A Factor Graph Approach to Link Loss Monitoring in Wireless Sensor Networks

Yongyi Mao, *Member, IEEE*, Frank R. Kschischang, *Senior Member, IEEE*,
Baochun Li, *Member, IEEE*, and Subbarayan Pasupathy, *Fellow, IEEE*

***Abstract*— The highly stochastic nature of wireless environments makes it desirable to monitor link loss rates in wireless sensor networks. In a wireless sensor network, link loss monitoring is particularly supported by the *data aggregation* communication paradigm of network traffic: the data collecting node can infer link loss rates on all links in the network by exploiting whether packets from various sensors are received, and there is no need to actively inject probing packets for inference purposes. In this paper, we present a low complexity algorithmic framework for link loss monitoring based on the recent modelling and computational methodology of factor graphs [2]. The proposed algorithm iteratively updates the estimates of link losses upon receiving (or detecting the loss of) recently sent packets by the sensors. The algorithm exhibits good performance and scalability, and can be easily adapted to different statistical models of networking scenarios. In particular, due to its low complexity, the algorithm is particularly suitable as a long-term monitoring facility.**

***Index Terms*— sensor networks, link loss monitoring, network tomography, factor graphs, Sum-Product Algorithm**

I. Introduction

Recent technological advances have made it feasible to deploy large-scale sensor networks using low-cost sensor nodes. However, as the scale of sensor networks becomes larger, two key challenges potentially arise. First, *node failures*. Due to their inherent instability and energy constraints, sensor nodes are prone to failures. It would thus be useful to determine which set of nodes or which geographical areas within the network are experiencing high loss rates. Such information is potentially valuable to the design of fault-tolerant protocols or monitoring mechanisms, so that the problem areas may be re-deployed, and critical data may be rerouted to avoid these failure-prone areas suffering high loss rates. Second, *resource constraints*. Since sensor nodes have limited computational resources, any algorithms developed for wireless sensor networks must not rely on the assumption of unlimited resources, and must sparingly use the limited resources that do exist. Further, due to wireless characteristics such as fading and interference, wireless sensor networks are subject to stringent bandwidth resource constraints. One can not rely on the use of active acknowledgments — which are not scalable or bandwidth-efficient — in the design of sensor network algorithms.

Motivated by the needs (fault-tolerance and reliability) and constraints (bandwidth and computational power) illustrated above, in this paper, we concentrate on the problem of efficiently determining link loss rates in wireless sensor networks. Particularly, we attempt to efficiently determine link loss rates based on the *data aggregation* communication paradigm in sensor networks. Due to the obvious need of centralized sensor data processing and monitoring, the paradigm of data aggregation, also referred to as *data fusion*, has been critical to the effective operation of sensor networks. Work in this area has been previously presented (refer to [1] as an example) and continues to be actively researched. In the process of data aggregation, a subset of nodes in the network attempts to forward the sensor data they have collected back to a *center* (or *sink*) node via a *reverse multicast tree*.

More specifically, in the process of data aggregation, before a node sends its data to the next node in the path to the *sink*, it waits to receive data from all of its child nodes in the reverse multicast tree (or until a specified period of time has elapsed). The node then aggregates its own data with the data it has received from its child nodes, and forwards this aggregated data to the *sink* via the reverse multicast tree. Information about which nodes' data is present in the aggregated data must also be sent to the *sink*. Thus, data fusion saves communication overhead at the cost of additional computation and memory resources. Fig. 1 depicts a simple example of a sensor network using the data aggregation paradigm.

In wireline networks, the field of network inference, also referred to as network tomography, involves the estimation of network performance parameters using measurements. In wireline networks, inferring link losses requires either network multicast support (which is not always the case), or sending series of back-to-back packet pairs with unicast (see, e.g., [3], [4] and the references included therein). In the case of multicast-based link loss inference, a center node sends out a batch of multicast probing packets to all terminals and, upon receiving the acknowledgments from the terminals

The first author is with the School of Information Technology and Engineering, University of Ottawa, 800 King Edward Ave, Ottawa, Ontario, K1N 6N5, Canada; the other authors are with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario, M5S 3G4, Canada. E-mail: The first author yymao@site.uottawa.ca, the second and last author: {frank,pas}@comm.utoronto.ca; the third author: bli@eecg.utoronto.ca.

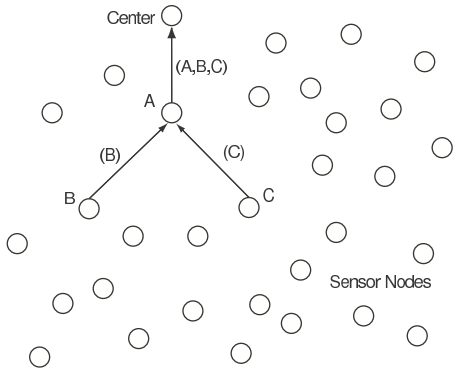

Fig. 1. Data aggregation (fusion) in wireless sensor networks: an example. Node B sends it data, (B), destined for the center node, to node A. Node C similarly sends its data, (C), destined for the center node, to node A. Node A then aggregates is own data, (A), with that of nodes B and C, and sends the fused data, (A,B,C) to the center node. With data aggregation, each node is only required to transmit once per data collection round. However, without data aggregation, node A would have to transmit three times per data collection round: once to send its own data to the center node, once to forward node B's data, and once to forward node C's data.

whether these probing packets are received, performs statistical inference on the link loss rates. In addition to requiring the multicast protocol to be supported by the network, such a strategy perturbs the network by sending out extra packets solely for the purpose of probing. Similar observations can also be made in the case of inference with unicast-based packet-pair measurements, though it does not require specific support from the network layer. Injecting additional traffic will further aggravate the link losses at the loaded links, which makes it impractical as a long-term monitoring daemon in sensor networks, given existing resource constraints.

In this paper, we propose and examine a new and efficient mechanism to monitor link losses in wireless sensor networks. In a wireless sensor network, a set of terminal (sensor) nodes send data, concerning some measurements of the physical world, to a center (data-collecting) node via a set of wirelessly connected links. We take advantage of the data aggregation communication paradigm in sensor networks, where the data-carrying traffic flows from the terminals to the center via a reverse multicast tree. Such a characteristic potentially enables the implementation of simple protocols and algorithms for constant link loss monitoring at virtually no cost. Our original contributions are two-fold: First, we present a novel algorithm for the purpose of link loss inference, based on the recent methodology of factor graphs and the Sum-Product Algorithm [2]. We show that this algorithm has very low complexity, and demonstrate by simulations its excellent performance and scalability. Second, we are one of the first to consider network inference exploiting reverse multicast trees for data aggregation in sensor networks. Most existing research in this area has dealt with the traditional multicast and unicast communication paradigms in wireline networks, where probes are sent from a single source to one or several receivers (see, e.g., [3], [4], [13]–[16], [19], [26]).

The remainder of the paper is organized as follows. We will first present the formulation of the problem in Section II, and introduce the algorithm in Section III. We will then give the simulation results and some discussion in Section IV and close this paper with a brief conclusion in Section V.

