a match in port number is considered the right thread for the incoming packet. The contents of the packet are then copied into the internal buffer that is provided by the thread, which is in turn waken up to handle the incoming packet. Note that this communication stack is similar to the port-based socket model in Unix, and the listening thread implement a multi-hop spanning-tree based routing protocol (e.g., the **surge** example that originally distributed by TinyOS 1.x, and reimplemented in the LiteOS environment) that will continue to forward the packets along the path.

Now we describe how we introduce the queueing model into this socket-like communication stack. Its implementation is closely integrated with the dynamic memory management

Link Prediction APIs	
setParameters	Set the link condition prediction parameters
getPRRPrediction	Get the link condition prediction for the next PRR reading
getRSSIPrediction	Get the link condition prediction for the next RSSI reading
Queue Management APIs	
getQueueLength	Get the current length of the queue
getQueueMax	Get the maximum length of the queue
Data Aggregation APIs	
getPacketFromQueue	Return a pointer to a packet in the queue
getFreePacketChunk	Return a pointer to a free slot for a new packet
releasePacketFromQueue	Release the packet as pointed by the pointer

TABLE II: User Level APIs for Queue Management

module of the LiteOS operating system. Specifically, we exploit the free space between the end of the global variables, or the **.bss** section, and the end of the growing stack, to implement a heap for memory allocation functions such as **malloc**. Whenever an incoming packet arrives, we allocate a chunk of memory whose size is the same as the size of a packet from the heap, and copy the contents of the packet to this chunk of memory. We assume that for each packet, it has been assigned a priority by the application layer. For example, a packet that contains aggregated results should have a higher priority compared to a packet that contains the initial raw data, since the former contains more condensed information. As another example, a packet that has an urgent deadline will have the highest priority to ensure that it gets transmitted first. Based on this priority, we order all packets in the doubly linked list into a queue, where those highest priority packets are always stored at the head of the queue. The design of the queue is shown in Figure 3.

Observe that there are several advantages of the queue management model being implemented as a doubly linked list instead of an array. First, it allows in-place aggregation of packets. When two packets are to be aggregated, we can allocate a free memory chunk to store the aggregation result, and release the earlier two packets. Then, the linked list is modified to insert the newly created packet in the right position. This way, we do not need to perform many copy operations to maintain the consistency of the doubly linked list. Second, by counting the total length of the queue, we can have an accurate estimate on the current congestion level. When the length of the queue grows beyond a certain threshold, the application layer can either decrease the rate of data generation, or perform aggressive data aggregation.

IV. Application Adaptation

In this section, we describe application layer adaptation. In particular, our design is based on the following premise: to achieve the best performance, the application layer behavior should be aware of the networking layer conditions in the collection protocol of wireless sensor networks, and adjust its behavior accordingly. To achieve this goal, we develop a suite of APIs as services that include not only the link prediction, but also the queue management. In this section, we first describe our proposed API, followed by an case study

```
Thread AdaptiveSampling:
while (application is running)
{
  get predicted channel condition;
  get current queue length;
  get new sample reading;
  if (channel condition gets worse)
     sampling interval increases;
  else if (channel condition gets better)
     sampling interval decreases;
  if (queue length is more than a threshold)
     perform more aggressive packet aggregation;
  wait for sampling period;
}

Thread DataTransmission:
while (there is packet in the queue)
{
  send the highest priority packet to the next node;
}
```

Fig. 4: The Design of the Adaptive Surge Case Study

to show how this API works in practice to regulate application behavior.

A. Summary of API

The APIs allow users to carry out a list of tasks such as getting the updated prediction of the link quality, reading the current size of the queue, among others. Table II shows a list of our proposed APIs for the management of link quality and queue as implemented on the LiteOS operating system in the form of a series of C functions.

These APIs are organized into three groups: link prediction, queue management detection, and data aggregation. The first group of APIs allow the user to set the prediction algorithm parameters, and read the next PRR or RSSI reading prediction. Note that the interval between current time and the next prediction is adjustable, depending on the user's needs. The second group of APIs allows the user to set the queue congestion level, where a maximum value is assumed to be the largest available queue size. The third group of APIs allows the user to manipulate packets in the queue, such as reading them, aggregating them, and releasing them, as needed.

B. Application Case Study

In this section, we design a modified version of the Surge example for multi-hop transmission of packets, by taking into account the current link layer realities. The skeleton of this application is shown in Figure 4.

As shown in this figure, the modified design has two threads: the application thread that performs adaptive sampling of the sensors, and the packet processing thread that continuously transmits the radio packets over to the next hop via the spanning tree. In the first thread, after each sample, the application checks if the radio condition is getting worse, or if the queue length is getting longer. If either of these happens, the application adjusts its own behavior by performing aggressive packet aggregation, or modifying its own sampling periods. For our example, the application that transmits raw packets adopts aggregation functions including **MAX**, **MIN**, **AVERAGE**, and **SUM** to profile the sensor readings and condense multiple packets into fewer, yet denser, representations.

