Empirical Model Development for Message Delay and Drop in Wireless Sensor Networks

Gursel Serpen* and Zhenning Gao

Abstract

Simulation of wireless sensor networks with very large number of motes poses significant challenges with respect to computational complexity. Application level code prototyping with reasonable accuracy and fidelity can however be accomplished through simulation that models only the effects of the wireless and distributed computations which materialize as delay and drop for the messages being exchanged among the motes. This study pursues that idea of empirical modelling of delay and drop and employs such a model to affect the reception times of wirelessly communicated messages. The delay and drop is therefore, modelled as random variables with probability distributions empirically approximated based on the data reported in the literature. The paper concludes with a case study that employs the proposed empirical delay and drop models for multilayer perceptron neural networks distributed across a wireless sensor network for a classification task on the Isolet dataset.

Keywords: Wireless sensor network application; simulation; message packet drop and delay; artificial neural networks; empirical probability distribution model

1. Introduction

Simulation of wireless sensor networks (WSN) may incur very high computation cost depending on the size of the sensor network in terms of the mote count and the detail level of simulation such as hardware emulation, bit-level simulation, packet-level simulation or application-layer simulation. Determining the appropriate level of simulation is of paramount importance to manage the computational complexity for establishing the feasibility of a simulation. Simulators that are intended for bit or packet level emulation or simulation are unnecessarily detailed for prototyping distributed application-level code. Therefore, it is desirable to develop simulation frameworks or tools for distributed applications to maintain the balance between good accuracy and reasonable computation cost.

* Corresponding author. Tel.: 1-419-530-8158; fax: 1-419-530-8140.
E-mail address: gursel dot serpen at utoledo dot edu
WSN simulators can be classified into three major categories based on the level of complexity: bit level, packet level and algorithm level. As the complexity goes up such as for a bit-level emulation, the time and memory consumption of the simulation grows. It is desirable to select the level of simulation based on the rigor requirements of the experiment. For instance, a timing-sensitive MAC protocol would probably need a bit level simulation while an algorithm level simulation is sufficient to test the prototype developed for an agriculture management application.

Bit-level simulators model the CPU execution at the level of instructions or even cycles, and they are often regarded as emulators. Examples are TOSSIM [1], which is a both bit and package level simulator for detailed simulation of TinyOS-based motes, and Avrora [2] which is another bit-level simulator that is open source and built using the Java programming language. Packet level simulators implement the data link and physical layers in a typical OSI network stack. The most widely used simulator is ns-2 [3], which is an object-oriented discrete event network simulator built in C++. A second example OPNET [4] is a commercial simulator, which provides a simulation environment for a variety of networking environments. Algorithm level simulators focus on the logic, data structure and presentation of algorithms. They do not consider detailed communication models. For instance, Shawn [5] is a simulator implemented in C++, which has its own application development model or framework based on so-called processors. The development of Shawn is predicated on the premise that there is no (or barely any) difference between a complete simulation of the physical environment (or lower-level networking protocols) and the alternative approach of simply using well-chosen random distributions on message delay and drop for algorithm design on a higher level, such as localization algorithms. Shawn simulates the effects caused by a phenomenon instead of the phenomenon itself. For example, instead of simulating a complete MAC layer including the radio propagation model, its effects (packet drop and delay) are modeled by Shawn. Therefore, the simulation time and the computation effort through Shawn is significantly reduced compared to those of other simulators.

Although Shawn’s simulation philosophy is appropriate for application-level code development, being a generic simulator to accommodate a comprehensive set of simulation cases for a large selection of applications, it suffers to some degree from overhead due to its generic nature. Much of this overhead can be eliminated if the philosophy of simulating the “effects rather than the phenomenon itself” is implemented directly into the algorithm of the application that needs to be simulated for distributed computation on a wireless sensor network. The proposed approach that trades generic nature of Shawn with more efficient domain-specific implementation would mainly entail the following steps. Any given application algorithm that is to be implemented through distributed computation would first be partitioned into subparts as in functional decomposition, and the communication interface among these subparts would be identified. Consequently, the messaging requirements are established and delay or drop is injected into the communication pathways through which packets carry those messages. Accordingly, for the purposes of verification and validation of the application layer code, modeling of effects of wireless communication as “delay and drop” is promising to deliver the appropriate level of accuracy and computational cost for simulations.

