Using End-to-End Data to Infer Lossy Links in Sensor Networks

Hung X. Nguyen Patrick Thiran
School of Computer and Communication Sciences, EPFL
CH-1015 Lausanne, Switzerland
{hung.nguyen, patrick.thiran}@epfl.ch

***Abstract*— Compared to wired networks, sensor networks pose two additional challenges for monitoring functions: they support much less probing traffic, and they change their routing topologies much more frequently. We propose therefore to use only end-to-end application traffic to infer performance of internal network links. End-to-end data do not provide sufficient information to calculate link loss rates exactly, but enough to identify poorly performing (lossy) links. We introduce inference techniques based on Maximum likelihood and Bayesian principles, which handle well noisy measurements and routing changes. We evaluate the performance of both inference algorithms in simulation and on real network traces. We find that these techniques achieve high detection and low false positive rates.**

I. Introduction

Sensor technology has matured to the point where several sensor networks have been built for real-life applications such as, to name a few, scientific data gathering, environment monitoring (air, water, soil, chemistry), surveillance, smart homes, smart offices, personal medical systems and robotics [1], [2]. Real-life experiences of sensor networks [2], [3] reveal that networks experience long periods of outage and that network managers do not have the appropriate tools to diagnose the problems. These experiences have demonstrated the obvious need for network monitoring tools. Monitoring wireless sensor networks (WSNs) has thus recently generated a surge of interest from the research community [3]–[8]. Similarly to other applications for sensor networks, monitoring systems are also restrained by the following limitations [9]:

1) Sensor nodes use a broadcast communication paradigm and have stringent bandwidth constraints.
2) Sensor nodes have limited resources (power, memory and computational power). Any algorithm therefore must sparingly use the resources that exist.

Many existing routing algorithms (e.g., [10]) for sensor networks require that each node continuously monitor the quality of links to all of its neighbors and use this information to select the routes. Each sensor node has thus a complete local knowledge about its neighboring links. However, regularly sending the loss information from sensor nodes to the sink requires significant communication overheads. Furthermore, contrary to wired networks where network monitoring information can be delivered reliably, reports sent from sensor nodes to the sink also suffer losses. In consequence, the sink has no guarantee that it will receive up-to-date information of link loss rates.

Each sensor network is deployed to support a certain application, hence its nodes regularly send application data to the sink. End-to-end application data can therefore be a valuable and reliable network monitoring tool, if they can be used for network diagnosis purposes. In this paper, we propose two algorithms that help network managers identify links with high loss rates (*lossy links*) without requiring each sensor to send their estimations of neighboring links to the sink. Our method is to passively monitor application traffic between sensor nodes and the sink, and to use the end-to-end data observations to infer the performance of interior links. We would like to emphasize here that our objective is *not* to infer instantaneous loss rates of all links (which is done by the routing algorithms), nor to locate every link with a high loss rate over a short period of time, but rather to identify links with high loss rates that are persistently used to route traffic from the sensors to the sink. Indeed, it is not essential to locate all lossy links in a wireless sensor network, as the network should quickly self-organize around them. The problem occurs when all links surrounding a sensor node have high loss rates because of low battery or physical obstacles. In this case, there is no other choice to access this node than to use a lossy link. There are a number of ways in which a network can benefit from the identification of lossy links. First, information on bottlenecks within the network could be used to tune network algorithms. Second, such information gives early warnings to the network managers about the status of the nodes surrounding the lossy links, as failing nodes may no longer be able to generate these warnings themselves.

Our contributions in this paper are: (i) a feasibility study of the lossy link inference problem in a real sensor network, including the analysis of link loss rates and end-to-end data transmissions, (ii) a new efficient and simple maximum likelihood algorithm (named LLIS in this paper), based on a set cover heuristic, to isolate links with high loss rates, despite routing changes and noisy measurements, (iii) thorough validations both by simulations and on real sensor network measurements, and (iv) a comparison of the LLIS algorithm with a more complex Bayesian inference technique (named MCMC) adapted to sensor networks with routing changes, and with existing passive WSN monitoring algorithms.

The paper is organized as follows. In Section II, we relate and contrast our approach with existing work on network monitoring and diagnosis. We describe our analysis of packet

1-4244-0222-0/06/$20.00 (c)2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the Proceedings IEEE Infocom.

losses in a real sensor network in Section III, from which we deduce a network model in Section IV. In Section V, we explain our inference techniques. In Section VI, we validate our algorithms in three ways: we compare our results with other approaches in the literature; we evaluate the performance of our algorithms in different network scenarios with simulations; and finally we validate our algorithms with data of a real sensor network. We conclude the paper in Section VII.

II. Related Work

A. Monitoring Sensor Networks

Like monitoring techniques of other networks, sensor network monitoring techniques are separated into two groups: active and passive monitoring.

1) Active Monitoring: Active monitoring methods [3]–[6] require the injection of additional traffic into the network. Zhao et al. [4] propose a scheme (eScan) where each node monitors its remaining energy level. Whenever the energy level of a node drops significantly, it reports this energy level and its location to the manager. The same authors also propose a hierarchical monitoring scheme, in which network nodes continuously calculate some high level, summarized information such as the average or maximum energy level among all nodes in the network [5]. When the high level information indicates anomalies, a low level and more energy consuming procedure such as the eScan scheme is used to accurately locate the trouble spots. Hsin et al. [6] proposed a different monitoring scheme for WSNs where each sensor monitors its neighbors by periodically sending them active probes. More recently, Tolle et al. [3] propose the Sensor Network Management System (SNMS) scheme to poll sensor nodes for monitoring information. All four schemes require additional traffic to be injected in the network, which may reduce the lifetime of wireless sensor networks because of communication costs.

2) Passive Monitoring: Recently, [7] and [8] have shown that the application data that the sensors send to the sink can be used to monitor the network. Hartl et al. [7] use the traditional network tomography techniques for wired-line networks [11] to infer loss rates of nodes in a reverse broadcast tree, whereas Mao et al. [8] use a factor graph decoding method to infer the link loss rates. These two loss rate inference methods require two restrictive assumptions: first, a stable, fixed network topology, and second, the aggregation of application data and transit traffic at each sensor node before being sent to the sink. The first assumption is not verified in many sensor networks, as we show in Section III-D. The second assumption is needed in these algorithms to guarantee a strict correlation between packets sent from sensor nodes to the sink, but it relies on a specific data forwarding scheme, where each node has to wait for packets from all of its children before forwarding upward and cannot therefore be applied to other more general settings.

B. IP Network Tomography

Wired-line IP Network tomography is an active research area (see e.g. [11] for a summary). Most tomography systems assume inherent correlations between the probe packets by using either multicast packets, or a cluster of unicast packets. Recently, Padmanabhan et al. [12] and Duffield [13] used uncorrelated end-to-end packets to locate lossy links in IP networks. A notable feature of the model considered in [12], [13] is that the link loss rates are not statistically identifiable from the data (the server-to-client loss rates), meaning that there exist different sets of link loss rates that give the same statistical distribution of data. Nevertheless, their methods are quite successful in identifying the lossiest links of the network, both in simulated and real networks. The underlying reason behind this success is that identifying bad links amounts to finding the smallest set of lossy links, which does not require the computation of the exact loss rates.

The two inference methods of this paper are closely related to those of [12], [13].

The first one is the smallest consistent failure set (SCFS) technique proposed by Duffield [13], who formulates the tomography problem as a set-cover problem and solves it on a tree topology. The main differences with the LLIS algorithm introduced in Section V-A are due to the specificities of wireless sensor networks: multiple routing trees and noisy monitoring information, which are absent from the SCFS algorithm, but need to be factored in our LLIS algorithm.

The second technique is a Gibbs sampler, which is one of the most accurate techniques developed in [12]. Its main idea is to estimate the distribution of loss rates on each link from observed end-to-end data and prior knowledge about network links. It is in principle more accurate, but also much more computationally intensive, than set cover heuristics. We adapt this technique to tolerate routing changes in Section V-B.

C. Analysis of Losses in Sensor Networks

Recently, quite a few studies carried out on wireless sensor networks with a significant number of nodes (most often Berkeley Mica and Mica2 motes), and in several different environments, have given a better understanding of packet delivery performance [14]–[17]. Ganesan et al. [14] and Zhao et al. [15] deployed experimental platforms that consist of hundreds of Rene [14] or Mica [15] motes to capture link, MAC and application layer characteristics in wireless multihop communication. Their data firmly establish that packet reception rate is not strictly correlated with the distance between the transmitter and the receiver. Especially, there is a "grey area" where two neighboring receivers can have very different reception rates. In a series of studies conducted at Berkerley, Woo et al. [16] used experimental measurements to derive a packet loss model based on aggregate statistical measure such as mean and standard deviation of reception rates. They showed that, despite statistical variation, a simple Binomial distribution is a fairly good model to approximate the number of losses in sensor links. The parameter of the Binomial distribution (i.e., packet loss rate) is stable in the absence

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the Proceedings IEEE Infocom.

of external physical influences. As a result [17], routes with good end-to-end delivery quality (i) are the ones using high quality and symmetric links at each sensor node, and (ii) are normally stable over several minutes. These conclusions have been supported by additional experiments and demonstrations of new protocols [1]. The minimum expected transmission (MT) routing protocol of [17] is widely used in many different real-life applications [1], [2].