II. Model and Problem Formulation

We consider a sensor network as a directed graph, where each node represents either a terminal (sensor), a router, or the center (data collecting) node, each directed edge represents the link between these nodes, and the direction of an edge indicates the direction of the data flow on the link. Based on the data aggregation paradigm, we consider a reverse multicast tree rooted at the data collecting node, where messages are sent from leaf nodes to the data collecting node. We will not allow, except for the data collecting node, degree-two nodes in the graph; that is, if a degree-two node is not the data collecting node, it is suppressed in the graph. In this setting, what we refer to as a "link" is not necessarily in its physical sense, since a "link" can be a path consisting of several connected physical links as long as no other paths are branched from an intermediate node in the path. It can be verified that such a notion of "link" is defined without loss of generality, as far as loss rates are concerned. In this paper, the term "path" of a network refers to a path that starts from a terminal node and ends at the center node.

We assume that all terminal nodes send packets constantly, in a synchronous manner, to the center node along the tree (in Section IV, we will briefly discuss the possibility of relaxing this assumption). Each intermediate node in the tree, upon receiving the packets from its children, creates an aggregated packet and forwards it to its parent. Here the notion of "packet" is also more conceptual than implementational. For example, we will not require packets sent by different nodes to have the same size, and rather assume that the aggregated packet sent by any intermediate node is large enough to "bundle" all the information contained in its children's packets. Throughout this paper, a packet sent from a sensor is said to be "received" or to "have arrived" if the data contained in the packet is received by the data collecting node in the aggregated packet. As part of the transmission protocol, for every packet transmitted by a terminal node, the center node expects it to arrive within a certain time frame; and if the packet is not received within that time frame, then a packet loss is suggested to have occurred on one of the links along the path. Based on successive observations on whether packets from each terminal have arrived, the center node can infer the link loss rates on all links in the network.

Formally, we will use E to denote the set of links of the network of interest, and W to denote all the paths in the network. Associate to each link $e \in E$ a *state* x_e, taking

values from $\{0,1\}$; when link e is at state 0 ("bad state"), no packet will pass through e, and when link e is at state 1 ("good state"), all packets can pass through e. For each path $w \in W$, let its state x_w be the logic AND of all the links consisting of w, for which we write

$$x_w = \bigoplus_{e \in w} x_e,$$

where $\bigoplus$ denotes the logic AND operator and "$e \in w$" reads "e is a link contained in w". For example, in the toy example of a sensor network in Figure 2, there are three links a, b, and c, and two paths $\{a, b\}$ and $\{c, b\}$; and the links states and path states are related by $x_{\{a,b\}} = x_a \oplus x_b$, and $x_{\{c,b\}} = x_c \oplus x_b$. Clearly, if and only if when the state x_w of path w is 1 can packets pass through w. Then whether a packet will be received essentially indicates the state of the path along which the packet is to travel.

At any time instant, the state x_e of every link e can be regarded as a Bernoulli random variable with probability α_e taking value 1 and with probability $1 - \alpha_e$ taking value 0. In this paper, we use

$$B(x, \alpha) = \begin{cases} \alpha, & x = 1, \\ 1 - \alpha, & x = 0. \end{cases}$$

to denote the probability mass function of a Bernoulli random variable parametrized by α.

We will assume that α_e at each link is quasi-static, namely, over a relatively small time window in which hundreds or thousands of packets may be sent by any terminal, α_e stays as a constant. Suppose that during a time window, there are n *batches* of synchronized packets transmitted from each terminal to the center, where the i^{th} *batch* of packets are transmitted at time $t_i, i = 1, 2, \ldots, n$, from all the terminals synchronously. Let $x_e^{(i)}$ be the link state of e at time t_i, then the path state $x_w^{(i)}$ of path w at time t_i is

$$x_w^{(i)} = \bigoplus x_e^{(i)},$$

experienced by the packet transmitted at t_i and traveling along w; the observation whether the packets in the i^{th} batch have arrived indicates the states of all paths at time t_i.

Collectively, we denote $X_E^{(i)} := \{x_e^{(i)} : e \in E\}$, $X_W^{(i)} := \{x_w^{(i)} : w \in W\}$, $X_E^{(l,m)} := \{X_E^{(i)} : l \le i \le m\}$, $X_W^{(l,m)} := \{X_W^{(i)} : l \le i \le m\}$, and $\alpha_E := \{\alpha_e : e \in E\}$. The problem of link loss inference in the network is then the problem of estimating α_E based on the observation of $X_W^{(1,n)}$. In this paper, we set up the problem as, for each $e \in E$, finding $\hat{\alpha}_e$ that maximizes the posterior probability $P[\alpha_e | X_W^{(1,n)}]$ of α_e conditioned on the observation $X_W^{(1,n)}$. In addition, it is important to realize that α_e changes with time. Thus we ideally desire an algorithm to have a sufficiently low complexity so as to serve as a daemon tracking the link loss rates constantly.

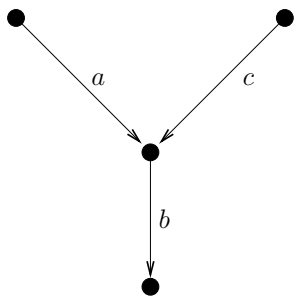

Fig. 2. A toy example of a sensor network.

III. The factor graph approach

Link loss inference belongs to the relatively recent area of networking research, network tomography (see [3], [4], [13]–[22], [26], [27] etc.). Typical network tomography problems include the inference of link loss or delay characteristics from end-to-end measurements [3], [13]–[19], [22], [26], [27], the estimation of origin-destination traffic intensities from link measurements [23]–[25], and the inference of network topology [20], [21].

Prior to this work, most of the literature on link loss inference concerns wireline networks, where multicast packets are sent actively to probe the network. When multicast addressing is not supported by the network, there have been alternative proposals on link loss inference based on sending unicast probing packets, where clever protocols (for example, using back-to-back packets) are incorporated which essentially turns the inference problem to a multicast problem (see, for example, [13], [14]).

For the multicast link loss inference problem, Caceres *et al.* [3] present an ML (maximum likelihood) estimator for the link loss rates that is *asymptotic optimal* (namely, approaching the true ML estimator for asymptotically large number of probes). The Expectation-Maximization algorithm is also presented as a solution to this problem [15], [26]. It may be arguable that these techniques can be applied to link loss inference in wireless sensor networks, a main perspective of this paper is however the concern of the algorithm complexity when it is used as a long-term monitoring daemon. Comparing with previous works, the factor graph approach we present, although sub-optimal, demonstrates good performance and most importantly, very low complexity.

In this section, we will first give a brief introduction on the factor-graph framework as an modeling inference methodology, and then proceed to introduce our algorithm. The performance of the algorithm will be shown in the next section.

A. Factor Graphs and the Sum-Product Algorithm

Recently, the notion of factor graphs has attracted intense research interest in areas of electrical engineering and computer science, since it was recognized that the framework

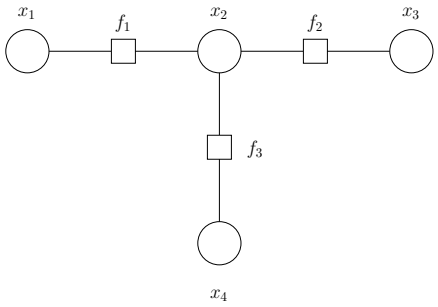

Fig. 3. The factor graph representing $f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_2, x_4)$.

of factor graphs and an iterative algorithm, call the Sum-Product Algorithm, operating on factor graphs unify a variety of previously discovered important algorithms, such as the Viterbi algorithm, BCJR algorithm, Kalman filtering, FFT, belief propagation, forward-backward algorithm etc. In particular, in the community of error correction coding, it is shown that factor graphs and the Sum-Product Algorithm underlie the methodology of the most celebrated error control coding schemes, turbo codes [8] and low-density parity-check codes [9]–[11].