2. Reduced Complexity Simulator Model

This section presents the empirical models for message packet delay and drop phenomena in wireless sensor networks (WSN). The probability of packet drop or delay during wireless transmission in WSNs is highly dependent on the specific implementation of the network and its protocol stack. There are many factors at play, such as the topology of a network, routing and MAC protocols, network traffic load, etc. It is not desirable to have the model for the probability distribution for drop or delay limited to a certain scenario (using certain protocols, set for a number of nodes, or set for a topology, etc.) since the results of such a study would not be applicable in general terms. The model to be developed instead should be generalized enough to be applicable for the widest variety of WSN realizations, implementations and applications possible. One readily available option to develop or formulate a model for packet delay and drop in general terms is to leverage the empirical data reported in the literature, which is the venue pursued in our study.

We conducted a survey to compile the empirical data of delay and drop reported in the literature [6]. In the process, we studied the simulation scenarios and compiled a record of the simulation settings and results. The simulation settings included routing protocols, MAC protocols, simulator type, number of motes, field size, radio range and other settings (traffic, source count, dead node count etc.). The simulations of interest reported the delivery rate (which is related to the probability of drop) and delay, which were extracted from tables and figures in the surveyed literature.
The packet delivery ratio or rate that was recorded in each study [6] is considered as the main variable of interest. Denoting the packet delivery ratio as \( p_{\text{delivery}} \), the probability of drop, \( p_{\text{drop}} \), can be calculated through \( p_{\text{drop}} = 1 - p_{\text{delivery}} \). Specific values for the packet delivery ratio versus node count for a number of WSN topologies and protocol stack implementations, which were used as the data to build the empirical model for the probability of drop, are retrieved from the same studies cited herein. The data points are chosen based on the following specifications:

- The node count is one of the primary independent variables, which means the data is collected for different node count values.
- The density of node distribution within the WSN topology will stay “approximately” the same although the node count may vary. This means that the area of deployment for the network or the transmission range should change to keep the node distribution density the same.
- Other factors such as the changing network traffic load or the static or time-varying percentage of dead nodes could not be considered due to lack of sufficient empirical data, relevant simulations or experiments.

Establishing the above specifications is intended to ensure that packet drop probability is fundamentally affected by the number of transmission hops only, which is assumed to approximate the distance between the sender and receiver mote pair under the assumption that the motes are uniformly distributed across the deployment field.

The relationship between the probability of drop and mote count for a variety of routing protocols is reported in a number of studies in the literature [6]. The routing protocols included QoS [9], Speed [9], GBR [7], LAR [10], LBAR [10], AODVjr [10], BVR [8], DD [11], and EAR [7]. Denoting the node count in a WSN as \( n_{\text{nodes}} \), analysis of empirical data shows that \( p_{\text{delivery}} \) decreases when the value of \( n_{\text{nodes}} \) increases. The relationship appears to be linear in general. Since these data are due to specific experiments, in order to generalize, it may not be a good idea to make the model fit the data precisely. Therefore, the linear regression (versus a polynomial) for fitting these data points is a reasonable option. Then the resultant empirical model is given by \( p_{\text{drop}} = 1 - p_{\text{delivery}} = 1 - \left( (\beta_0 + \beta_1 \times n_{\text{nodes}}) / 100 \right) \), where coefficients \( \beta_0 \) and \( \beta_1 \) are real numbers for the linear model and \( n_{\text{nodes}} \in [0,1000] \). The coefficients \( \beta_0 \) and \( \beta_1 \) calculated for each routing protocol case are shown in Table 1.