III. Empirical Studies of Link Loss Rates and End-to-End Data Transmissions

In order to develop a practical lossy link inference algorithm, we first need to understand the packet loss behavior of the network. In this section, we study packet losses and end-to-end data transmissions in a real sensor network, the Sensorscope [2]. The observations made in this section will drive the network model and the algorithms of the following sections. Sensorscope is a wireless sensor network deployed in the BC building on the EPFL campus mainly for research purposes. Contrary to existing analyses of real sensor networks [16]–[18], which concentrated mostly on understanding packet delivery on individual wireless links or on the performance of routing protocols, our analysis focuses on the combined effects of wireless links and routing algorithms on the packet losses and end-to-end data delivery in real sensor networks. The analysis of Sensorscope, however, is not our primary objective. Indeed, all of our findings from Sensorscope agree with previous findings albeit in different settings (i.e., network topology and surrounding environment). We analyze Sensorscope data mainly to understand packet losses, link loss rates and routing in real sensor networks. We then use our knowledge of losses to develop suitable inference algorithms for sensor networks, which is our main objective. Specifically, we analyze the data to answer three questions:

1) How do link loss rates vary among the links? The end-to-end loss rate of a path is influenced by the loss rates of all links in the path. To infer lossy links from end-to-end observations, we need to know the distribution of link loss rates among different links. We call the variation of link loss rates among different links the *spatial* variation.
2) How do link loss rates vary with time? Inferences from the measurements are most valuable when they can be used for prediction. This happens only when the qualities that we try to infer remain stationary for a certain period in the future. We call the variation of link loss rates with time the *temporal* variation.
3) How stable are end-to-end paths? The stability of end-to-end paths directly affects the way data should be collected and analyzed.

A. Hardware Platform and Analysis Method

Sensorscope consists of approximately 20 Mica2 and Mica2dot motes, each equipped with a variety of sensors such as light and temperature. The motes are static and do not move. A typical routing tree in Sensorscope is shown in Figure 1. The motes use tinyOS (version 1.17) [19]. Basic tinyOS multihop routing and the full AM stack implementations are used. At the MAC layer, Sensorscope uses B-MAC [20] and its low-power listening scheme. The routing algorithm is the MintRoute algorithm of tinyOS [17]. The traffic traces were gathered at the sink. The sink is connected to a computer where data is stored and processed. The data that we analyze here were collected over 10 days, from 26.12.2004 to 6.1.2005. The network settings during this period are as follows [2]: Low Power Listening is set to 4; RF power is -14 dBm; each mote sends application data every 2 minutes, reports its parent every 5 minutes and its estimations of neighboring links every 15 minutes to the sink.

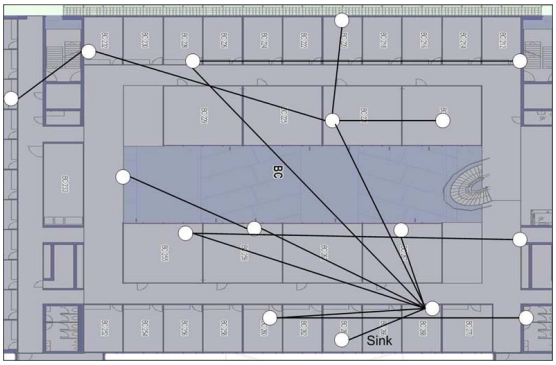

Fig. 1. A typical routing graph of the Sensorscope network. The Sensorscope network consists of approximately 20 motes located on two floors of the BC building at EPFL. The motes form a multihop sensor network to carry data to the sink.

All communications in Sensorscope are done via wireless channels. Hence, similar to application data, monitoring data in Sensorscope are unreliably delivered: during the period of observation, only 60% of monitoring data were successfully delivered to the sink.

In this paper, we are only interested in the long-term behavior of network links, and therefore will only work with average, and not instantaneous, loss rates. These average loss rates are calculated over a time window of length T. The choice for the values of T is a trade-off between the accuracy (the longer T, the smaller the variance) and ability to detect changes at small time scales (the shorter T, the better the average follows changes at small time scales). In Sensorscope, each sensor node is configured to send 30 packets to the sink every hour. To guarantee that we have enough data to accurately calculate link and path loss rates, we choose a time window T of 4 hours, or equivalently a window size of 120 data packets. Having chosen T, we calculate the average link loss rate as follows. The instantaneous link loss rates in Sensorscope are estimated using the tinyOS default window mean with exponential weighted moving average (WMEWMA) estimator [17] and are reported to the sink every 15 minutes. Note here that the analyzed version of Sencorscope requires nodes to report their estimations of neighboring links to the sink. Our inference methods do not use this information and indeed our goal is to eliminate the need for such information. Assume that in the time window

W (of length T), we receive n instantaneous loss rates for link e_k. The average loss rate of e_k is the average of these reported loss rates. We calculate the path loss rates based on the number of packets received and the number of packets lost over the time window W: if r_i packets are received and f_i are lost on a path P_i, the loss rate on this path is: $f_i/(r_i + f_i) = f_i/T$.

B. Spatial Variation of Link Loss Rates

We start with the analysis of how link loss rates vary among the links at a given time. We use the data collected in Sensorscope over 10 days, from 26.12.2004 to 6.1.2005. In this 10-day period, we divide time into 60 time slots of 4 hours. Figure 2 plots the cumulative distribution (CDF) of link loss rates at 60 different time slots, each curve represents a time slot.

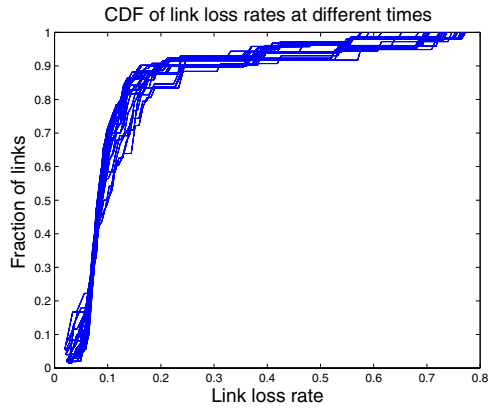

Fig. 2. Cumulative distribution (CDF) of link loss rates averaging over 120 packets (4 hours) intervals. There are 60 curves in the figure, each corresponding to a time window of 4 hours.

The figure shows that link loss rates follow an elbow-curve with the elbow in the interval $[0.1, 0.2]$. Specifically, more than 80% of links have loss rates rates below 0.2, and a non-negligible number (20%) of links have loss rates higher than 0.2. The observation of the elbow curves for link loss rates has not been made before in the literature but can be explained by the link characteristics and the routing algorithm as reported in [17], [18]. [17] and [18] report that the loss rate of a wireless link is either small or large, but is rarely in between (intermediate loss rates). Furthermore, links that have intermediate loss rates (links that are in the so-called "grey" region [17]) tend to be one-directional and are rarely used by the MintRoute routing algorithm of tinyOS, which gives priority to links with small and symmetric loss rates. The routing policy in tinyOS is to route data on links with low loss rates. A bad link is used only when there is no better alternative. The identification of these few bad links can thus help improve the network performance.

C. Temporal Variation of Link Loss Rates

We now turn our attention to how a link loss rate varies with time. Over the 10-day period, 90 links were used to route data. Similarly to Section III-B, we divide the 10-day period into 60 time slots of 4-hours (120 data packets) each. In each time slot, we calculate the average loss rate for all links that are used to transmit data in that time window. We plot in Figure 3 the loss rates of these 90 links in the time slots that they are used to route data. For each link, we plot the maximum, average and minimum loss rates.

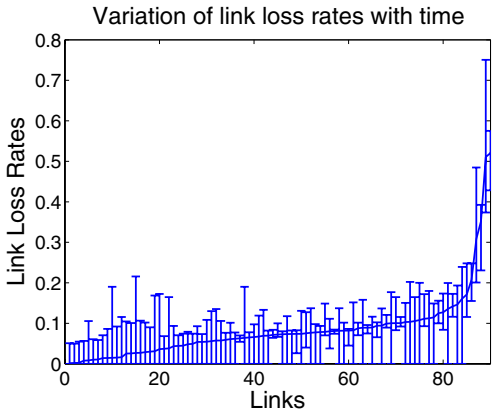

Fig. 3. Variation of link loss rates with time. There are 90 links and 60 time slots. The 90 links are ordered from 1 to 90 in increasing order of their loss rates averaging over the 60 time slots. Each vertical bar shows the maximum (the upper bound), average and minimum (the lower bound) loss rates for each link.