To date, the notion of factor graphs includes multiplicative factor graphs and convolutional factor graphs [2], [5], [6]. In this paper, we will mainly deal with multiplicative factor graphs, referred to as factor graphs from here on, for simplicity.

A factor graph is a bi-partite graph representing the factorization structure of a multivariate function into a product of functions (factors), each involving only a subset of the variables. There are two types of vertices in the graph, variable vertices, representing the variables of the global multivariate function, and function vertices, representing the factors in the factorization; a variable vertex is connected to a function vertex by an edge if the variable is an argument of the factor. For example, Figure 3 is a factor graph representing the factorization $f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_2, x_4)$, where each square box is a function vertex representing factor f_1, f_2 or f_3, and each circle is a variable vertex representing variable x_1, x_2, x_3 or x_4.

A factor graph can be used as a probabilistic graphical model which represents a joint probability mass function (PMF)[1] of random variables. In this case, each variable node represents a random variable, and each factor represents either the joint or conditional joint PMF (PDF) of a subset of random variables, and conditioned upon any subset of random variables corresponding to a cut-set of the graph, the separated two subgraphs (induced by removing the cut-set vertices) are independent. For example, let $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$ and $\mathbf{X}_4$ be a set of random variables where conditioned on $\mathbf{X}_2$, random variables $\mathbf{X}_1$, $\mathbf{X}_3$ and $\mathbf{X}_4$ are independent of each other. Then the joint PMF of the four random variables may be represented by the factor graph in Figure 3, where we may interpret $f_1(x_1, x_2)$ as the joint PMF of $\mathbf{X}_1$ and $\mathbf{X}_2$, $f_2(x_2, x_3)$ as the conditional PMF of $\mathbf{X}_3$ given $\mathbf{X}_2$, and $f_3(x_2, x_4)$ as the conditional PMF of $\mathbf{X}_4$ given $\mathbf{X}_2$. We note that the interpretation of f_1, f_2 and f_3 is in general not unique; for example, $f_1(x_1, x_2)$ may represent the conditional PMF of $\mathbf{X}_1$ given $\mathbf{X}_2$, $f_2(x_2, x_3)$ may represent the conditional PMF of $\mathbf{X}_3$ given $\mathbf{X}_2$, and $f_3(x_2, x_4)$ may represents the joint PMF of $\mathbf{X}_2$ and $\mathbf{X}_4$. That is, as a probabilistic model, a factor graph representing a joint PMF fundamentally specifies a set of conditional independence relationships and the functions (factors) represented by the function vertices may take an arbitrary scale, subject to the constraint that the product of the functions satisfy as a PMF or PDF (the sum or integral of the product over all variables equals to 1).

A useful function for representing a deterministic constraint in a factor graph is the constraint indicator function: let $C(x)$ be a constraint (a boolean proposition) on a possibly vector-valued variable x, then the constraint indicator function of $C(x)$ is defined as:

$$\delta[C(x)] := \begin{cases} 1, & if\ C(x), \\ 0, & otherwise. \end{cases}$$

That is, the constraint indicator function evaluates to 1 if the constraint is satisfied, and to 0 otherwise.

Unifying various algorithms, the Sum-Product Algorithm is an algorithm that operates iteratively on a factor graph by "passing messages" between function vertices and variable vertices. The "messages" are essentially functions (for continuous-valued variables) or tables (for discrete valued variables) computed in the intermediate steps of the algorithm. If $f(x_1, x_2, \ldots, x_m)$ is a function that factors according to a factor graph having no cycles, it is known that the Sum-Product Algorithm can simultaneously compute $\sum_{\sim x_1} f(x_1, \ldots, x_m)$, $\sum_{\sim x_2} f(x_1, \ldots, x_m), \ldots,$ and $\sum_{\sim x_m} f(x_1, \ldots, x_m)$ in parallel, where $\sum_{\sim x_i}$, refers to summation [2] over all variables except x_i. For a concrete understanding of the Sum-Product Algorithm, the reader is referred to [2] and [7]. In essence, what underlies the Sum-Product Algorithm is the distributive law between multiplication and summation (or the generalized distributive law on any semiring, with arbitrarily defined multiplication and summation [7]).

When the function $f(x_1, \ldots, x_m)$ represented by the factor graph is a joint (or conditional joint) PMF, then the objective of the Sum-Product Algorithm coincides with the objective of many inference problems, i.e., finding the maximizing config-

[1] or probability density function (PDF). For simplicity, we will often omit mentioning the term PDF.

[2] In fact the summation operation here can be made more general. In particular, if the summation operation is the $\max$ operation, the Sum-Product Algorithm is referred to as the Max-Product Algorithm. See also footnote 3 below.

uration for marginal PMF $\sum_{\sim x_i} f(x_1, \ldots, x_m)$[3]. Clearly, our formulation for the link-loss inference problem in the previous section is such an example.

When the factor graph representing the function $f(x_1, \ldots, x_m)$ contains cycles, it has been shown in various recent works that the Sum-Product Algorithm can still be used as an excellent approximation algorithm, particularly when the objective is to find the maximizing configuration for $\sum_{\sim x_i} f(x_1, \ldots, x_m)$ and not the maximum itself. In fact, the decoding methods for turbo codes [8] and low-density parity-check codes (see e.g., [11]) are precisely the Sum-Product Algorithm applied on factor graphs with cycles, and the performance of the Sum-Product Algorithms enables these codes to achieve the Shannon limit of digital communications. An intuitive explanation on why the algorithm works so well is that the factor graphs used in these schemes are large and sparse, and the effects of cycles fades away after a few iterations. Also due to the fact that the graphs are sparse, the complexity of the algorithm is essentially linear in the average vertex degree, which make the Sum-Product Algorithm highly scalable.

In the cases when the factor graph contains cycles the passing of messages in the factor graph may be carried out in various orders, typically referred to as the *schedules* of the algorithm [2], [28]. For example, a popular schedule, known as the "flooding" schedule, is that in each iteration, all variable vertices first pass messages and then all function vertices pass messages, where the message-passing rules (the definitions of the messages) stay the same. Summary messages may also be computed at each iteration for the purpose of identifying convergence. The algorithm is usually terminated upon convergence of the summary messages or upon reaching a pre-set number of iterations.

It should be noted that when the factor graph contains cycles, any schedule of the Sum-Product Algorithm will lead to a sub-optimal solution to the maximization problem. However, it has been reported that by adjusting the schedule of the algorithm there can be extra gain in the sub-optimality, i.e., the found solution can be closer to the true optimal configuration [28].

B. Proposed Algorithm

We assume that α_e for each e takes discrete values from set $S := \{1/L, 2/L, \ldots, 1\}$, where L is a positive integer. With no *a priori* knowledge, we assign to each α_e a uniform prior over S.

