Monitoring Routing Topology in Dynamic Wireless Sensor Network Systems

Rui Liu liurui@iupui.edu
Yao Liang yliang@cs.iupui.edu
Xiaoyang Zhong xiaozhon@cs.iupui.edu
Department of Computer and Information Science, Indiana University-Purdue University Indianapolis

***Abstract*—In large-scale multi-hop wireless sensor networks (WSNs) for data collection, the ability of monitoring per-packet routing paths at the sink is essential in better understanding network dynamics, and improving routing protocols, topology control, energy conservation, anomaly detection, and load balance in WSN deployments. In this study, we consider this important problem under tremendous WSN routing dynamics, which cannot be addressed by previous methods based on a routing tree model. We formulate the WSN topology inference as a novel optimization problem, and devise efficient decoding algorithms to effectively recover WSN routing topology at the sink in real-time using a small fixed-size path measurement attached to each packet. Rigorous complexity analysis of the devised algorithms is given. Performance evaluation is conducted via extensive simulations. The results reveal that our approach significantly outperforms other state-of-the-art methods including MNT, Pathfinder, and CSPR. Furthermore, we validate our approach intensively with a real-world outdoor WSN deployment running collection tree protocol for environmental data collection.**

***Index Terms*—Wireless sensor networks, routing dynamics, path reconstruction, performance analysis, real-world testbed**

I. Introduction

Monitoring routing dynamics in large-scale multi-hop wireless sensor networks (WSNs) is of paramount importance in better understanding network behaviors under dynamic communication environments, and improving routing protocols, topology control, energy conservation, anomaly detection, and load balance, making optimized operations and management of WSN deployments possible (e.g.,[1, 2, 3]). Due to the severe resource limitation (e.g., memory, bandwidth, and battery power) of unattended wireless sensor nodes deployed in outdoor and harsh environments, typically only limited indirect measurements can be obtained at the WSN sink(s) for the instrumentation of WSN dynamics. However, most studies on WSN inference have focused on link loss and delay monitoring [4-9], where the key assumption made is that routing topology is already known a priori. A few recent studies on WSN routing path inference are mainly restricted to either the static tree routing model due to its minimum overhead [2], or some heuristic techniques [1, 10, 11]. The routing tree model implies that an intermediate node would use its parent node in forwarding a through-packet within any data collection cycle, which may not be realistic due to time-varying wireless channel dynamics in many real-world deployments. In fact, the wireless channel quality (e.g., fading and interference) is highly dynamic for outdoor WSNs deployed in harsh environments, making the routing tree based approaches unreliable and undesirable. To this end, we ask a fundamental question: what approaches would be desirable to reconstruct a WSN per-packet path in dynamic routing under *drastic* communication link dynamics?

To address the above challenges, we consider developing a systematic approach. Using a general routing topology model, we first formulate the WSN topology inference as a novel and compressed sensing (CS) inspired optimization problem, and then devise new efficient algorithms to effectively recover WSN dynamic routing topology at the sink using a small fixed-size path measurement attached to each packet. With such tiny and fixed-size measurement information attached to each packet, the linear combination of labels of traversed communication links is encoded along the path, minimizing packet overhead and saving the severe node resources. Our approach is able to directly and accurately recover the per-packet path and thus the routing topology dynamics of the entire network beyond the routing tree model in near real time. We feel that a CS inspired approach is particularly suited for the problem of monitoring WSN routing topology because the complexity resides at the sink side for topology recovery, where the sink, different from the sensor nodes, is assumed to be not resource-constrained. Furthermore, our approach is general and systematic without imposing any constraints on applications and/or underlying routing schemes. We devise algorithms to reconstruct per-packet paths in both reliable and lossy WSNs.