Confirming our observation of Section III-B, the majority of links (about 80 links) have loss rates consistently below 0.2, among them about 50 links have loss rates below 0.1. Furthermore, all these links with low loss rates (below 0.2) have loss rates that vary in a small range. A small fraction of links have loss rates consistently higher than 0.2 and the loss rates of these links are spread out over a larger range than the loss rates of links with low loss rates. Figure 3 shows that links either have consistently low or high loss rates, only 4 links (links number 14, 83, 84 and 85) out of 90 have loss rates fluctuating between the two categories. Remember that we only analyze links that appear in the routing trees, which are presumably the best available links. Our observation is consistent with the observations in [17] and [18]: without external physical influences, loss rates of a wireless link are stable. In an indoor environment such as the environment of Sensorscope, random physical influences are often short-lived and do not greatly affect the link loss rates averaging over long periods of 4 hours as in our analysis.

We also investigate the correlation between loss rates on different links. We calculate the 0.01 level significant 2-tailed Pearson Correlation for all pairs of links. We observe that links physically far apart are not correlated (most of them have a correlation coefficient less than 0.1), as we can expect. More surprisingly, neighboring links (links that share one common end node) also appear to be quite uncorrelated. Note here that the loss rates are average loss rates over intervals of 4 hours (120 packets); over these long intervals, correlated influences have a small effect on link loss rates. We conclude that the loss rates of links in Sensorscope are not far from

being independent.

D. Stability of End-to-End Paths

In this section, we investigate the stability of end-to-end paths that are used by the sensor nodes to transport data to the sink. It is well documented in [1] and [2] that, even in static networks, nodes frequently change their parents because of fluctuation in link quality and node batteries dying out. Previous results in [1] show that 80% of packets in their network are delivered by less than 20% of all links. We want to investigate whether the 80-20 behavior for links also applies to end-to-end paths in Sensorscope. We do this by looking at the lifetime of paths in Sensorscope and the number of packets delivered through the specific paths. Path lifetime is defined as the number of packets successfully delivered through the path. Figure 4 shows the CDFs of both path lifetimes and of the number of packets delivered by each path.

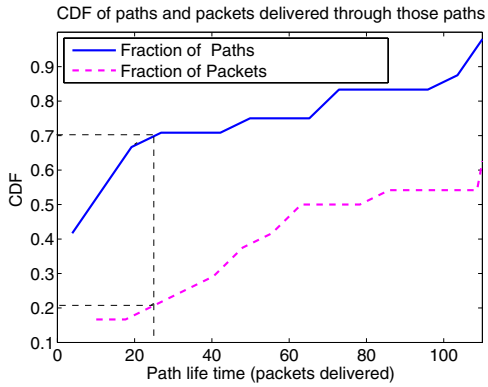

Fig. 4. CDF of path life times and of packets delivered through those paths. Long-lived, stable paths (one that delivered more than 25 packets and are on the right side of the vertical dashed line) constitute almost 30% of all the paths, yet they deliver more than 78% of the total packets.

Figure 4 shows that 70% of the paths are used to deliver less than 25 packets. We call these paths short-lived paths. Less than a third of the paths are stable (deliver more than 25 packets). The graph (the part on the right side of the vertical dashed line) shows the 78-30 behavior: 78% of the packets are delivered by only 30% of paths. Our observation is consistent with that of [1] (with the same hardware) for link lifetime. Therefore, even though sensor nodes change their end-to-end paths frequently, a few dominant paths are used to transporting most of the data.

E. Summary

The findings in this section suggest that: (i) In sensor networks, links are separated into two categories: good links with consistently low loss rates, and bad links with consistently high loss rates. With Sensorscope, the value 0.2 appears as a clear threshold to separate good and bad links. (ii) It is reasonable to perform inference based on end-to-end packet loss information gathered over a few end-to-end paths because a link remains good or bad for a long period of time and a few dominant paths are used to transport most of the data.

Our findings are based on the data collected in Sensorscope, and are consistent with or can be explained by findings in other experiments in the literature. We expect that the conclusions in this section hold for other sensor networks that use the same hardware and routing mechanisms.

IV. Network Model and Problem Settings

In this section, we develop the network model and define the problem of identifying lossy links based on the observations made in Section III. As noted in Section II, most of the existing works on estimating the loss rates of network links have been based on the assumption that the network uses one fixed routing tree. In contrast, our goal here is to study the inference of lossy links where the routing topologies change frequently.

A. Network Topology

We consider a static sensor network with a single sink (as in Sensorscope). We collect and analyze data that arrive at the sink in a time window W of length T. At any given instant, the sensor nodes form a tree to route data to the sink (as depicted in Figure 5). The construction of such a routing tree is described in [17]. Empirical studies in Section III-D have shown that in sensor networks, the routing trees change frequently. To accommodate routing changes, we divide the time window W into small time slots of equal length T_R, which is the time between routing updates (The window is made of 48 intervals of $T_R = 5$ minutes in the case of Sensorscope). We call these slots the *routing time slots* and we assume that the routing tree remains the same in each time slot. We aggregate the routing trees that appear in the time window W to obtain a forest of trees that have the sink as the common root. Let $\mathcal{P}$ be the set of all end-to-end paths in this forest that are used to route data. Denote by $\mathcal{S}$ the set of all sensor nodes (nodes that are involved in collecting or routing the data) and by $\mathcal{E}$ the set of links that appear in $\mathcal{P}$. We use a matrix R of dimension $|\mathcal{P}| \times |\mathcal{E}|$, called the routing matrix, to represent the information relating paths and links. Each row (respectively, column) of R represents a path in $\mathcal{P}$ (respectively, link). The entry R_{ij} is 1 if the link e_j is in the path P_i. In Sensorscope, we obtain R from the routing reports sent to the sink from the sensor nodes every 5 minutes. The routing matrix for the trees in Figure 5 can be constructed as follows.

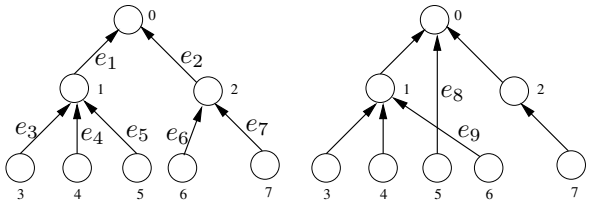

Fig. 5. Two routing trees are used to route data in a time window W. The set of network paths consists of all the paths of the two trees and the set of links consists of all the links appearing in either trees.

The aggregated routing topology contains 7 end-to-end paths: $P_1 : 3-1-0, P_2 : 4-1-0, P_3 : 5-1-0, P_4 : 6-2-0, P_5 : 7-2-0, P_6 : 6-1-0$, and $P_7 : 5-0$; and 9 directed links: e_1 to e_9 as shown in the figures, with the routing matrix

$$R = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

The routing matrix R contains therefore the information of links belonging to all the paths in the forest, and not only in a single tree. This enables us to accommodate topology changes.

B. Performance Model

Our performance model is as follows. During each routing time slot T_R, we passively observe traffic from the nodes to the sink to determine the number of packets arrived at the sink from each sensor node. Based on the sampling rate of the sensors, we also know how much data should be transmitted to the sink during this time interval and hence the number of packets that were lost on the path. (This assumption of a fixed data reporting rate can be easily eliminated by using the sequence number of packets). We assume that packets traverse a link e_k independently of other links. This assumption holds for wireless links as observed in [17]. When traversing a link e_k, each packet is lost with a probability $1 - \phi_{e_k}$, with ϕ_{e_k} being the average transmission rate of the link. If the path P_i comprises m links $e_1, ..., e_m$, the transmission rate of the path $P_i = \{e_1, ..., e_m\}$, denoted by ϕ_i, is given by $\phi_i = \prod_{k=1}^{m} \phi_{e_k}$.

Knowing the end-to-end packet transmission rates usually is not sufficient enough to calculate the transmission rate on each network link. Each path P_i gives an equation relating the path transmission rate and the link transmission rates as $\phi_i = \prod_{k=1}^{m} \phi_{e_k}$. Given $n_p = |\mathcal{P}|$ paths and $n_e = |\mathcal{E}|$ links, we have at most n_p constraints defined over n_e variables. The solution is not unique for this set of constraints if $n_p < n_e$, which is often the case (with Sensorscope, typically, $n_p = 30$ and $n_e = 50$ for a time window of 4 hours). The non-uniqueness of link loss rates is illustrated in the example of Figure 6 [13].