Let the joint PMF of $X_E^{(i)}$ parametrized by α_E be denoted by $B_E(X_E, \alpha_E)$. Upon the assumption that each link suffers from independent link loss rates, we have

$$B_E(X_E, \alpha_E) = \prod_{e \in E} B(x_e, \alpha_e). \tag{1}$$

Then the joint PMF of α_E, $X_E^{(1,n)}$ and $X_W^{(1,n)}$ factors as

$$P(\alpha_E, X_E^{(1,n)}, X_W^{(1,n)}) \propto \prod_{i=1}^{n} B_E(X_E^{(i)}, \alpha_E) P_{W|E}(X_W^{(i)}, X_E^{(i)}), \tag{2}$$

where $P_{W|E}(X_W^{(i)}, X_E^{(i)})$ is a the conditional PMF of $X_W^{(i)}$ given $X_E^{(i)}$, and in fact,

$$P_{W|E}(X_W^{(i)}, X_E^{(i)}) = \prod_{w \in W} \delta[x_w^{(i)} = \bigoplus_{e \in w} x_e^{(i)}]. \tag{3}$$

Then our objective becomes finding $\hat{\alpha}_e$ for each $e \in E$ that maximizes

$$P[\alpha_e | X_W^{(1,n)}] \propto \sum_{\sim \alpha_e} P[\alpha_E, X_E^{(1,n)}, X_W^{(1,n)}] \propto \sum_{\sim \alpha_e} \prod_{i=1}^{n} B_E(X_E^{(i)}, \alpha_E) P_{W|E}(X_W^{(i)}, X_E^{(i)}). \tag{4}$$

Notice that the objective of finding for each $e \in E$, the maximizing $\hat{\alpha}_e$ for function

$$\sum_{\sim \alpha_e} \prod_{i=1}^{n} B_E(X_E^{(i)}, \alpha_E) P_{W|E}(X_W^{(i)}, X_E^{(i)})$$

precisely coincides with the objective of the Sum-Product algorithm [4], and function

$$\prod_{i=1}^{n} B_E(X_E^{(i)}, \alpha_E) P_{W|E}(X_W^{(i)}, X_E^{(i)}) \tag{5}$$

is also ready to be represented by a factor graph as of Figure 4 (a). That is, the sum-product algorithm can be used to simultaneously find $\hat{\alpha}_e$ for all $e \in E$ in parallel.

If we further express the factors $P_{W|E}(\cdot)$ and $B_E(\cdot)$ in (5) according to (3) and (1), the factor graph can be expanded to a form similar to Figure 4 (b). Note that in the factor graph of Figure 4 (b), we consider the case where $n = 2$ and the network takes the topology in Figure 2 to simplify the factor graph, merely for illustration purpose.

The Sum-Product Algorithm will be applied on the factor graph in Figure 4 (b) to obtain the estimate $\hat{\alpha}_e$ of α_e for each e which maximizes (4). Although the factor graph in Figure 4 (a) is cycle-free, typically its expanded form as of Figure 4 (b) contains cycles. This makes the application of the Sum-Product algorithm an approximation algorithm. Notice in Figure 4 (b), the factor graph consists of "layers" of subgraphs, each corresponding to a time instant t_i. This allows a natural schedule for message-passing in the Sum-Product Algorithm, i.e., first passing messages in each layer, which we refer to

[3]In some inference problems, the objective is to find the maximizing configuration for the joint PMF $f(x_1, \ldots, x_m)$; this can be solved using the Max-Product Algorithm on the factor graph representing f; see [2].

[4]More rigorously speaking, the objective of the Sum-Product Algorithm is to find the marginals, not the maximum of the marginals.

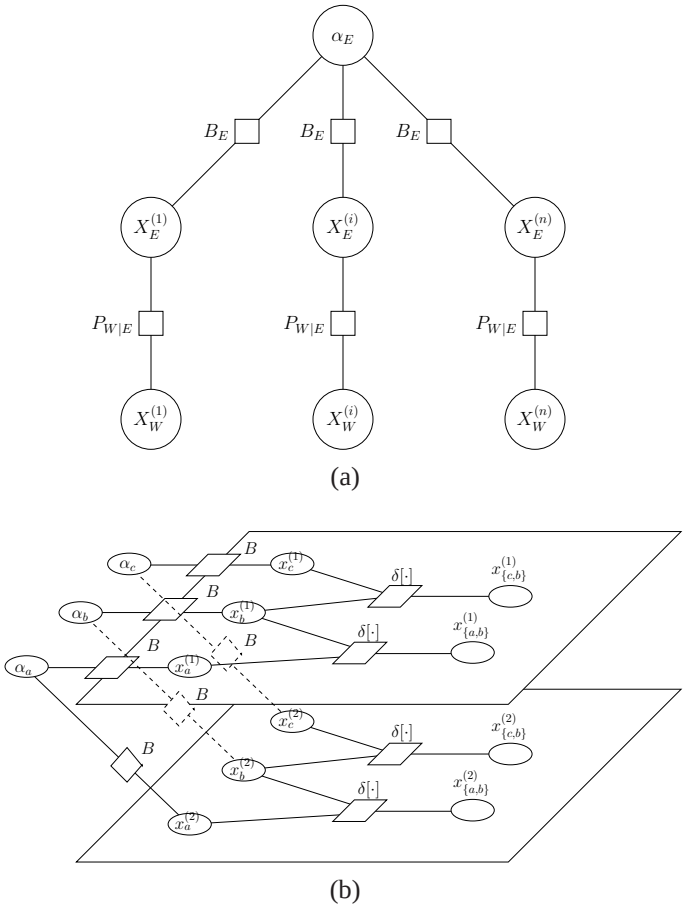

Fig. 4. (a) The factor graph representing the function $\prod_{i=1}^{n} B_E(X_E^{(i)}, \alpha_E) P_{W|E}(X_W^{(i)}, X_E^{(i)})$ and (b) its expanded form, by letting $n = 2$ and taking the network as of Figure 2.

as "intra-layer" message passing, and upon convergence, each layer of the graph passes messages to the vertices representing α_e, which we refer to as "inter-layer" message passing. With such a schedule, the processing of the path state at each time instant can be carried out independently, this will serve to significantly reduce the complexity in long-term monitoring, as will be addressed in Section IV.

We now present the algorithm.

Step 1. Intra-Layer Message Passing. The goal of this step can be understood as obtaining the posterior distribution for each link state at time instant t_i, conditioned upon observation on the i^{th} batch of packets. In each layer, we use a "flooding" schedule similar to what was explained in Subsection III-A. One may follow the recipe discussed in [2] for the derivation the message-passing rule, where for the i^{th} layer, the involved vertices are variable vertices representing $x_e^{(i)}$ and $x_w^{(i)}$ and the function vertices representing $\delta[\cdot]$. Following the derivation, one should see that the message-passing rule can be in fact made more compact by, in each layer, disregarding the vertices representing $x_w^{(i)}$ and passing messages only between the $\delta[\cdot]$ function vertices and the $x_e^{(i)}$ vertices — note that the messages passed from the $x_w^{(i)}$ vertices stay as constant from iteration to iteration; this is simply because the $x_w^{(i)}$ vertices are leaf vertices. We will then use w to denote function vertex $\delta[\cdot]$ that connects to $x_w^{(i)}$, e to denote variable vertex $x_e^{(i)}$.

The message-passing rule for Step 1 is summarized as follows, where each message is a single number representing the (posterior) probability of a link taking state 1 (this is possible since each message is originally a function defined on $\{0, 1\}$, but since the values of the function at 0 and at 1 are dependent, i.e., summing to 1, one can reformulate the message as a single number).