<table>
<thead>
<tr>
<th>Routing protocol</th>
<th>EAR</th>
<th>GBR</th>
<th>BVR</th>
<th>QoS</th>
<th>Speed</th>
<th>LBAR</th>
<th>LAR</th>
<th>AODVjr</th>
<th>DD</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \beta_0 )</td>
<td>100.82</td>
<td>94.50</td>
<td>94.44</td>
<td>97.00</td>
<td>97.40</td>
<td>95.79</td>
<td>92.57</td>
<td>90.57</td>
<td>89.60</td>
</tr>
<tr>
<td>( \beta_1 )</td>
<td>0.0107</td>
<td>0.1130</td>
<td>0.0760</td>
<td>0.0980</td>
<td>0.0840</td>
<td>0.0198</td>
<td>0.0154</td>
<td>0.0154</td>
<td>0.0440</td>
</tr>
</tbody>
</table>

The number of hops can be used as the primary factor affecting the probability of drop. For a two-dimensional deployment topology for a WSN, let \( n_{\text{hops}} \) denote the average hop count between a source and a destination mote pair. Defining \( p_{\text{drop}} \) in terms of \( n_{\text{hops}} \) is of interest. Given that we know the value for \( n_{\text{nodes}} \), which is the number of motes in the WSN, a relationship between \( n_{\text{nodes}} \) and \( n_{\text{hops}} \) needs to be derived for a given specific deployment topology. For instance, \( n_{\text{nodes}} \) nodes may be assumed to be uniformly randomly distributed across a square deployment area. Consequently, the number of motes along any edges will be approximately \( \sqrt{n_{\text{nodes}}} \). Assume the sink mote is located at the center, while the source node is close to one of the corners. Average hop count for any given message, \( n_{\text{hops}} \), can be approximated by the length of the diagonal in terms of number of hops divided by two. Then the relationship between \( n_{\text{nodes}} \) and \( n_{\text{hops}} \) is given by \( n_{\text{nodes}} = 2(n_{\text{hops}})^2 \). In the worst case for a source and sink mote pair where the motes are located at the end points of a given diagonal, the relationship between these two variables becomes \( n_{\text{nodes}} = (n_{\text{hops}})^2 / 2 \). Consequently, the hop count values are bounded as follows: \( 1 \leq n_{\text{hops}} \leq \sqrt{2n_{\text{nodes}}} \). The square topology assumed for the above analysis is a reasonable approximation for many of the deployment realizations. If necessary, other topologies can also be readily analyzed following a similar approach. In more general terms, the relationship between \( n_{\text{nodes}} \) and \( n_{\text{hops}} \) can be represented as \( n_{\text{nodes}} = \tau n_{\text{hops}} \), where \( \tau \) is the coefficient whose value is positive and will vary based on a number of WSN-related parameter settings including the shape of the topology and the density of mote deployment. In the linear regression curves obtained for each routing protocol earlier, the coefficients \( \beta_0 \) and \( \beta_1 \) and the parameter \( n_{\text{nodes}} \) values are substituted to yield the empirical models shown in Table 1. The calculations of \( \tau \) are done based on a specific topology implemented in the literature. When any of these models is employed in a simulation study, specific \( \beta_0 \) and \( \beta_1 \) values can be generated based on how “similar” the routing protocol to one of the given in Table 1.
Delay is an inherent property of WSN operation with respect to wireless communications. As any literature survey will indicate, the length of delay varies for different implementations (such as variations in number of nodes, routing protocol, MAC protocol, packet length, traffic load etc.). Figure 1 shows the histogram for the delay based on the literature survey data. In a real scenario where one or more neurons are embedded into a given mote, the exchange of neuron outputs among motes will be subject to certain delay that is inherent in wireless communications. This delay, which will dictate the duration of a waiting period by a given neuron for its inputs to arrive from other neurons on other motes is not a fixed value but rather a random variable. The delay-induced wait time will be denoted as $t_{\text{wait}}$. Note that $t_{\text{wait}}$ is both application dependent and network dependent: its value was found to vary from 10 ms to 3000 ms per the literature survey. For a specific network, the delay between for a pair of motes varies substantially from one pair to another, and even for the same pair of motes the delay variance is significant. Additionally, the maximum delay could be much larger than the mean delay. In simulating a neural network embedded across motes of a wireless sensor network, the $t_{\text{wait}}$ is set according to the mean delay value and the specific network topology to make sure that a good number of inputs successfully arrive for any given neuron to be able to calculate its own output.