We are not interested in the exact loss rate of each link, we are only interested in detecting links with high loss rates. To this end, we use a threshold t_l, the link threshold, to determine whether a link e_k is good ($\phi_{e_k} \geq t_l$); or bad (lossy) ($\phi_{e_k} < t_l$). The threshold t_l can be set either to meet a given transmission rate target, or on the basis of data history that shows a clear value separating good and bad performing links. From the analysis of Section III we adopt the latter approach with $t_p = 0.8$ in Sensorscope. We evaluate other values of t_p for different loss models in Section VI.

The problem of identifying bad links without finding exact link loss rates amounts to finding the most probable solution for the observed end-to-end data. Knowing that bad links are not frequent, the most probable solution is the one giving the least number of lossy links. Let us consider the example in Figure 6. Under the loss model in Section III, where the threshold separating good and bad links is 0.8, if the end-to-end transmission rates to the sink of both nodes 2 and 3 are below 0.64 ($= 0.8 \times 0.8$), the most probable explanation is that link 1-0 is lossy (having transmission rate less than 0.8). Other explanations require at least two links to be lossy and are much less likely.

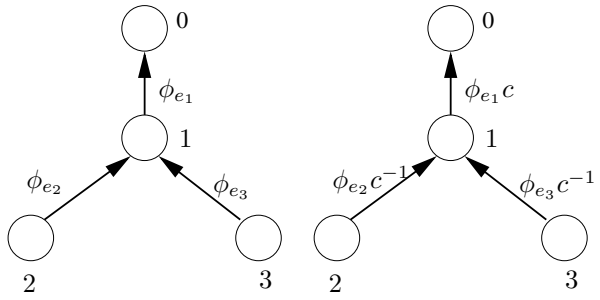

Fig. 6. In the figure, ϕ_{e_k} denotes the transmission rate of link e_k and c is a constant between $\max\{\phi_{e_2}, \phi_{e_3}\}$ and $1/\phi_{e_1}$. Both set of link transmission rates give the same end-to-end transmission rates. Link transmission rates therefore cannot be exclusively calculated from end-to-end transmission rates. Using end-to-end data, however, one can infer the links with high loss rates.

C. Problem Settings

The lossy link inference (LLI) problem is defined as follows. We are given the following information:

- The set of wireless links $\mathcal{E} = \{e_1, ..., e_{n_e}\}$;
- The set of paths $\mathcal{P} = \{P_1, ..., P_{n_p}\}$;
- The set of end-to-end data $\mathcal{D} = \{D_1, ..., D_{n_p}\}$ where each entry D_i contains the number of successful packet transmissions (r_i) and the number of failed transmissions (f_i) for each path P_i, i.e., $D_i = \{r_i, f_i\}$;
- The routing matrix R of the network, as defined in Section IV-A.
- A threshold t_l to separate good and bad links. A link is good if $\phi_{e_k} \geq t_l$.

The LLI problem is to find the most probable candidate bad links $\mathcal{X} \subseteq \mathcal{E}$ that are consistent with the outcomes of the end-to-end data deliveries. For simplicity, we define an indicator vector $\underline{x}$ of size $n_e = |\mathcal{E}|$, where $x_k = 1$ if the link $e_k \in \mathcal{X}$ ($\phi_{e_k} < t_l$); and $x_k = 0$ otherwise. The LLI problem can be formulated as finding

$$\underset{\mathcal{X} \subseteq \mathcal{E}}{\operatorname{argmax}} \, \mathbb{P}(\mathcal{X}|\mathcal{D}). \tag{1}$$

In addition to the above information, our inference techniques also make use of the following assumptions:

- Lossy links are rare, that is, the probability of a link to be lossy is much less than 0.5;
- Links have the same probability p of being lossy;
- The event that a link is lossy is independent of events on other links.

All of the above assumptions are supported by our analysis of Sensorscope data in Section III.

In the remaining sections of the paper, we first propose a fast and efficient set cover heuristic (named LLIS) that uses end-to-end transmission rates to solve the LLI problem in (1). When the end-to-end transmission rates can not be calculated accurately, we adapt the Bayesian inference technique [12] to solve the LLI problem.

V. Inference Techniques

A. Set-Cover Inference (LLIS Algorithm)

We begin with a fast algorithm to solve the LLI problem in (1). As mentioned in Section IV-B, link transmission rates and path transmission rates are related by an under-dimensioned system of equations. Since there are many solutions for such an under-dimensioned system of equations, there are also multiple solutions for the questions of whether transmission rates of links are below or above a threshold. The set-cover technique relies explicitly on the assumption that bad links are rare, and tries to find the solution with the least number of bad links. The ideas are as follows. From the observed data D_i for each path P_i, we calculate the observed transmission rate on the path P_i as $\phi_i = r_i/(r_i + f_i)$.

We define a path to be lossy if at least one of the links in the path is lossy, otherwise the path is lossless. The set cover approach begins by specifying a threshold t_p, the path threshold, to determine whether a path P_i is good (lossless); $\phi_i \geq t_p$ or bad (lossy); $\phi_i < t_p$. Note here that the path threshold t_p is an artificial threshold and is used purely to facilitate the lossy link inference problem. A path is called false positive if the path is actually good but we identify it as bad. A path is called false negative if the path is bad but we identify it as good. For each choice of t_p, each path has a false positive probability of p_p and a false negative probability of p_n as explained in Figure 7.

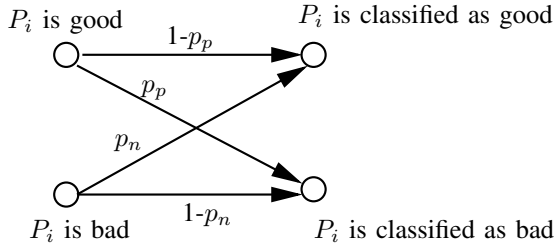

Fig. 7. A path is called false positive if the path is actually good but we identify it as bad. A path is called false negative if the path is bad but we identify it as good.

If we set $t_p = t_l$, any path P_i containing a bad link ($\phi_{e_k} < t_l$ for some $e_k \in P_i$) will have a transmission rate below t_p ($\phi_i = \prod_{e_k \in P_i} \phi_{e_k} \leq t_l = t_p$). Hence, there cannot be a false negative, and $p_n = 0$. Furthermore, if we set $t_p = t_l^m$, where m is the number of links in the path, then $p_p = 0$ because if a path has a transmission rate smaller than t_l^m then at least one of its links must have a transmission rate less than t_l. Without knowing the distribution of link loss rates, an optimal choice for t_p cannot be determined analytically. In this paper, unless otherwise stated, we choose an intermediate value $t_p = (t_l + t_l^m)/2$ to obtain small values of p_p and p_n.

We denote $\mathcal{P}_G$ and $\mathcal{P}_B$ as the sets of good and bad paths from $\mathcal{D}$ with a given path threshold t_p. We define the domain, $Domain(e_k)$, of a link e_k as the set of paths that contain e_k. The LLI problem in (1) now can be viewed as

$$\underset{\mathcal{X} \subseteq \mathcal{E}}{\operatorname{argmax}} \mathbb{P}(\mathcal{X}|\mathcal{P}_G, \mathcal{P}_B) \tag{2}$$

where $\mathbb{P}(\mathcal{X}|\mathcal{P}_G, \mathcal{P}_B)$ is the probability that $\mathcal{X}$ is the set of bad links given the data observations $\mathcal{D}$ (i.e., $\mathcal{P}_G$ and $\mathcal{P}_B$).