In the initialization phase, each vertex e passes message $\mu_{e\to w}$ representing the uniform distribution over the link state x_e to every adjacent $\delta[\cdot]$ function vertex $w \in \mathcal{N}(e)$. That is, $\mu_{e\to w} = 1/2$.

In the propagation phase, then messages are passed iteratively between the variable vertices $e \in E$ and function vertices $w \in W$. Similar to the "flooding" schedule in Subsection III-A, each iteration begins with every function vertex w passing messages to all its adjacent variable vertices $\mathcal{N}(w)$; then every variable vertex e passes messages to its adjacent vertices $\mathcal{N}(e)$; the message sent from any vertex u to any of its neighbor vertex v is calculated using only the incoming messages from $\mathcal{N}(u) \setminus \{v\}$. The message passed from a function vertex w to a variable vertex e is given by

$$\mu_{w\to e} = \begin{cases} 1, & \text{if } x_w^{(i)} = 1, \\ \dfrac{1 - \prod\limits_{e' \in \mathcal{N}(w)\setminus\{e\}} \mu_{e'\to w}}{2 - \prod\limits_{e' \in \mathcal{N}(w)\setminus\{e\}} \mu_{e'\to w}}, & \text{if } x_w^{(i)} = 0. \end{cases} \tag{6}$$

and the message passed from a variable vertex e to a function vertex w is given by

$$\mu_{e\to w} = \frac{\prod\limits_{w' \in \mathcal{N}(e)\setminus\{w\}} \mu_{w'\to e}}{\prod\limits_{w' \in \mathcal{N}(e)\setminus\{w\}} \mu_{w'\to e} + \prod\limits_{w' \in \mathcal{N}(e)\setminus\{w\}} (1 - \mu_{w'\to e})}. \tag{7}$$

At the end of each iteration, a summary message μ_e is computed for each variable vertex e using all incoming messages to e, as

$$\mu_e = \frac{\prod\limits_{x_{w'} \in \mathcal{N}(e^\star)} \mu_{w'\to e}}{\prod\limits_{x_{w'} \in \mathcal{N}(e^\star)} \mu_{w'\to e} + \prod\limits_{x_{w'} \in \mathcal{N}(e^\star)} (1 - \mu_{w'\to e})}. \tag{8}$$

The iterative process is terminated when the summary messages converge to a steady state or when a pre-set maximum number of iterations is reached.

We will denote the computed value μ_e in the i^{th} layer of the graph by $\mu_e^{(i)}$. One may verify that when each layer of the factor graph is cycle-free, then the algorithm converges definitely with a finite number of iterations, and the computed $\mu_e^{(i)}$ is precisely the posterior of x_e given the observation $X_W^{(i)}$.

Step 2. Inter-Layer Message Passing. In this step, we apply the Sum-Product algorithm by passing-messages from

network size	links/path	paths/link
5000	2.97	2.72
10000	3.71	3.46
20000	7.41	7.10
50000	6.33	6.16

TABLE I

THE AVERAGE NUMBER OF LINKS THAT A PATH CONTAINS AND THE AVERAGED NUMBER OF PATHS THAT A LINK JOINS.

every layer of the graph to vertices $\{\alpha_e : e \in E\}$ (through the vertices representing function $B(\cdot)$), to compute the posterior $P[\alpha_e|X_W^{(1,n)}]$. Due to the simple cycle-free graph structure at this level (that shown in Figure 4 (a)), this step of the Sum-product algorithm can be in fact formulated in the following closed form:

$$P[\alpha_e|X_W^{(1,n)}] \propto \prod_{i=1\ldots n} \left(\mu_e^{(i)}\alpha_e + (1-\mu_e^{(i)})(1-\alpha_e)\right). \quad (9)$$

We then choose $\hat{\alpha}_e$ that maximizes (9) as the estimate of α_e. This can be done by evaluating the function for every element of $S = \{1/L, 2/L, \ldots, 1\}$ numerically and finding the maximizing value $\hat{\alpha}_e \in S$.

IV. SIMULATION AND DISCUSSION

A. Simulation Setup

To investigate the performance and the scalability of the proposed algorithm, we generate sensor networks with random tree topologies, consisting of 5000, 10000, 20000, and 50000 nodes. Table I lists the averaged number of links that a path contains in a network and the averaged number of paths that a link joins.

To each edge (link) e in the network, a random loss rate $(1-\alpha_e)$ is assigned, where α_e is drawn from distribution with probability density function

$$f(\alpha) = \lambda\alpha^{(\lambda-1)}, \alpha \in (0,1].$$

Easy to generate and tune, this random variable has the expected value $\lambda/(1+\lambda)$.

At time instant t_i, the state $x_e^{(i)}$ of edge e is generated from the Bernoulli distribution parametrized by α_e, and the state $x_w^{(i)}$ of each path w is observed by the center node via the i^{th} batch of packets. In each simulation, the number n of packet batches is chosen as $50, 100, 200, 500, 1000,$ and 2000, and the average value $\bar{\alpha}$ of α_e over all the links of each network is chosen as $0.9, 0.92, 0.94, 0.96,$ and 0.98.

For Step 1 of the algorithm, we conservatively set the maximum number of iterations to 30 (we observe in simulation that in the majority of cases, the algorithm converges within a small number of iterations, within 10-15 iterations.). In the second step of the algorithm, we discretize α_e to $L = 100$ levels. Estimation error is computed for each link as $\alpha_e - \hat{\alpha}_e$. For each simulated network, the root mean square error (RMSE) is computed as

$$\text{RMSE} = \left(\sum_{e\in E} \frac{|\alpha_e - \hat{\alpha}_e|^2}{|E|}\right)^{1/2}.$$

B. Simulation Results

Figure 5 and Figure 6 show the RMSE as a function of the number n of batches, and as a function of the $\bar{\alpha}$, respectively, for each simulated network. These results exhibit the general performance trend of this algorithm, which is also observed in other parameter settings of our simulations.

First, the estimation error decreases as the number of batches increases. However, it is worth noting that error will have a lower bound of $1/2L$, the quantization error of α_e. If we can infinitely quantize α_e, as the number of packet batches n approaches infinity, we expect $\hat{\alpha}_e$ approaches the actual α_e.

Secondly, the algorithm favors large networks. That is, in general, the larger the network, the better the algorithm performs. This is because, as the network grows, the factor graph corresponding to the network becomes more sparsely connected; to a certain extent, Table I indicates the sparsity of the factor graphs. This result is consistent with the observation of the Sum-Product Algorithm in other applications where the factor graph graph contains cycles (see, for example, [11]), since the independence assumption [2] of the incoming messages in that case is a closer approximation.

Thirdly, without the need of increasing the number of batches, the same or better estimation accuracy can be achieved for large networks. For practical purposes, our simulations suggest that 500-2000 packet batches are sufficient for inferring the link loss rates in large networks.

Furthermore, the estimation error decreases as $\bar{\alpha}$ increases. This implies that more batches are required for the same estimation error, as the average loss rate increases. In another experiment, we observe that in the 50000-node network, based on the observations of 2000 batches of packets, the RMSE increases from 0.03 to 0.11, as $\bar{\alpha}$ decreases from 0.8 to 0.7. That is, significantly more batches are needed for an accurate inference of the latter loss rates. This is consistent with the discussion in [3], [15]. Fortunately, in reality, action should have been taken before the network experiences such heavy link losses, and there is unlikely a need for inferring the link losses for these cases.