The major contributions of this work are as follows:

- We present a new systematic approach and algorithms for path reconstruction in multi-hop non-synchronized WSNs, inspired by the compressed sensing concept.
- We conduct extensive simulations to evaluate our approach versus other state-of-the-art methods including MNT [2], Pathfinder [11], and CSPR [18]. Evaluation results reveal that our approach significantly outperforms MNT, Pathfinder, and CSPR.
- We validate our approach through real-world experiments in an environmental multi-hop WSN testbed deployed in a watershed in Western Pennsylvania.

The reminder of the paper is organized as follows. Section 2 presents related works as well as a brief highlight of the novelty of this work. Section 3 presents our approach and formulation. Section 4 presents our WSN topology recovery algorithms. In Section 5, we provide our simulation evaluation results and analyses. Section 6 reports our real-world WSN testbed validation experiments which were conducted for months with more than 200 thousands of received packets. Finally, Section 7 gives the conclusions and outlines our future work.

II. Related Works

The most related works for path inference in WSNs are Multi-hop Network Tomography (MNT) [2], Passive Diagnosis (PAD) [1], PathZip [10], Pathfinder [11], and Compressive Sensing based Path Reconstruction (CSPR) [18]. Following a tree model, MNT utilizes the parent node (i.e., first-hop receiver) information of the locally generated packets (called as anchor packets) from an intermediate node to infer the routing path of each forwarded packet by the node based on the assumption that the routing path is mostly static and packet loss rate is low. The assumptions, however, do not hold in most real-world WSN deployments in extreme communication environments. Thus, MNT fails when consecutive anchor packets travel through different parent nodes due to wireless link dynamics. The advantage of MNT is the minimum packet overhead needed to each packet. Targeting at the application of WSN diagnosis, PAD is a probabilistic inference approach based on Belief network for inferring the root causes of network abnormal phenomena. In PAD, a marking scheme is proposed at sensor nodes for the topology reconstruction at the sink, but each intermediate node has to maintain a cache for its downstream source nodes, which could be adversely large when network size increases. PathZip compresses the path information into a 64-bit hash value carried by each packet. Along a packet route, each forwarder computes the new hash value using a hash function, taking the current forwarder's node ID and the attached hash value in the packet as inputs. Then the sink conducts path search in an exhaustive manner. Pathfinder only stores path difference information in each packet. Different from MNT which uses a set of anchor packets to infer the routing path, Pathfinder uses only one previous packet originating from a forwarder as reference packet to infer the routing path. Pathfinder thus can handle with more routing dynamics for path reconstruction. However, Pathfinder requires that every WSN node must send out local packets with a fixed inter packet interval due to its reference packet identification problem. According to [11], Pathfinder achieved higher path reconstruction ratio than both MNT and PathZip. CSPR, based on CS, represents a path as a sparse node vector whose element corresponds to a node in the WSN. CSPR requires to collect a certain number of packets (i.e., the path measurements) for a particular routing path from multiple data collection cycles before it can reconstruct the path using the traditional CS technique. It fails to recover infrequent paths as no sufficient number of packets could be collected even after lot of collection cycles. Our approach does not rely on any reference packet to infer the per-packet routing path, which is not only more robust in lossy WSNs, but also more general in the sense of no specific restrictions/requirements imposed on WSN deployments and applications. In contrast to CSPR, our approach has a fundamentally different optimization formulation, achieving significantly better performance with much less node resource than CSPR. This paper significantly extends our previous work [21], addressing general non-synchronized lossy WSNs with new algorithms, theoretical analyses, performance comparisons, and real-world testbed validations.

III. Our Approach

A. Routing Model

To account for WSN routing dynamics even within a single cycle of data collection, we consider a general routing topology model for WSN by a directed acyclic and connected graph $G = (V, E)$, where V is a set of n nodes (i.e., the sink s and $n-1$ sensor nodes) with its cardinality $|V| = n$, and E is a set of edges. A directed edge $e_{u,v}$, where $(u, v) \in V \times V$, represents the wireless communication link from node u to node v. Let $p_i = \{e_{i,r_1}, e_{r_1,r_2}, \cdots, e_{r_j,s}\}$ denote a routing path originating from source sensor node i to the sink s where $r_1, r_2, \cdots, r_j$ are intermediate nodes for relay in the route.