From Bayes' rule,

$$\mathbb{P}(\mathcal{X}|\mathcal{P}_G, \mathcal{P}_B) = \mathbb{P}(\mathcal{P}_G, \mathcal{P}_B|\mathcal{X})\mathbb{P}(\mathcal{X})/\mathbb{P}(\mathcal{P}_G, \mathcal{P}_B).$$

As $\mathbb{P}(\mathcal{P}_G, \mathcal{P}_B)$ only depends on $\mathcal{D}$, and not on the choice of $\mathcal{X}$, we are left with the equivalent maximization problem

$$\underset{\mathcal{X} \subseteq \mathcal{E}_c}{\operatorname{argmax}} \mathbb{P}(\mathcal{P}_G, \mathcal{P}_B|\mathcal{X})\mathbb{P}(\mathcal{X}) = \underset{\mathcal{X} \subseteq \mathcal{E}_c}{\operatorname{argmax}} \mathbb{P}(\mathcal{P}_G|\mathcal{X})\mathbb{P}(\mathcal{P}_B|\mathcal{X})\mathbb{P}(\mathcal{X}), \tag{3}$$

where $\mathbb{P}(\mathcal{X})$ is the probability that $\mathcal{X}$ is the set of bad links, which reads

$$\mathbb{P}(\mathcal{X}) = \prod_{k=1}^{n_e} p^{x_k}(1-p)^{(1-x_k)}, \tag{4}$$

and where $\mathbb{P}(\mathcal{P}_G|\mathcal{X})$ (resp, $\mathbb{P}(\mathcal{P}_B|\mathcal{X})$) is the probability that all paths in $\mathcal{P}_G$ (resp, $\mathcal{P}_B$) are good (resp, bad), given the set $\mathcal{X}$ of bad links. Making the reasonable assumption that the qualities of paths are independent of each other if we know the quality of all of their constituent links, we have:

$$\mathbb{P}(\mathcal{P}_G|\mathcal{X}) = \prod_{P_i \in \mathcal{P}_G} \mathbb{P}(P_i \text{ is good}|\mathcal{X}), \tag{5}$$

$$\mathbb{P}(\mathcal{P}_B|\mathcal{X}) = \prod_{P_j \in \mathcal{P}_B} \mathbb{P}(P_j \text{ is bad}|\mathcal{X}). \tag{6}$$

Let us denote by $\mathcal{P}_B^0(\mathcal{X})$ the set of bad paths when $\mathcal{X}$ is the set of bad links in the ideal case without corrupted paths: $\mathcal{P}_B^0(\mathcal{X}) = \bigcup_{e \in \mathcal{X}} Domain(e)$. The set of false positive paths when all links of $\mathcal{X}$ are diagnosed as bad is given by $\mathcal{A}_F(\mathcal{X}) = \mathcal{P}_B \backslash \mathcal{P}_B^0(\mathcal{X})$. Similarly, the set of false negative paths when all links of $\mathcal{X}$ are diagnosed as bad is given by $\mathcal{A}_M(\mathcal{X}) = \mathcal{P}_G \cap \mathcal{P}_B^0(\mathcal{X})$. Now $\mathbb{P}(\mathcal{P}_G|\mathcal{X})$ becomes:

$$\begin{aligned} \mathbb{P}(\mathcal{P}_G|\mathcal{X}) &= \prod_{P_i \in \mathcal{P}_G} \mathbb{P}(P_i \text{ is good}|\mathcal{X}) \\ &= p_p^{|\mathcal{A}_F(\mathcal{X})|}(1-p_n)^{|\mathcal{P}_G| - |\mathcal{A}_F(\mathcal{X})|} \end{aligned}$$

$$\begin{aligned} \mathbb{P}(\mathcal{P}_B|\mathcal{X}) &= \prod_{P_i \in \mathcal{P}_B} \mathbb{P}(P_i \text{ is bad}|\mathcal{X}) \\ &= p_n^{|\mathcal{A}_M(\mathcal{X})|}(1-p_p)^{|\mathcal{P}_B| - |\mathcal{A}_M(\mathcal{X})|}. \end{aligned}$$

Note that $\mathbb{P}(\mathcal{P}_G|\mathcal{X}) = \mathbb{P}(\mathcal{P}_B|\mathcal{X}) = 1$ if $p_n = p_p = 0$. Taking the logarithm of (3) with the values of $\mathbb{P}(\mathcal{P}_G|\mathcal{X})$ and $\mathbb{P}(\mathcal{P}_B|\mathcal{X})$ given above, and eliminating the constant terms, we obtain the following problem:

$$\underset{\mathcal{X}\subseteq\mathcal{E}}{\text{argmax}} \quad \Big[\quad |\mathcal{X}| \log \frac{p}{1-p} + |\mathcal{A}_F(\mathcal{X})| \log \frac{p_p}{1-p_n} + \quad |\mathcal{A}_M(\mathcal{X})| \log \frac{p_n}{1-p_p} \Big]. \tag{7}$$

We call the above maximization problem the Lossy Link Inference using Set-Cover (LLIS) problem. Since the probabilities p, p_n and p_t are unknown, the maximization problem in (7) cannot be solved as is and requires some approximations. We propose the heuristic to first minimize the number of corrupted paths $|\mathcal{A}_M(\mathcal{X})| + |\mathcal{A}_F(\mathcal{X})|$ and next minimize the number of lossy links with the corrupted paths cleaned. The LLIS algorithm below contains two steps. In the first step, it iteratively chooses at each iteration a link whose domain contains: (i) more bad paths than good paths and (ii) the smallest number of good paths (compared with all other links). The output of the first step is a set of links $\mathcal{X}_{EC}$, which approximately minimizes $|\mathcal{A}_M(\mathcal{X})|+|\mathcal{A}_F(\mathcal{X})|$. The algorithm then corrects the false negative paths $\mathcal{A}_M(\mathcal{X})$ by turning them into bad paths, and the false positive paths $\mathcal{A}_F(\mathcal{X})$ by turning them into good paths. In the second step, the LLIS algorithm tries to minimize the number of lossy links $|\mathcal{X}|$ with the cleaned set of paths by iteratively choosing at each iteration a link whose domain contains the largest number of bad paths. The detailed algorithm is as follows. The algorithm uses auxiliary set variables $\mathcal{Q}_B$, $\mathcal{Q}_G$, $\mathcal{X}_{EC}$ and $\mathcal{P}_B^C$.

The LLIS Algorithm

Minimize Detection Errors

- *Step 1*:
  1) Initialize $\mathcal{X}_{EC}$ to an empty set: $\mathcal{X}_{EC} = \emptyset$, and $\mathcal{Q}_G = \mathcal{P}_G$, $\mathcal{Q}_B = \mathcal{P}_B$.
  2) Find e that minimizes $|\mathcal{Q}_G \cap Domain(e)|$.
- *Step 2*: While $|\mathcal{Q}_B \cap Domain(e)| - |\mathcal{Q}_G \cap Domain(e)| \geq 0$
  1) Add e to $\mathcal{X}_{EC}$: $\mathcal{X}_{EC} = \mathcal{X}_{EC} \cup \{e\}$.
  2) Update the sets: $\mathcal{Q}_B := \mathcal{Q}_B \setminus Domain(e)$, $\mathcal{Q}_G := \mathcal{Q}_G \setminus Domain(e)$ and $Domain(e_k) := Domain(e_k) \setminus Domain(e)$ for all $e_k \in \mathcal{E}_c$.
  3) Find e that minimizes $|\mathcal{Q}_G \cap Domain(e)|$.
- *Step 3*: Calculate the inferred set of good and bad paths: $\mathcal{P}_{\mathcal{B}}^{\mathcal{C}} = \{\mathcal{P}_{\mathcal{B}} \setminus \mathcal{A}_F(\mathcal{X}_{EC})\} \cup \mathcal{A}_M(\mathcal{X}_{EC})$ and $\mathcal{P}_G^C = \mathcal{P} \setminus \mathcal{P}_{\mathcal{B}}^{\mathcal{C}}$.

Minimize the Number of Lossy Links

- *Step 4*: Initialize $\mathcal{X}$ to an empty set, i.e., $x_k = 0$ for all $1 \leq k \leq |\mathcal{E}_A|$, and $\mathcal{Q}_B = \mathcal{P}_B^C$.
- *Step 5*: While $\mathcal{Q}_B \neq \emptyset$
  1) Find a link $e_k \in \mathcal{E}_A$ that minimizes $|\mathcal{Q}_B \setminus Domain(e_k)|$.
  2) Add e_k to the solution $\mathcal{X}$: $\mathcal{X} := \mathcal{X} \cup \{e_k\}$, set $x_k = 1$.
  3) Update the sets: $\mathcal{Q}_B = \mathcal{Q}_B \setminus Domain(e_k)$ and $Domain(e_j) = Domain(e_j) \setminus Domain(e_k)$ for all $e_j \in \mathcal{E}_A$.
- *Step 6*: Output $\mathcal{X}$.

The LLIS algorithm has the advantage of being simple, but it is sensitive to estimation errors of end-to-end transmission rates and the choice of the path threshold t_p. The end-to-end transmission rates are only accurate when we have a sufficiently large number of packets. To handle the cases where there are not sufficient data to calculate the end-to-end transmission rates, we propose to adapt the second technique, namely the Bayesian inference technique [12], which is less vulnerable to end-to-end loss rates but also much more complex.

B. Bayesian Inference (MCMC Algorithm)

In this section, we model the lossy link inference problem as a Bayesian inference problem. The idea here is to try to generate a set of possible link transmission rates that can explain the observations of end-to-end data. This technique has been used to identify lossy links in IP networks [12]. We adapt it here to solve the lossy link inference problem in sensor networks. We begin by presenting some brief background information; for details, please refer to [21].