In summary, the simulation results suggest that the loss-rate estimates obtained from the proposed algorithm are sufficiently accurate for any practical purpose.

C. Implementation and Complexity

The complexity of this factor-graph based algorithm is linear in the number n of batches and in the number $|E|$ of links. This is in the same order as the existing algorithms (e.g., [3] and

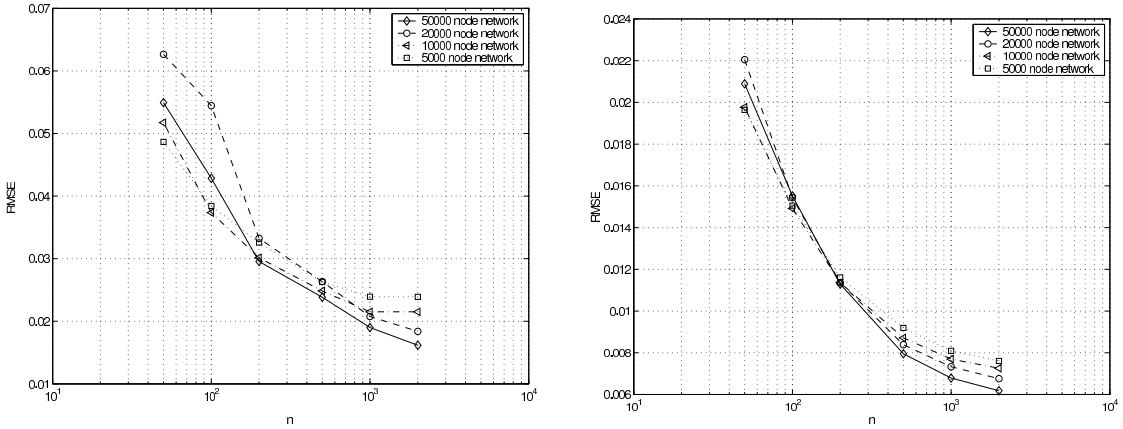

Fig. 5. RMSE for $\bar{\alpha} = 0.9$ (left) and for $\bar{\alpha} = 0.98$ (right), as a function of the number n of batches.

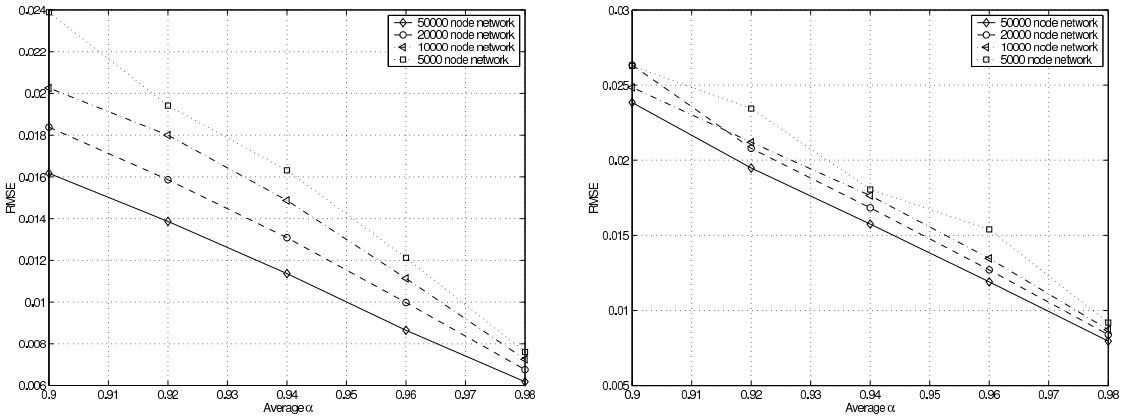

Fig. 6. RMSE with $n = 2000$ (left) and with $n = 500$ (right), as a function of $\bar{\alpha}$.

others). However, as we will explain next, when the algorithm is implemented as a constantly-running monitoring daemon, its complexity for inferring loss rates *at any time instant* is in fact independent of the number of packet batches, n. This makes the proposed algorithm much more suited for long-term monitoring purpose, as compared with other algorithms. We describe this implementation next.

As one may notice, a salient feature of this algorithm is its independent processing of the path states obtained by observing each batch of packets. That is, in practice, we may implement the link-loss monitoring daemon based on the path states obtained by observing a moving window of the most recent n batches of packets, and little computation is in fact needed to infer the current link loss rates. In detail, we may fix a choice of n; then as the monitoring daemon is initiated, upon observing the arrival or loss of the packets in the i^{th} batch, we start computing $\mu_e^{(i)}$ (for all $e \in E$), for $i = 1, 2, \ldots, n$, sequentially. This only involves Step 1 of the algorithm. After obtaining the arrival or loss of the packets in the n^{th} batch, we finish computing $\mu_E^{(n)}$. Then we move to Step 2 of the algorithm and for each $e \in E$, compute the the posterior

$$P[\alpha_e | X_W^{(1,n)}] \propto \prod_{i=1\ldots n} \left(\mu_e^{(i)} \alpha_e + (1 - \mu_e^{(i)})(1 - \alpha_e) \right) \quad (10)$$

and find the maximizing $\hat{\alpha}_e$. After this initial computation, at any later time instant t_k, $k > n$, the posterior of the link-loss rate for any link e can be simply updated recursively by

$$P[\alpha_e | X_W^{(k-n+1,k)}] = P[\alpha_e | X_W^{(k-n,k-1)}] \times \frac{\mu_e^{(k)} \alpha_e + (1 - \mu_e^{(k)})(1 - \alpha_e)}{\mu_e^{(k-n)} \alpha_e + (1 - \mu_e^{(k-n)})(1 - \alpha_e)}.$$

That is, at any time instant t_k, we only need to compute $\mu_e^{(k)}$ according to Step 1, and reuse the previously computed posterior of α_e. This significantly decreases the required computation. In this implementation, the computational complexity for inferring link loss rates at any time instant becomes independent of n. Comparing with other existing algorithms as mentioned earlier (the complexity of which all increases with n), this translates to an appealing computational saving of hundreds or thousands of folds, if the algorithm is to be implementated as a monitoring daemon.

It is worth noting that with this "moving window" implementation, one should expect a trade-off between the estimation accuracy and the estimation sensitivity to the change of link-loss rates. That is, for a small window size n, the estimate is less accurate, but can better track the change of α_e with time; whereas for a large window size n, the estimate is more accurate (provided the link loss rates stay static), but is less

sensitive to the change of link loss rates.