V. Evaluation

In this section, we present the evaluation results of LIPS. We present the results of LIPS by implementing the state space model in Matlab. We have also modified the LiteOS operating system and the original surge application to take into account the queue management to adapt application behavior to channel quality changes.

A. Evaluation of the State Space Algorithm

To demonstrate that the state space algorithm can predict the link quality with high accuracy, we carry out experiments as following. We place two nodes, one sender and one receiver, in this experiment. The sender repeatedly send out packets with sequence numbers. The receiver receives the data, and logs RSSI and sequence numbers (used to calculate PRR). We compare the measured metrics with the prediction results. The results are shown in Figure 5, for high frequency communication (once per 100ms), and Figure 6, for low frequency communication (once per 500ms). Both the prediction curve and the error ratios are shown in these figures. We observe that the predicted results match the measurements very well. Indeed, the error ratio is mostly within a bound of 5%. These results demonstrate the effectiveness and accuracy of our proposed state space algorithm.

B. Adaptive Behavior of Surge Application

We now describe the adaptive behavior of the surge application by modifying the original version with the invocations to the APIs for queue management. To demonstrate this, we implemented an instrumented version of Surge, and the sender sends data via three hops to the receiver. After a while, we deliberately change the link quality by making two intermediate nodes farther and farther away from each other. Therefore, the link quality decreases. When this happens, the adaptive version of Surge is able to detect such changes in the length of queues, so it will reduce the sampling rate and perform aggressive aggregation, while the original version of Surge could not. In fact, the original version of Surge quickly grows the internal packet queue to the maximum size within a short period of time, and starts to lose packets. In contrast, the adaptive version of Surge is able to maintain a stable queue length, and adjusts its performance in the presence of poor link quality.

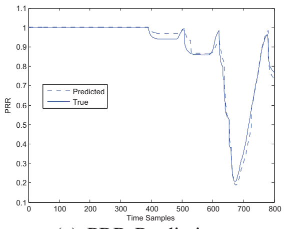
VI. Related Work

In this section, we first describe another example of the application of the state space model. This example relates to wireless communication channels. [4] used state space to model the multipath fading channels published in [6]. In this paper, the authors show that the time varying impulse responses, which capture variations in propagation environment in wireless communication networks, can be approximated in a mean square sense as close as desired by impulse responses that can be realized by the state space model. In [13] [14], novel stochastic state space models driven by Brownian motion (BM) are proposed to represent wireless communication channels. These models are time-varying and capture the spatiotemporal variations of the propagation environment.

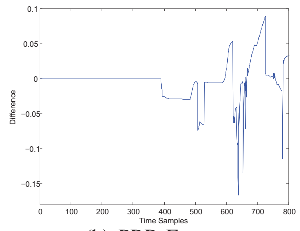
Other related works can be found in [12], [15]$\sim$ [17], where the great flexibility and utility offered by state space model generate an extensive applications to a number of different areas. The Bureau of Labor Statistics (BLS) in the U.S.A uses state space models for the production of all the monthly employment and unemployment estimates for the 50 states and the District of Columbia, where the state space models are fitted independently between states and build a model for the true population values with a model for the sampling errors [15]. The authors in [12] use state space approach to model, estimate and predict the short term power consumption which can provide an insight of the future power demand and thus make decisions on the power generation, e.g., whether activating reserve power generators or decrease the generator output. [16] generalized the linear time-discrete state space model from the single dimensional time to two dimensional space and then used it as the model for linear image processing. Moreover, [17] developed a non-Gaussian state space model for censored data. These applications guarantee its successful in predicting RSSI and PRR readings.

VII. Conclusion

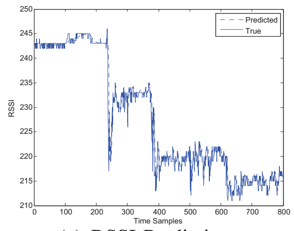
In this paper, we have presented the design and evaluation of an integrated system architecture for providing link layer quality estimation as a service to upper layer applications. The contributions of this work is three-fold. First, we present a novel, state-space driven prediction method for link quality. Our evaluation results in Matlab show that this method can predict future link quality with high accuracy. Second, we present a queue management model that integrates dynamic memory with a doubly linked list for implementing packet queues. Our evaluation results on the LiteOS platform shows that this is feasible even on extremely resource constrained environments. Finally, we present an application case study where we modified the well-known surge example with consideration for changes in link conditions. The experiences we learnt and presented can be beneficial for future designs of similar sensor network applications.