Denote by Φ_k the random variable that represents the transmission rate of link e_k and Φ as the multivariate random variable $\Phi = \{\Phi_1, \Phi_2, ..., \Phi_{n_e}\}$. The goal of Bayesian inference is first to determine the posterior distribution, $\mathbb{P}(\Phi|\mathcal{D})$, of Φ given the observed data, $\mathcal{D}$. The notation $\mathbb{P}(\Phi|\mathcal{D})$ is a short hand for $\mathbb{P}(\Phi = \phi|\mathcal{D})$ where $\phi = \{\phi_{e_1}, \phi_{e_2}, ..., \phi_{e_{n_e}}\}$ is a specific realization of the link transmission rates. Knowing $\mathbb{P}(\Phi|\mathcal{D})$, we can draw samples from this distribution where each sample is a vector containing the transmission rates for all the links in the network that can explain the observed data. We then collect the transmission rates of each link in all samples and compare them with the threshold t_l. If the majority of the sampled transmission rates of a link are bad ($< t_l$) then the link is declared as bad. Otherwise it is declared good.

In general, it is hard to compute $\mathbb{P}(\Phi|\mathcal{D})$ directly due to the complex integrations when Φ is a vector. It is also hard to obtain samples of the distribution $\mathbb{P}(\Phi|\mathcal{D})$.

An indirect approach to obtain the samples from $\mathbb{P}(\Phi|\mathcal{D})$ is to construct a Markov chain whose stationary distribution is exactly equal to $\mathbb{P}(\Phi|\mathcal{D})$. When such a Markov chain is run for a sufficiently large number of steps, it converges to its stationary distribution. We can then obtain samples from this stationary distribution and view the samples as samples from the posterior distribution $\mathbb{P}(\Phi|\mathcal{D})$. This way, we do not have to determine the distribution $\mathbb{P}(\Phi|\mathcal{D})$ and then draw the samples from it. This method is called the Markov Chain Monte Carlo (MCMC) simulation method.

To construct the Markov chain whose stationary distribution matches $\mathbb{P}(\Phi|\mathcal{D})$, Gibbs sampling [21] is commonly used. The basic idea of Gibbs sampling is that at each transition of the Markov chain, only a single variable (i.e., only one component of the Φ) is varied. In the remaining of this section, we will

show how Gibbs sampling can be used in our specific lossy link inference problem.

Remember here that each entry D_i in the observed data set $\mathcal{D}$ contains the number of successful packet transmissions (r_j) and the number of failed (i.e., lost) transmissions (f_j) on path P_i. Also remember that $\Phi = \{\Phi_1, ..., \Phi_{n_e}\}$ is the multivariate random variable that represents link transmission rates. The objective of the lossy link inference problem is to draw samples from the posterior distribution $\mathbb{P}(\Phi|\mathcal{D})$ that can be computed using Bayes' rule:

$$\mathbb{P}(\Phi = \phi|\mathcal{D}) = \frac{\mathbb{P}(\Phi = \phi)\mathbb{P}(\mathcal{D}|\Phi = \phi)}{\int_{\Phi} \mathbb{P}(\Phi = \phi)\mathbb{P}(\mathcal{D}|\Phi = \phi)\mathrm{d}\phi}. \quad (8)$$

Let $\phi = \{\phi_{e_1}, \phi_{e_2}, ..., \phi_{e_{n_e}}\}$ be a vector of specific loss rates of links in the network. To calculate (8), we need to have $\mathbb{P}(\Phi)$ and $\mathbb{P}(\mathcal{D}|\Phi = \phi)$ for all possible values of ϕ.

$\mathbb{P}(\Phi)$ is the prior knowledge about the lossiness of the links. Since we do not have reasons to privilege any links over any others, so we assume that $\mathbb{P}(\Phi)$ is uniform (by default of any better assumption).

The likelihood $\mathbb{P}(\mathcal{D}|\Phi = \phi)$ is the probability that we observe the data $\mathcal{D}$ given the link loss rates ϕ, and is given by

$$\mathbb{P}(\mathcal{D}|\Phi = \phi) = \prod_{P_i \in \mathcal{P}} \phi_i^{r_i}(1 - \phi_i)^{f_i} \quad (9)$$

where $\phi_i = \prod_{e_k \in P_i} \phi_{e_k}$ is the transmission rate for path P_i.

Now to obtain samples from $\mathbb{P}(\Phi|\mathcal{D})$ and to make inference decisions from the samples, we use MCMC with Gibbs sampling as in the following algorithm. We call the algorithm the MCMC algorithm.

The MCMC algorithm starts with an arbitrary initial assignment of link transmission rates, $\Phi = \phi$, i.e., $\{\Phi_1 = \phi_{e_1}, ..., \Phi_{n_e} = \phi_{e_{n_e}}\}$.

The algorithm then repeatedly performs the following procedure. It sequentially loops through all links. For each link e_k, the algorithm performs the following two steps. In the first step, it computes the posterior distribution of the transmission rate for that link e_k alone conditioned on the observed data $\mathcal{D}$ and the transmission rates assigned to all other links. Let $\overline{\phi}_{e_k} = \cup_{j \neq k} \phi_{e_j}$, the posterior distribution for link e_k is

$$\mathbb{P}(\Phi_{e_k}|\mathcal{D}, \overline{\Phi}_{e_k}) = \frac{\mathbb{P}(\mathcal{D}|\{\Phi_{e_k}\}, \{\overline{\Phi}_{e_k}\})\mathbb{P}(\Phi_{e_k})}{\int_{\phi_{e_k}} \mathbb{P}(\mathcal{D}|\{\Phi_{e_k}\}, \{\overline{\Phi}_{e_k}\})\mathbb{P}(\Phi_{e_k})\mathrm{d}\phi_{e_k}}.$$

Since $\mathbb{P}(\Phi)$ is uniform, $\mathbb{P}(\Phi_{e_k})$ is the same for all ϕ_{e_k}. Furthermore, $\mathbb{P}(\mathcal{D}|\{\Phi_{e_k}\}, \{\overline{\Phi}_{e_k}\}) = \mathbb{P}(\mathcal{D}|\Phi)$, and thus

$$\mathbb{P}(\Phi_{e_k}|D, \overline{\Phi}_{e_k}) = \frac{\mathbb{P}(\mathcal{D}|\Phi)}{\int_{\Phi_{e_k}} \mathbb{P}(\mathcal{D}|\Phi)\mathrm{d}\phi_{e_k}}. \quad (10)$$

In the second step, using equation (10), the algorithm numerically computes the posterior distribution $\mathbb{P}(\Phi_{e_k} = \phi_{e_k}|D, \{\overline{\Phi}_{e_k}\})$ and draws a sample from this distribution to get a new value, $\phi^*_{e_k}$, for the transmission rate of link e_k.

The MCMC algorithm iterates the above procedure 1000 times. That is, it cycles through all links and assigns each link a new loss rate 1000 times. After the burn-in period of 500 iterations, it starts collecting sample transmission rates for the next 500 iterations. These 500 samples can be viewed as the samples from the distribution $P(\Phi|\mathcal{D})$. For each link e_k, if more than 250 samples of ϕ_{e_k} are smaller than t_l, then e_k is added to the solution set $\mathcal{X}$, which originally is empty.

Finally, the algorithm returns $\mathcal{X}$ as the solution for the lossy link inference problem.

The MCMC algorithm only requires the number of packets sent and received on each end-to-end path. It therefore can be used for end-to-end paths that do not carry enough data packets for the transmission rates to be calculated exactly. The MCMC algorithm however requires much more computation time than the LLIS algorithm.

VI. Evaluation

We verify our proposed algorithms using three levels of evaluation. First, we compare our algorithms with the existing algorithms that use passive traffic as a monitoring tool [7], [8] by simulations using the same settings (one fixed routing tree and correlated monitoring traffic). Second, we evaluate our algorithms in more realistic settings using simulations with changing routing trees and uncorrelated monitoring traffic. Finally, we test our algorithms on the real data collected in Sensorscope.

The performance of the algorithms are evaluated in terms of two metrics: the detection rate (DR), which is the percentage of links that are correctly diagnosed as bad, and the false positive detection rate (FPR), which is the percentage of links that are working correctly but are diagnosed as bad. With $\mathcal{F}$ denoting the set of the actual bad links, and $\mathcal{X}$ the set of links identified as bad by an inference algorithm, these two rates are given by:

$$\text{DR} = \frac{|\mathcal{F} \cap \mathcal{X}|}{|\mathcal{F}|} \text{ ; FPR} = \frac{|\mathcal{X} \backslash \mathcal{F}|}{|\mathcal{X}|}.$$

A. Comparison with Previous Algorithms

We begin our validation by comparing our algorithms with the existing algorithms of [7] and [8]. These algorithms cannot be directly compared with our algorithms because of the different assumptions (see section II-A.2) and objectives: we try only to infer whether a link loss rate is above or below a threshold, not the actual value of this loss rate as in [7] and [8]. But these algorithms can be compared in terms of the ability to detect lossy links. The network topologies are the random trees generated as in [7] and [8].