In contrast, the commonly used routing tree topology model is a directed spanning tree $T = (V, E^0)$, where the sink is the root, and E^0 is the edge set with $|E^0| = n - 1$. Clearly, $T(T \subseteq G)$ is a minimum connected graph, which means that each node has a unique path toward the sink in any given collection cycle, whether initially sending or relaying packets. This static routing tree model allows for a simple recovery of the routing topology, but it is not feasible in large-scale multi-hop WSNs where dynamic routing is inevitable during a data collection cycle due to drastic channel dynamics.

To facilitate our study, the general routing topology model of G is represented as a (directed) spanning tree augmented with some additional edge(s), where these additional edge(s) are called as 'shortcut(s)'. Hence, our general routing topology G is referred to as a (directed) Augmented 'Tree' (A-Tree). Let E^+ denote the set of shortcuts, then for our A-Tree model $G = (V, E^A)$ we have $E^A = E^0 \cup E^+$, with $|E^A| = |E^0| + |E^+| = n - 1 + |E^+|$. That is, an A-Tee model allows an individual sensor node to send and forward packets that may traverse different paths to the sink at different time instances in a collection cycle due to the dynamic routing under tremendous wireless channel dynamics. An illustration of an A-Tree is given in Fig. 1. The A-Tree in Fig. 1(a) is shown as a static tree plus a set of two shortcuts, as illustrated in Fig. 1(b). In this example, the routing path of node 3 was $p_3 = \{e_{3,1}, e_{1,0}\}$, but edge $e_{3,1}$ was no longer available at the moment, due to the degradation of the link quality, when a packet from node 4 reached node 3 and needed to be forwarded to the sink; thus edge $e_{3,2}$, a shortcut, was chosen for this packet, forming the routing path $p_4 = \{e_{4,3}, e_{3,2}, e_{2,0}\}$. Similarly, when another packet from node 5 arrived at node 3, both edges $e_{3,1}$ and $e_{3,2}$ became not available; therefore, another new shortcut $e_{3,0}$ was selected for forwarding at node 3, resulting in the path for the packet as $p_5 = \{e_{5,3}, e_{3,0}\}$.

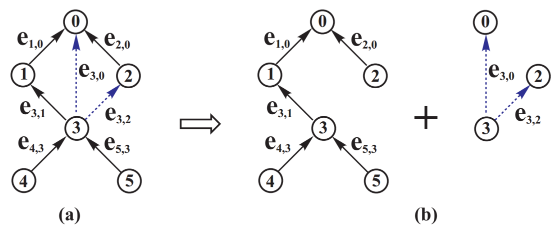

Fig. 1. An illustration of an A-Tree.

B. Problem Formulation

Given a WSN of size n, let y_i denote a path measurement of path p_i piggy-back to a packet received at the sink. Therefore, $Y = \{y_1, y_2, \cdots, y_M\}^T$ denotes a measurement vector ($M = n - 1$), a complete set of M path measurements collected from all sensor nodes of the WSN in a data collection cycle. If each per-packet path of M paths $\{p_1, p_2, \cdots, p_M\}$ is reconstructed at the sink for each collection cycle, the entire dynamic routing topology $G(V, E^A)$ in that cycle can be reconstructed accordingly.

We introduce the concept of Base Topology of multi-hop WSN upstream routing for data collection, denoted by $G^*(V, E^*)$ with $|V| = n$, defined as the superset of all possible routing topologies of the WSN. Any link e ($e \in E^*$) is assigned a unique label value l_e by a labeling function $L : E \to N$, where N denotes the set of positive integers. As outgoing links from the sink are excluded for WSN data collection, the total number of all possible directed wireless links (considering asymmetry wireless channel property) for G^* should be $|E^*| = n(n-1) - (n-1) = (n-1)^2$.