D. Extension to Other Models

Until this point, we have assumed that packets are constantly sent from the terminals to the center in a synchronous manner. In fact, the proposed approach can be adapted to other network scenarios where packets are sent by the terminals asynchronously, or when the terminals do not send packets continuously. However, in those cases, there will be a need of a reliable signaling mechanism from the terminals to the center, or a delicate protocol that allows the center to be aware of a packet sent from a terminal. For example, an out-of-band signaling mechanism may be employed whereby each terminal node informs the center node directly or via multiple hops when it transmits a packet to the center. Then the center node will expect the arrival of the packets sent from the terminals; upon receiving (or not receiving) the packets, the center node can perform link-loss inference using the method presented in this paper. Alternatively, similar mechnisms may be made possible via higher-level protocols. For example, with TCP, when a short series of out-of-order packets are received, congestion is assumed to have occurred and source transmission rates are reduced using window-based flow control. Built on TCP, by checking the order of the received packets, a node (intermediate node or center node) can decide whether an expected packet has arrived — it is safe to assume, with high probability, that an expected packet is lost, if a number of out-of-order packets are received. With this strategy, the details of such a protocol will depend on the data-fusion mechanism employed in the sensor network.

The i.i.d assumption of link losses in our model is perhaps in reality over simplified. In those cases, direct application of the algorithm presented in this paper may lead to less accurate estimates. Nevertheless we note that the frameowork of factor graphs is a universal language for probabilistic modeling, and can be applied to models with arbitrary dependency structure. By using the simple i.i.d assumption, this paper establishes a "proof of concept" for applying factor graph-based approaches to link-loss inference in sensor networks. To demonstrate how to extend the presented method to more realistic (non-i.i.d) loss models, we give a small contrived example: in the the toy example shown in Figure 2, we will allow loss rates of link a and link b to be dependent. Then this dependency can be modeled by introducing a function $\phi_{ab}(\alpha_a, \alpha_b)$ to the joint distribution $P(\alpha_E, X_E^{(1,n)}, X_W^{(1,n)})$, i.e.,

$$P(\alpha_E, X_E^{(1,n)}, X_W^{(1,n)}) \propto \phi_{ab}(\alpha_a, \alpha_b) \prod_{i=1}^{n} B_E(X_E^{(i)}, \alpha_E) P_{W|E}(X_W^{(i)}, X_E^{(i)}),$$

where function ϕ_{ab} accounts for the dependency between α_a and α_b. By properly choosing function ϕ_{ab}, one can obtain any desired dependence model between α_a and α_b. The factor graph representing the above factorization is shown in Figure 7. For real networks, it is possible that the loss rates at a number of links have dependency. This would correspond to function node(s) in the factor graph connecting the variable nodes representing the loss rates at these links.

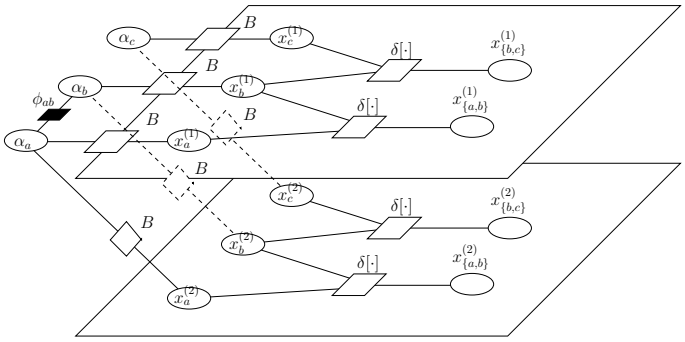

Fig. 7. The factor graph for the example in Figure 2 where α_a and α_b are modeled as dependent. The function node ϕ_{ab} is to model this dependency.

The Sum-Product Algorithm can be derived similarly on such a graph for the inference of α_E. Specifically, one may notice as in Figure 7 that such a factor graph still contains "intra-layer" connections and "inter-layer" connections. However, it is possible in this case that the structure of "inter-layer" connections contains cycles (Figure 7 is *not* such an example; if we introduced two extra function nodes ϕ_{bc} connecting α_b and α_c and ϕ_{ac} connecting α_a and α_c, the resulting graph would be such an example). On such a graph, one may still carry out the two-step passing of messages for the Sum-Product Algorithm. The "intra-layer" message passing may remain the same as we presented earlier, whereas the "inter-layer" message-passing may potentially need modification. Alternatively, one may consider different message-passing schedules. We expect the trade-off between estimation accuracy and computational complexity to depend on the structure of the graph and the choice of message-passing schedules. In practice, more careful investigation for this trade-off is likely to be necessary.

Finally we remark that the Sum-Product Algorithm is not the only algorithm for factor-graph based inference. Other algorithms, such as the Max-Product algorithm, the EM algorithm and various variational methods[5] have also been developed in the framework of factor graphs (see, e.g., [2], [29]). The performance and complexity of these algorithms for solving the problem of this paper certainly deserve further investigation.

[5]In fact, it has been shown that in a variational formulation, the Sum-Product Algorithm on factor graphs with cycles may be understood as iterative maximization of Bethe free energy [30]. Such a nature appears similar to the method of [31], where the true likelihood function is approximated by a "pseudo likelihood function" for computational tractability.

V. Conclusion

In this paper, we exploit the data-aggregation characteristic of wireless sensor networks in the implementation of a link-loss monitoring daemon, where the network-wide link loss rates are inferred upon observing whether packets sent from terminals have arrived. We present a factor-graph based algorithm for this purpose. We show that with very low complexity, this algorithm gives a good estimation of link loss rates, and the algorithm scales particularly well for large networks. We are one of the first to explore the design space towards efficient network inference algorithms in large-scale sensor networks with respect to link loss rates, and the first to use the factor-graph model and the Sum-Product Algorithm to derive suboptimal but computationally lightweight inference mechanisms. We also take the communication paradigm of data aggregation into consideration in our design, which leads to the development of a network inference algorithm with virtually no costs of active probes.

References

[1] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann and F. Silva, "Directed Diffusion for Wireless Sensor Networking," *IEEE Trans. Networking*, vol. 11, no. 1, pp. 2–16, Feb 2003.

[2] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb 2001.

[3] R. Caceres, N. Duffield, J. Horowitz, and D. Towsley, "Multicast-based inference of network-internal loss characteristics," *IEEE Trans. Inform. Theory*, vol. 45, no. 7, pp. 2462–2480, Nov. 1999.

[4] M. Coates, A. O. Hero III, R. Nowak, and B. Yu, "Internet tomography," *IEEE Sig. Proc. Mag*, vol. 19, no. 3, May 2002.

[5] Y. Mao and F. R. Kschischang, "On factor graphs and the Fourier transform," in *Proc. IEEE International Symposium on Information Theory 2001*, Washington, D.C., Jun. 2001, p. 224.

[6] Y. Mao and F. R. Kschischang, "On factor graphs and the Fourier transform," accepted by *IEEE Trans. Inform. Theory*, June, 2002.

[7] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, no. 2, pp. 325–343, Mar. 2000.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Commun. 1993*, pp. 1064–1070, Geneva, Switzerland, May, 1993.

[9] R. G. Gallager, *Low Density Parity Check Codes*, MIT Press, Cambridge, Mass., 1963.

[10] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[11] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[12] M.J. Coates and R. Nowak, "Network inference from passive unicast measurements," Rice University, ECE Department Technical Report TR-0002, Jan. 2000.

[13] M. Coates and R. Nowak, "Network loss inference using unicast end-to-end measurement," in *Seminar on IP Traffic, Measurement and Modelling*, Monterey, CA, Sep 2000, pp. 28:1–9.

[14] A. Bestavros, K. Harfoush, and J. Byers, "Robust identification of shared losses using striped unicast probes," in *Proc. IEEE Int. Conf. Network Protocols*, Osaka, Japan, Nov. 2000, pp. 22–33.