We use two different models for assigning loss rates to the links. In the first loss model (LM1), which was introduced in [7], good links have transmission rates of 0.99 and bad links have deterministic transmission rates of 0.75. For this loss model, we choose the link threshold $t_l = 0.8$. In the second loss model (LM2), which was introduced in [8], link transmission rates are drawn from a distribution with probability density function $f(\xi) = \lambda\xi^{(\lambda-1)}$, for $0 < \xi \leq 1$, parameterized by $\lambda > 1$. The expected value of this random variable is $\lambda/(1 + \lambda)$. In our simulations, we choose $\lambda = 4$

so that the expected link loss rate is 0.8, and we set $t_l = \lambda/(1+\lambda) = 0.8$. Once each link is assigned a loss rate, we use Bernoulli loss processes at each link for both LM1 and LM2. That is, each packet traversing a link is dropped with a fixed probability independently of the others. The end-to-end traffic is generated as in [7] and [8]: in each simulation, 200 packets are sent from each sensor node to the sink, in rounds. The results are shown in Table I.

We observe that our algorithms provide similar detection rates but much lower false positive rates. The explanation for the high false positive rates of the algorithms in [7], [8] is that they try to infer exact loss rates. Even though the estimation errors are small (with the mean absolute error in the order of 0.1), they are still large enough to make many inferred loss rates erroneously cross the threshold, which results in links being wrongly identified as lossy.

We note that the LLIS algorithm has a higher DR than the MCMC algorithm in LM1, and a smaller FPR for both loss models. The MCMC has higher FPR because it tries to find the most probable answer for the lossy link inference problem whereas the LLIS algorithm explicitly finds the least number of bad links.

We also observe that our algorithms are less accurate for LM2 than LM1. There are two reasons for this. First, in LM2, the transmission rates of good and bad links are contiguous (bad links have transmission rates in the interval $[0, 0.8]$, and good links have transmission rates in $[0.8, 1]$) whereas in LM1 transmission rates of bad links and good links are sufficiently far apart (0.75 and 0.99, respectively). The use of link threshold $t_l = 0.8$ to separate good and bad links in LM2 is therefore less accurate than in LM1. Second, the network sizes of the simulations for LM2 are much larger than for LM1.

B. Simulations with Different Network Settings

In this section, we evaluate the effect of changing topologies and non-correlated monitoring traffic on the performance of our algorithms. Our inference algorithms are the first solutions to the lossy link inference problem under these conditions.

Packets are delivered to the sink in rounds of data collection and are not aggregated before being forwarded upward in the routing trees. In all simulations of this section, 200 packets are sent from each sensor, i.e, there are 200 rounds of data collection. In each round, a routing tree is used to deliver data to the sink. After every r rounds of packet delivery, the routing tree changes. The overall routing topology is the aggregate of multiple routing trees with the same root node and the same number of nodes (v). To construct the routing matrix R, we update the routing trees after every 10 rounds of data delivery (i.e., the length of the routing time slot T_R is 10), as explained in Section IV-A. The branching ratio at each non-leaf node is randomly chosen between 1 and an upper bound of 10. A fraction of links (f) are classified as "bad" and the rest as "good". We vary the time between routing changes (r), the number of nodes (v), the fraction of bad links (f), and the link loss models in our simulations. We repeat each setting 5 times and report the average results.

1) Effects of Routing Changes: We begin with the evaluation of the effect of routings changes on the inference algorithms in Section V. The network size is kept at $v = 1000$ and the fraction of bad links is $f = 0.2$. The link loss model is LM1 with Bernoulli losses. The length of the routing time slot T_R is always kept equal to 10, but we vary the time between routing change r.

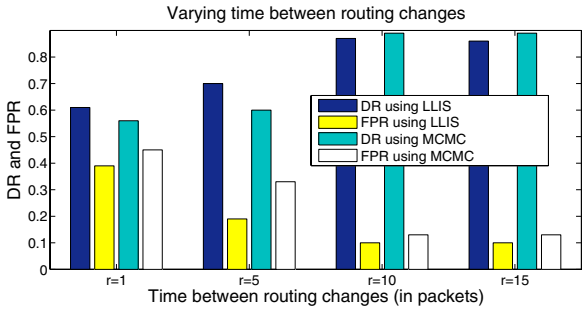

Fig. 8. Varying the time between routing changes, the network size is $v = 1000$, the fraction of lossy links is $f = 0.2$, and the loss model is LM1 with Bernoulli losses.

Figure 8 shows the results when r varies from 1 to 15. When the time between routing changes is longer than or equal to the length of the routing time slot ($r \geq T_R$), all the routing changes are captured, hence the numbers of packets delivered (r_i) and lost (f_i) calculated for each each path are correct. However, this is no longer the case when $r < T_R$. When routing changes are more frequent than routing updates, we are not able to exactly identify which paths the packets take to get to the sink and we can only assume that the packets take the same path between routing updates, which may not be true. Therefore, the numbers r_i and f_i for each path P_i may not be correct. As expected the smaller r, the worse the performance of the two algorithms. We also observe that the LLIS algorithm handles errors caused by frequent routing changes better than the MCMC algorithm because the former explicitly corrects the errors, whereas the latter does so only implicitly.

2) Effects of Topology Sizes: We present in this section simulation results for different number of nodes (v). The time between routing changes r is set to 10 and the loss model is LM1 with Bernoulli losses.

Figure 9 shows the simulations with different network sizes (v ranges from 100 to 3000). We observe that as the network size increases, both algorithms have high DR and low FPR but the accuracies of the two algorithms are slightly reduced.

3) Effects of the Number of Lossy Links: Figure 10 shows the simulation results in 1000-node topologies with f varying from 0.05 to 0.25. We note that as the fraction of bad links increases, the accuracy of both algorithms decreases. Both algorithms implicitly (the LLIS algorithm) or explicitly (the MCMC algorithm) rely on the assumption that bad links are rare, therefore as the fraction of bad links increases the assumption becomes weaker and the two algorithms perform

# of nodes	Loss model	EM	FG	LLIS	MCMC
9	LM1	DR = 99%, FPR = 50%	N/A	DR = 99%, FPR = 2%	DR = 95%, FPR = 7%
115	LM1	DR = 97%, FPR = 55%	N/A	DR = 98%, FPR = 5%	DR = 92%, FPR = 9%
5000	LM2	N/A	DR = 92%, FPR = 31%	DR = 89%, FPR = 7%	DR = 91%, FPR = 14%
10000	LM2	N/A	DR = 90%, FPR = 33%	DR = 85%, FPR = 10%	DR = 88%, FPR = 16%

TABLE I

COMPARISON BETWEEN THE TWO ALGORITHMS OF SECTION V AND THE ALGORITHMS IN [7] AND [8]. THE COLUMN UNDER "EM" (RESPECTIVELY, FG) REPRESENTS THE RESULTS OF THE EM ALGORITHM IN [7] (RESPECTIVELY, THE FACTOR GRAPH ALGORITHM IN [8]). THE "N/A" NOTATION INDICATES THAT WE DO NOT SIMULATE THE ALGORITHM OF [7] IN THE SETTING OF [8] AND VICE VERSA.

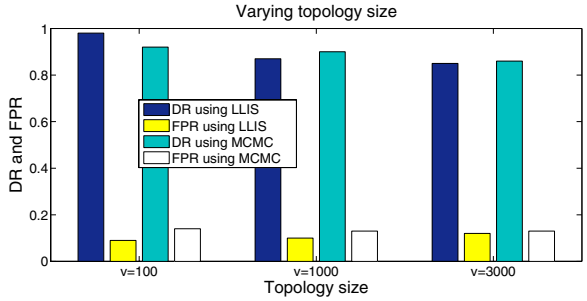

Fig. 9. Varying the number of nodes v, the fraction of lossy links $f = 0.2$, the time between routing change is $r = 10$ and the link loss model is LM1 with Bernoulli losses.

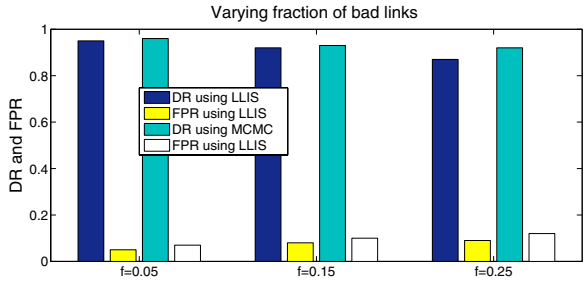

Fig. 10. Varying the fraction of lossy links. Networks have fixed size $v = 1000$, the time between routing change is $r = 10$ and the link loss model is LM1 with Bernoulli losses.

less accurately.