With commonly used WSN routing protocols (e.g., Collection Tree Protocol [12]), good wireless links will be reused, to improve the WSN performance. Our insight is that the number of different wireless links actually used to form a WSN routing topology $G_i(V, E_i^A)$ in any data collection cycle i would be much fewer than the potential choices in its base topology G^*. In other words, the dynamic routing topology G_i is sparse in comparison with the base topology G^*, i.e., $|E_i^A| \ll |E^*|$. Hence, we formulate our approach inspired by the compressed sensing concept [13, 14]. The standard CS framework is as $Y = \Phi X$, where X is an $N \times 1$ sparse discrete signal vector, Φ is an $M \times N$ measurement matrix and Y is an $M \times 1$ measurement vector. The CS theory enables, under certain conditions, the recovery of X from Y where $M \ll N$, as long as signal X is sparse. This can be achieved by solving the following optimization:

$$\hat{X} = argmin_x \|X\|_p \text{ subject to } Y = \Phi X, \tag{1}$$

where $\|X\|_p$ ($p = 0, 1$) denotes l_p-norm of X.

In our formulation, let $N = |E^*| = (n-1)^2$. Let $X' = \{x'_1, x'_2, \cdots, x'_N\}^T$ be a binary indicator vector of N dimension. Let $X = \{l_{e_1} x'_1, l_{e_2} x'_2, \cdots, l_{e_N} x'_N\}^T$, a link vector where l_{e_i} is an assigned integer label for link e_i, represent an A-Tree and shall be sparse. Given any WSN of size n, the sink receives a set of M different path measurements, denoted as an $M \times 1$ vector $Y = \{y_1, y_2, \cdots, y_M\}^T$ where $M = n-1$. The routing matrix of WSN for a given collection cycle can be represented as $\Phi = \{\varphi_{i,j}\}$ $(1 \le i \le M, 1 \le j \le N)$ where the i^{th} row represents the i^{th} path while the j^{th} column represents the j^{th} link, whose elements $\varphi_{i,j}$ are defined as

$$\varphi_{i,j} = \begin{cases} 1, & \text{the } i^{th} \text{ path traverse over the } j^{th} \text{ link;} \\ 0, & \text{otherwise.} \end{cases}$$

Note that $|E^A| = n - 1 + |E^+|$, where $|E^+|$ is the number of shortcuts in A-Tree topology G. Since X is sparse, $|E_i^+|$ should be a relatively small number (e.g., $|E^+| \ll n$) for one collection cycle. Thus, we can formulate the dynamic routing topology inference problem as follows. Given a measurement vector Y at the WSN sink, recover the X and routing matrix Φ, so that

$$\hat{X} = argmin \|X\|_0 \text{ subject to } Y = \hat{\Phi} X, \tag{2}$$

where l_0-norm $\|X\|_0$ is the number of nonzero elements in the vector X, that is $\|X\|_0 = |E^A|$.

We point out that in the traditional CS formulation [13, 14], measurement matrix Φ in (1), whether randomly or deterministically generated, is known a priori. In contrast, the routing matrix Φ in our problem formulation of (2) above is completely *unknown* and determined by the underlying routing scheme operated in an undeterministic real-world dynamic communication environment. However, by a labeling function, we know a link's label value l_e for each potential link $e \in E^*$ a priori. That is, our formulation of (2), different from the traditional CS formulation (1), is to infer Φ and the sparseness pattern of the X, given a Y.

In our approach, a piggy-back path measurement y_i of a packet is encoded by each forwarder as the packet routed through the network towards the sink. We employ modular sum with mod m (SUM_m) rather than arithmetic sum for path measurement encoding, for the purpose of scalable WSN communications and efficient in-network processing. If two routes originating from the same node have the same encoded path measurement value (i.e., a tie), these two routes are basically indistinguishable using the given measurement metric. To reduce the probability of tie in path reconstruction at the sink, exclusive or (i.e., XOR) can be additionally adopted as the second encoding metric of path measurement.