[15] C. Ji and A. Elwalid, "Measurement-based network monitoring and inference: scalability and missing information," *IEEE JSAC*, vol. 20, no. 4, pp. 714–725, May 2002.

[16] N.G. Duffield, F. Lo Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in Proc. of IEEE INFOCOM 2001, Anchorage, AK, April 2001.

[17] A. Adams, T. Bu, R. Caceres, N.G. Duffield, T. Friedman, J. Horowitz, F. Lo Presti, S.B. Moon, V. Paxson, and D. Towsley, "The use of end-to-end multicast measurements for characterizing internal network behavior," IEEE Communications Magazine, May 2000.

[18] D. Rubenstein, J. Kurose, and D. Towsley, "Detecting shared congestion of flows via end-to-end measurement," in Proc. of ACM SIGMETRICS 2000, Santa Clara, California, June 2000.

[19] R. Caceres, N. G. Duffield, J. Horowitz, D. Towsley, and T. Bu, "Multicast-based inference of network-internal characteristics: accuracy of packet loss estimation," in Proc. of IEEE INFOCOM, March 1999.

[20] N. G. Duffield, J. Horowitz, and F. L. Presti, "Adaptive multicast topology inference," in Proc. of IEEE INFOCOM 2001, Anchorage, Alaska, April 2001.

[21] N. G. Duffield, J. Horowitz, F. L. Presti and D. Towsley, "Multicast topology inference from measured end-to-end loss," in IEEE Trans. Inform. Theory, to appear.

[22] N. G. Duffield and F. L. Presti, "Multicast inference of packet delay variance at interior network links," in Proc. of IEEE INFOCOM 2000, Tel Aviv, Israel, March 2000.

[23] Y. Vardi, "Network tomography: estimation source-destination traffic intensities from link data," *J. Amer. Stat. Assoc*. vol. 91, no. 433, pp365–377, 1996.

[24] C. Tebaldi and M. West, "Bayesian inference on network traffic using link count data (with discussion)," *J. Amer. Stat. Assoc*. vol. 93, no. 442, pp557–576, June 1998.

[25] J. Cao, D. Davis, S. Vander Wiel, B. Yu and Z. Zhu, "Time-varying network tomography: router link data," *J. Amer. Stat. Assoc.* vol. 95, pp1063–1075, 2000.

[26] N. G. Duffield, J. Horowitz, D. Towsley, W. Wei, and T. Friedman, "Multicast-based loss inference with missing data," *IEEE JSAC*, vol. 20, no. 4, pp. 700–713, May 2002.

[27] T. Bu, N. Duffield, F. L. Presti, and D. Towsley, "Network tomography on general topologies," in Proc. of ACM SIGMETRICS 2002, Marina Del Rey, California, June 2002.

[28] Y. Mao and A. H. Banihashemi, "Decoding low-density parity-check codes with probabilistic schedule," *IEEE Comm. Letters*, vol. 5, no. 10, pp. 414-416, Oct. 2001.

[29] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families and variational inference," UC Berkeley, Dept. of Statistics, Technical Report 649. Sept, 2003.

[30] D. J. C. MacKay, J. S. Yedidia, W. T. Freeman and Y. Weiss, "A conversation about the Bethe free energy and Sum-Product," MERL Technical Report, TR-2001-18, 2001. Available at http://www.merl.com/reports/TR2001-18/index.html or http://www.inference.phy.cam.ac.uk/mackay/abstracts/bethe.html.

[31] G. Liang and B. Yu, "Maximum pseudo likelihood estimation in network tomography," *IEEE Trans. Signal Processing*, vol. 51, no. 8, pp. 2043-2053, Aug, 2003.

Yongyi Mao Yongyi Mao received his Bachelor of Engineering degree at Southeast University, Nanjing, China, in 1992. In 1995, he received his medical degree at Nanjing Medical University, Nanjing, China. In 1998, he obtained his Master of Science degree at the University of Toronto, in the Department of Medical Biophysics. In 2003, he completed his PhD in electrical engineering at the University of Toronto. He is currently an Assistant Professor at the School of Information Technology and Engineering at the University of Ottawa, Canada. Dr. Mao's research interest includes statistical inference, graphical models, and their applications in communications and bioinformatics.

Frank R. Kschischang Frank R. Kschischang received the B.A.Sc. degree with honors from the University of British Columbia, Vancouver, BC, Canada, in 1985 and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 1988 and 1991, respectively, all in electrical engineering.

He is a Professor of Electrical and Computer Engineering and Canada Research Chair in Communication Algorithms at the University of Toronto, where he has been a faculty member since 1991. During 1997 – 1998, he spent a sabbatical year as a Visiting Scientist at the Massachusetts Institute of Technology (MIT), Cambridge, MA. His research interests are focused on the area of coding techniques, primarily on soft-decision decoding algorithms, trellis structure of codes, codes defined on graphs, and iterative decoders. He has taught graduate courses in coding theory, information theory, and data transmission.

Dr. Kschischang is a recipient of the Ontario Premier's Research Excellence Award. From October 1997 to October 2000, he served as the Associate Editor for Coding Theory for the IEEE Transactions on Information Theory. He was a member of the technical program committee for the 1995 International Symposium on Information Theory (ISIT) held in Whistler, BC, he was Co-Chair and organizer of the 1997 Canadian Workshop on Information Theory held in Toronto, and he served as Publicity Chair for the 1998 ISIT held at MIT. He was Program Co-Chair of the 2004 ISIT held in Chicago.

Baochun Li Baochun Li received his B.Engr. degree in 1995 from Department of Computer Science and Technology, Tsinghua University, China, and his M.S. and Ph.D. degrees in 1997 and 2000 from the Department of Computer Science, University of Illinois at Urbana-Champaign. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently an Assistant Professor and holds the Nortel Networks Junior Chair in Network Architecture and Services. In 2000, he was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems. His research interests include network-level and application-level Quality of Service provisioning, wireless ad hoc networks, and mobile computing.

Subbarayan Pasupathy Subbarayan Pasupathy was born in Chennai(Madras), Tamilnadu, India, on September 21, 1940. He received the B.E. degree in telecommunications from the University of Madras in 1963, the M.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, in 1966, and the M.Phil. and Ph.D. degree in engineering and applied science from Yale University in 1970 and 1972, respectively.

He joined the faculty of the University of Toronto in 1973 and became a Professor of Electrical Engineering in 1983. He has served as the Chairman of the Communications Group and as the Associate Chairman of the Department of Electrical Engineering at the University of Toronto. His research interests are in the areas of communication theory, digital communications, and statistical signal processing. He is a registered Professional Engineer in the province of Ontario. During 1982-1989 he was an Editor for *Data Communications and Modulation* for the IEEE Transactions on Communications. He has also served as a Technical Associate Editor for the IEEE Communications Magazine (1979-1982) and as an Associate Editor for the *Canadian Electrical Engineering Journal* (1980-1983). He wrote a regular humour column entitled "Light Traffic" for the IEEE Communications Magazine during 1984-98.

Pas S. Pasupathy was elected as a Fellow of the IEEE in 1991 "for contributions to bandwidth efficient coding and modulation schemes in digital communication", was awarded the Canadian Award in Telecommunications in 2003 by the *Canadian Society of Information Theory* and was elected as a Fellow of the *Engineering Society of Canada* in 2004.