Per the literature, a specific delay distribution is highly dependent on many factors such as the MAC protocol, traffic, queue capacity, channel quality, back-off time setting in MAC protocol, etc. [12,13,14,15, and 16]. It is impossible to get a highly accurate model of delay distribution considering so many factors play a role in affecting its value. A reasonably good but approximate model however can be formulated by using the Gaussian distribution which has been used in the literature to model the delay distribution [13, 14], and has also been shown to be relevant in other literature studies [16, 17 and 18]. Empirical evidence suggests that the delay distribution is truncated and heavy-tailed [16, 18]. Consequently, a truncated Gaussian distribution for modeling the delay variance can be employed. The truncated Gaussian distribution is the probability distribution of a normally distributed random variable whose value is bounded. Suppose $x \sim N(\mu, \sigma^2)$ has a normal distribution and lies within a range of $(a, b)$, then $x$ conditional on $a < x < b$ has a truncated normal distribution. Its probability density function $f$ is given by

$$f(x; \mu, \sigma, a, b) = \frac{1}{\sigma} \phi((x-\mu)/\sigma)/(\Phi((b-\mu)/\sigma)-\Phi((a-\mu)/\sigma)),$$

where $x$ is the random variable, $\mu$ represents the mean, $\sigma$ represents the standard deviation, $a$ represents the minimum value, $b$ represents the maximum value, $\phi(.)$ is the probability density function of the standard normal distribution, and $\Phi(.)$ is the cumulative distribution function.

The overall delay for a given neuron output is positive integer-valued by definition. This parameter value, namely overall delay or OD for short, is computed by summing per hop delays for the total number of hops between the sending-receiving neuron pair $i$ and $j$, and dividing their sum by the $t_{\text{wait}}$ parameter value. Truncated Gaussian distribution is used as the model for the average per hop delay (APHD) parameter. In other words, the computation employs the following formula: $OD_{ij} = \left[\frac{1}{n_{\text{hop}}} \times \text{APHD} \right] / t_{\text{wait}}$. The parameter $t_{\text{wait}}$ is defined in terms of three other parameters as follows: $t_{\text{wait}} = \frac{\phi \times \text{APHD}}{\text{n}_{\text{hop}} \times \text{l}_{\text{max}}}$, where $\phi$ is a coefficient to be set in the simulator, $\mu$ is the mean value of the truncated Gaussian distribution, and $l_{\text{max}}$ is the max hop count of the topology being considered. For ease of computation, the mean value of truncated Gaussian distribution may be normalized to relocate it to the value of 1.0. Based the empirical studies in the literature [12,13], other parameters of the truncated Gaussian distribution may be set as $a=0.3$, $b=5$ and $\sigma=0.6$. 

![Figure 1. Histogram of Delay for Empirical Data Reported in Literature](image-url)
3. Simulation Study

The simulation study presents the classification performance of the WSN-MLP design using the delay and drop empirical models presented above. We will consider, without loss of generalization, a multilayer perceptron (MLP) type artificial neural network with at least three layers of neurons, namely an input layer, one or more hidden layers, and an output layer. The input layer is not considered as a “true” layer since no computation is performed by the neurons in that layer. Neurons in the input layer simply distribute the components of an input pattern vector to neurons in the hidden layer without any other processing. Distribution of training patterns can be accomplished by either a multi-hop routing scheme or by a gateway or cluster-head mote that can reach all the WSN motes directly. Outputs of neurons in one layer must be communicated to inputs of neurons in the other layer during training and following the deployment. When delay occurs and its value varies and, in the worst case, the drop happens, past values of the output for the neuron whose output is delayed are made available to the input of the neuron whose dynamics needs to be updated.