C. Labeling Function

A good labeling function for communication links should satisfy the following conditions: (1) reducing the probability of path measurement ties, and (2) making it easy to generate the link label by each link's endpoint nodes. In this regard, we devise a novel labeling function as given in Theorem 1. It assigns a unique integer for any edge $e_{u,v}$ based on the unique odd integer IDs of two endpoint nodes u and v. This way, any node receiving a packet can easily compute the unique label of the traversed link by this packet on-the-fly, without any pre-stored link label table which can be very big.

Theorem 1. *Assume each node i has a T-bit unique and odd integer ID id_i, for any directed edge $e_{u,v}$, the edge label*

$l_{u,v} = (id_u \times 2^T)\ XOR\ id_v + (id_v - id_u)$ *is a* $2T$*-bit unique and odd integer.*

Proof. For any directed edge $e_{u,v}$, both two node ID id_u and id_v are T-bit integers, so $(id_u \times 2^T)\ XOR\ id_v$ will be a $2T$-bit integer value as well as the edge label $l_{u,v}$.

The two node ID id_u and id_v are also odd integers. Therefore, $(id_u \times 2^T)\ XOR\ id_v$ is an odd integer while $(id_v - id_u)$ is an even integer, that is the sum of these two integers $l_{u,v}$ is an odd integer value.

To prove the edge label $l_{u,v}$ is a unique value, assume there is another edge $e_{u',v'}$ having the same value as $l_{u,v}$. Since the XOR operation in the edge label function has the same effect as addition, we have
$l_{u,v} = (id_u \times 2^T) + id_v + (id_v - id_u)$
$= (id_{u'} \times 2^T) + id_{v'} + (id_{v'} - id_{u'})$,
which could be written as

$$(2^T - 1) \times (id_u - id_{u'}) = 2(id_{v'} - id_v). \qquad (3)$$

If $id_u - id_{u'} \neq 0$ and $id_{v'} - id_v \neq 0$ in equation (3), the right hand side is less than $2(2^T - 1)$ but the left hand side is no less than $2(2^T - 1)$, which is impossible. Thus, it must be $id_u = id_{u'}$ and $id_{v'} = id_v$ for equation (3) to hold. Since each node ID is an unique integer, we show there does not exist another edge $e_{u',v'}$ whose label equals to $l_{u,v}$. Therefore, each edge label is a $2T$-bit unique and odd integer. □

IV. Algorithms

A. NS-RTR Algorithm

Considering that practically nodes in a WSN may not be synchronized in their data sending, we develop our algorithms to infer routing topology for general non-synchronized WSNs. It can be seen that to solve an instance of the problem of (2) is to solve M individual instances of the subset sum problem, which is NP-hard. However, based on the sparsity of X, some effective reconstruction algorithms are possible.

1) Assumptions: Assume that every sensor node in a WSN sends (at least) a packet to the sink in a collection cycle. First, we consider WSN routing topology inference for reliable WSNs, i.e., no packet loss during a cycle of data collection. From our WSN testbed observation, as well as the reported result in [11] which showed 98% of packets had less than two simultaneous shortcuts in a real-world large-scale outdoor WSN deployment with high routing dynamics, we have the following assumptions of sparseness with respect to A-Tree routing model, to simplify the design of algorithms.

- Any packet originating from a source node will not introduce more than one new shortcut link in its route towards the sink;
- The total number of the shortcuts in the A-Tree is bounded by a given constant K in any collection cycle, i.e., $|E^+| < K$, where $K \ll n$.