4) Effects of Loss Processes: We now evaluate the effect of loss processes on our algorithms. The network size is kept at $v = 1000$. In Section VI-A, we have shown the results of the LLIS and MCMC algorithms with Bernoulli losses. We want to investigate how our inference algorithms perform with other non i.i.d loss processes such as the correlated Gilbert losses, where links fluctuate between good and bad states. When in the good state, the link does not drop any packets, when in the bad state the link drops all packets. The transition between good and bad states are chosen so that the average loss rate matches the loss rate assigned to the link.

Figure 11 shows the results of the loss model LM2 with Bernoulli and Gilbert losses. We observe that our algorithms are quite insensitive to loss models. We attribute the reasons for the above observation to the fact that our algorithms only infer good and bad links based on average loss rates, and not the loss rates themselves.

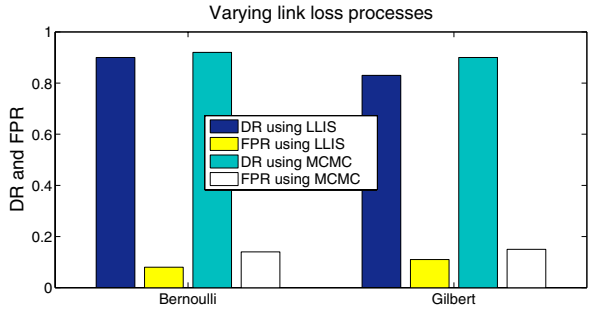

Fig. 11. Varying the loss processes, network size is $v = 1000$, the fraction of lossy links $f = 0.2$, the time between routing change is $r = 10$ and the loss model is LM2.

C. Experimental Results on Sensorscope

In this section, we evaluate our inference algorithms using the Sensorscope data. The validation approach we use is to compare our inference results with the real loss rates reported by the sensor nodes and to check for consistency in the inferences made by the two algorithms (LLIS and MCMC).

We present here the evaluation of the data trace collected in Sensorscope in December, 2004. We divide the time into 60 slots of 4 hours, as described in Section III. The set of true bad links $\mathcal{F}$ are determined using reports sent from sensor nodes. We run our algorithms on these 60 slots and report the average DR and FPR values. Many paths in Sensorscop are used to transport a small number of packets. These paths do not provide enough data to reliably calculate the end-to-end loss rates. We therefore only consider paths that delivered at least a threshold number of packets, t, which is set to 20, 60 and 100 in our evaluation. The choice of t is a trade-off between the number of links that can be covered and accuracy of the inference results. The larger t, the more accurate the inference results, but the smaller the coverage. We plot in Figure 12 the results of our algorithms.

A total of 50 links appear in the paths that deliver at least 20 packets over 60 time slots. The link estimations reported to the sink indicate that 12 out of these 50 links are lossy. The LLIS algorithm correctly identifies 7 lossy links and the MCMC algorithm correctly identifies 9 lossy links. The

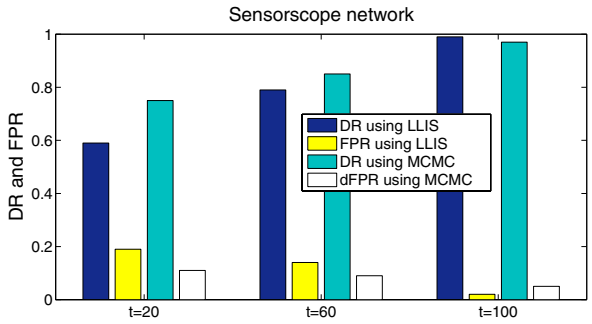

Fig. 12. Applying the inference algorithms to Sensorscope. The algorithms achieve high DR and low FPR.

MCMC algorithm also gives 1 false positive and the LLIS algorithm gives 2 false positives. The DR and FPR rates are plotted in Figure 12. We observe that the minimum trace length t must be large enough to obtain good detection rates, especially for the LLIS algorithm. An additional source of noise in the measurements for Sensorscope is the fact that the interval r between routing changes may be much smaller than the routing time slot T_R, which is set to 5 minutes. When $r < T_R$, there is noise in calculating the numbers r_i and f_i on each path P_i as we saw in the previous subsection.

When $t = 100$, both algorithms identify almost all bad links and have negligible false positive rates. However, as we can expect, the number of paths that deliver more than $t = 100$ packets is much smaller than when $t = 20$, hence the number of links appearing in these paths is also small (20 links).

Most of the lossy links are persistent. A bad link usually remains so in at least 2 time slots (i.e., 8 hours). More importantly, the lossy links identified by the two algorithms are consistent. All 7 lossy links identified by the LLIS algorithm when $t = 20$ are also the lossy links identified by the MCMC algorithm.

VII. Conclusions

In this paper, we have proposed an algorithm called the LLIS algorithm to infer lossy links using only end-to-end data in sensor networks. The LLIS algorithm is fast and accurate if there are enough data to calculate the end-to-end transmission rates accurately. When this is not the case, we adapt the second algorithm, the MCMC, that takes much longer time than the LLIS algorithm to solve the lossy link inference problem. We observe that the LLIS algorithm can handle routing changes better than the MCMC algorithm because it explicitly corrects errors caused by routing changes. Both algorithms perform better than existing passive monitoring methods in the literature, especially in terms of false positive rates. They also perform well in simulations and with real data collected from Sensorscope.

Our ongoing work is centered on making the inference algorithms real time on large sensor networks.

Acknowledgements

We are grateful to Henri Dubois-Ferriere and Thomas Schmid (EPFL) for providing and helping us understand the Sensorscope data. We also thank Maciej Kurant and Alaeddine El Fawal for feedback on the early drafts. This work is financially supported by grant DICS 1830 of the Hasler Foundation and by the Swiss NCCR "Self-Organizing Mobile Information and Communication Systems".

References

[1] R. Szewczyk, J. Polastre, A. Mainwaring, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *Proceedings of The Second ACM Conference on Embedded Networked Sensor Systems (Sensys)*, 2004.

[2] T. Schmid, H. Dubois-Ferriere, and M. Vetterli, "Sensorscope: Experiences with a wireless building monitoring," in *Proceedings of Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, 2005.

[3] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Proceedings of Second European Workshop on Wireless Sensor Networks (EWSN)*, 2005.

[4] J. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'02)*, May 2002.

[5] ——, "Computing aggregates for monitoring wireless sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[6] C. Hsin and M. Liu, "A distributed monitoring mechanism for wireless sensor networks," in *Proceedings of the ACM workshop on Wireless security*, Atlanta, GA, USA, April 26-27 2002, pp. 57 – 66.

[7] G. Hartl and B. Li, "Loss inference in wireless sensor networks based on data aggregation," in *Proceedings of the Third IEEE/ACM International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, April 26-27 2004, pp. 396–404.

[8] Y. Mao, F. R. Kschischang, B. Li, and S. Pasupathy, "A factor graph approach to link loss monitoring in wireless sensor networks," *IEEE JSAC, Special Issue on Self-Organizing Distributed Collaborative Sensor Networks*, April 2005.

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

[10] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," in *Proceedings of Mobile Ad Hoc and Sensor System (MASS 2004)*, 2004.

[11] M. Coates, A. Hero, R.Nowak, and B. Yu, "Internet tomography," *IEEE Signal Processing Magazine*, vol. 19, May 2002.

[12] V. N. Padmanabhan, L. Qiu, and H. J. Wang, "Server-based inference of internet performance," in *Proceedings of the IEEE INFOCOM'03*, San Francisco, CA, April 2003.

[13] N.Duffield, "Simple network perormance tomography," in *Proceedings of the IMC'03*, Miami Beach, Florida, October 2003.

[14] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: An experimental study of low-power wireless sensor networks," in *UCLA Computer Science Technical Report UCLA/CSD-TR 02-0013*, February 2002.

[15] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proceedings of Sensys 03*, Los Angeles, CA, USA, November 2003.

[16] A. Woo and D. Culler, "Evaluation of efficient link reliability estimators for low-power wireless networks," in *UCB Technical Report*, November 2003.

[17] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multhop routing in sensor networks," in *Proceedings of Sensys 03*, Los Angeles, CA, USA, November 2003.

[18] N. Reijers, G. Halkes, and K. Langendoen, "Link layer measurements in sensor networks," in *Proceedings of the Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[19] www.tinyOS.net.

[20] J. Polastre and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of Sensys 04*, 2004.

[21] M. A. Tanner, *Tools for Statistical Inference*. Springer-Verlag, 1991.