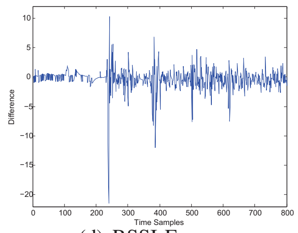
(a) PRR Prediction



(b) PRR Error

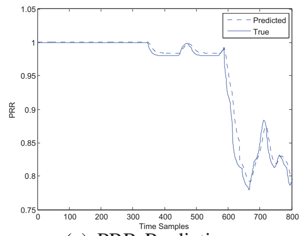


(c) RSSI Prediction

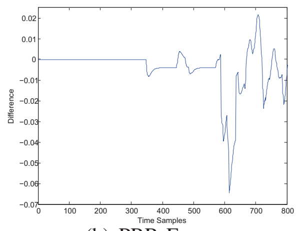


(d) RSSI Error

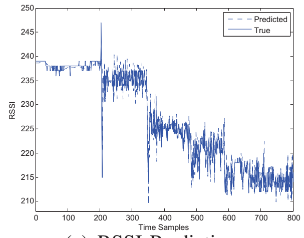
Fig. 5: Evaluations of Prediction for High-Frequency Transmission



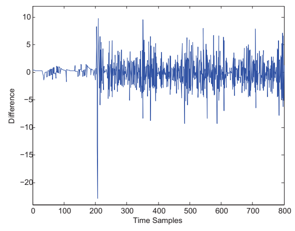
(a) PRR Prediction



(b) PRR Error



(c) RSSI Prediction



(d) RSSI Error

Fig. 6: Evaluations of Prediction for Low-Frequency Transmission

Acknowledgments

We gratefully acknowledge the anonymous reviewers for their insightful comments. This paper is supported in part by NSF grants, CNS 09-53238 and CNS 10-17156.

References

[1] CC2420 Product Data Sheet. In *http://www.ti.com*. Texas Instruments, 2010.

[2] Qing Cao, Tarek Abdelzaher, John Stankovic, and Tian He. The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks. *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 233–244, April 2008.

[3] Qing Cao, Tian He, Lei Fang, Tarek Abdelzaher, John Stankovic, and Sang Son. Efficiency Centric Communication Model for Wireless Sensor Networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.

[4] C. D. Charalambous and N. Menemenlis. A state space approach in modeling multipath fading channels via stochastic differential equations. In *Proc. IEEE Int. Conf. Commun.*, pages pp. 2251–2255, 2001.

[5] Yin Chen and Andreas Terzis. On the Mechanisms and Effects of Calibrating RSSI Measurements for 802.15.4 Radios. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, pages 256–271, 2010.

[6] S. M. Djouadi, M. M. Olama, and Y. Li. Optimal approximation of the impulse response of wireless channels by stochastic differential equations. In *IEEE Signal Processing letters*, volume 15, 2008.

[7] R.J. Elliott and V. Krishnamurthy. New finite-dimensional filters for parameter estimation of discrete-time linear guassian models. In *IEEE Trans. on Automatic Control*, volume 44, pages 938–951, 1999.

[8] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection Tree Protocol. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, page 1, 2009.

[9] Tian He, Sudha Krishnamurthy, Liqian Luo, Ting Yan, Lin Gu, Radu Stoleru, Gang Zhou, Qing Cao, Pascal Vicaire, John A Stankovic, Tarek F Abdelzaher, Jonathan Hui, and Bruce Krogh. VigilNet : An Integrated Sensor Network System for Energy-Efficient Surveillance. *ACM Transactions on Sensor Networks*, 2(1):1–38, 2006.

[10] Ljung. In *System Identification: Theory for the User*. Prentice-Hal PTR, 1999.

[11] Liqian Luo, Qing Cao, Chengdu Huang, Tarek Abdelzaher, John A Stankovic, and Michael Ward. EnviroMic : Towards Cooperative Storage and Retrieval in Audio Sensor Networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 1–22, 2007.

[12] Xiao Ma, Husheng Li, and Seddik M. Djouadi. Stochastic modeling of short-term power consumption for smart grid: A state space approach and real measurement demonstration. In *45th Conference on Information Sciences and Systems*, 2011.

[13] M.M. Olama, S.M. Djouadi, and C.D. Charalambous. Stochastic channel modeling for ad-hoc wireless networks. In *Proceedings of the American Control Conference*, pages 6075–6080, 2006.

[14] M.M. Olama, S.M. Djouadi, and C.D. Charalambous. Time varying channel modeling for ad-hoc mobile wireless networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, volume 3, pages 1277 – 1282, 2006.

[15] D. Pfeffermann and R. Tiller. State-space modeling with correlated measurements with application to small area estimation under benchmark constraints. In *Southampton, UK, Southampton Statistical Sciences Research Institute, 21pp. (S3RI Methodology Working Papers*, 2003.

[16] R.P. Roesser. A discrete state-space model for linear image processing. In *IEEE Transaction on Automatic Control*, volume AC-20, pages 1 – 10, 1975.

[17] R.L. Smith and J.E. Miller. A non-gaussian state space model and application to prediction of records. In *J.R. Statistic. Soc. B*, volume 48, pages 77 – 89, 1986.

[18] M. Verhaegen and V. Verdult. In *Filtering and System Identification: A Least Squares Approach*. CAMBRIDGE, 2007.

[19] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 7, 2006.