The simulator was custom developed in-house and implemented in C++. It simulates the delay and drop effects on the transmission of neuron outputs, thus provides a highly computationally efficient simulation bypassing the details not relevant for performance assessment associated with application development within a wireless sensor network context. Accordingly, a probabilistic model for delay and drop has been developed and employed in the simulation study [6]. The simulator implements several phases of data access, initialization, delay and drop instantiation, neural network training, and performance recording. Training of the MLP network entails forward propagation, back propagation, and weights update. After each complete iteration over the entire training dataset, the MLP performance is validated on the testing data. The delay and drop affect transmission of outputs from the hidden layer neurons in the forward propagation step, and the error signals generated at the output layer and communicated back to hidden layer neurons in the backward propagation step. In this study, the Isolet data set from the UCI machine learning repository [19] was employed for the supervised classification task.

The parameter \( t_{wait} \) is set to different values to vary the delay and drop probabilities. In order to exclude the cases where motes are positioned in outlying or extreme areas, the parameter \( l_{max} \) is set to a value, which covers approximately 90% of node pairs. The coefficient \( \sigma \) is set to different values to cause the percentage values of neuron output delay to vary. The value of \( \sigma \) should be positive. A too small \( \sigma \) value results in nearly all the packets to be dropped, while a very large value for this parameter will result in no dropped packets. By exploration, we determined that (0.3, 2.1) is a reasonable range for \( \sigma \) which makes the percentage of neuron output delay to range from 0.4% to 99%. Consequently, in this study, we equally divided the range of values for \( \sigma \) and therefore set it to 0.3, 0.6, 0.9, 1.2, 1.5, 1.8 and 2.1.

In simulations, we also assumed that there is a gateway mote, which can communicate with each mote in the WSN directly through single-hop transmission: additionally, potential delays or drop for the communications originating from the gateway mode, were not considered. Simulation is repeated five times with different initial weights. The packets carrying neuron outputs and error signals are subject to delay and drop during the training phase only and not during the testing phase.

In the Table 2, the max and min performance of WSN-MLP as well as the performance of the non-distributed MLP (through in-house implementation) in comparison to other machine learning classifiers reported for the same dataset in the literature are presented for comparison. Results show that the WSN-MLP has a competitive performance in comparison with a very diverse set of machine learning classifiers. The maximum performance for the WSN-MLP is among the upper-middle tier of the entire set of classifiers included in the same table.

4. Conclusions

This paper presented an empirical model for delay and drop probability distribution associated with messaging in wireless sensor networks, which serve as a computing platform for parallel and distributed processing for neurocomputing. Utility of the delay and drop empirical model was demonstrated through a case study that employed a multi-layer perceptron neural network for classification. The MLP neural network was distributed across the motes of a wireless sensor network for parallel computation for the classification task using the Isolet dataset. Wireless message delays and drop were implemented using the empirical models developed. Simulation results showed that the proposed simulation framework with reduced complexity is conducive to facilitating the effective learning of the classification task.
Table 2. Comparison of Classification Accuracy for Isolet Data Set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Reference</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
<td>[20]</td>
<td>97.0</td>
</tr>
<tr>
<td>kLOGREG</td>
<td>Kernelized logistic regression</td>
<td>[20]</td>
<td>97.0</td>
</tr>
<tr>
<td>WSN-MLP (max)</td>
<td>Multilayer Perceptron on Wireless Sensor Net</td>
<td></td>
<td>95.9</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron (non distributed)</td>
<td></td>
<td>95.8</td>
</tr>
<tr>
<td>WSN-MLP (min)</td>
<td>Multilayer Perceptron on Wireless Sensor Net</td>
<td></td>
<td>94.9</td>
</tr>
<tr>
<td>NBC</td>
<td>Naive Bayes</td>
<td>[21]</td>
<td>84.4</td>
</tr>
<tr>
<td>C4.5</td>
<td>C4.5 Tree</td>
<td>[21]</td>
<td>80.2</td>
</tr>
<tr>
<td>kNN(LDA)</td>
<td>K-Nearest Neighbor with Principal component analysis</td>
<td>[22]</td>
<td>71.2</td>
</tr>
<tr>
<td>kNN(PCA)</td>
<td>K-Nearest Neighbor with Linear discriminant analysis</td>
<td>[22]</td>
<td>59.9</td>
</tr>
</tbody>
</table>

References