In addition, we assume that the ID of source sensor node, the hop count of route and the parent node ID of source node are available in each packet, such as in the popular CTP, without adding any new overhead to a WSN packet. We note that with the hop count, our devised algorithms can reconstruct *loopy* routing paths, although loops are not included in A-Tree model.

```
Notation
$select(s)$: select the sparest solution(s) from the
set $s$, and return them in a set.
Function NS-RTR($Packets, root$)
1:  $TPSet \leftarrow \{\}$; $staticTree \leftarrow \{\}$; $dependentMap$
    $\leftarrow \{\}$; /*initializing variables*/
2:  $\{staticTree, leftPackets, dependentMap\} \leftarrow$
    buildStaticTree($Packets, root$)
3:  buildATrees($staticTree, leftPackets$)
4:  return select($TPSet$)
```

Fig. 2. NS-RTR algorithm.

2) Design of Algorithm: In this section, we develop our Non-Synchronized Routing Topology Recovery (NS-RTR) algorithm for general non-synchronized multi-hop WSNs. Fig. 2 outlines the main NS-RTR algorithm. With the given parent node ID in each packet, a static routing tree could be built by the edges from each source node to its parent in a straight-forward way. Then, for the packets which do not follow the routing paths of their parent nodes, their introduced shortcuts and thus their paths will be recovered in the process of building A-Trees. Finally, the sparest possible path set(s) will be chosen as the solution(s).

Some details of NS-RTR algorithm are described as follows. Each packet from source node t is attached with a path measurement $y_t = \{y_t^1, y_t^2\}$ encoded based on SUM_m (with mod $m = 2^{2T}$) and XOR, respectively. For convenience, we sometimes use "recovering node i" to refer "recovering the path of a packet originating from node i". These two terms are exchangeable in this paper.

First, a static tree $staticTree$ is built according to the given packet set $Packets$ received at the sink from the call of function buildStaticTree($Packets, root$). This static tree $staticTree$ is a spanning tree as the parent node ID is included for each generated packet at a source node. The $leftPackets$ set contains the packets whose routing paths do not follow the same routing paths of their parent nodes. The dependent map $dependentMap$ is used to record the relations between each parent node and its dependent children nodes that are following the same routing path as their parent node. Thus, if the path of a parent node is recovered, we could easily recover the paths for its dependent children nodes by checking the dependent map. Next, function buildATrees($staticTree, leftPackets$) will recover the routing paths for the source nodes whose packets are in $leftPackets$, where shortcuts are hence introduced in their paths. The buildATrees function may find more than one possible path because of tie situations. All of the possible paths will be put in set $TPSet$ and the sparest one(s) will be chosen as the solution(s).

Here are the details of function buildStaticTree ($Packets$). For each packet, since parent node ID $parent$ of node t is given, a spinning tree $staticTree$ could easily be built

Notation

findMatchedPaths($packet, tree$): find the paths matched the measurement in $packet$ with at most one newly introduced shortcut in the tree $tree$;
updateATree($tree, p$): update A-Tree $tree$ by adding path p and its dependent children in $dependentMap$ if there is any;
group(s): group the same topologies in the set s.

```
Function buildATrees($tree, leftPackets$)
 1:  if ($leftPackets = \{\}$)
 2:  then $TPSet \leftarrow$ group($\{tree\} \cup TPSet$); return;
 3:  $packets \leftarrow leftPackets$;
 4:  for ($i \leftarrow 1; i \leq getSize(leftPackets); i++$)
 5:    $paths \leftarrow$ findMatchedPaths($leftPackets[i], tree$)
 6:    if ($paths \neq \{\}$)
 7:    then
 8:      for all $p \in paths$ do
 9:        $newTree \leftarrow$ updateATree($tree, p$);
10:        $newLeftPackets \leftarrow$
               $packets[i+1$, getSize($packets$)];
11:        buildATrees($newTree$, $newLeftPackets$);
12:        return;
13:      end for;
14:    else
15:      $packets \leftarrow packets[i+1$,
             getSize($packets$)$] \cup \{packets[i]\}$;
16:  end for
```

Fig. 3. Function buildATrees in NS-RTR.

by adding edge $e_{t,parent}$. The path measurement of y_t is compared against the computing result based on the label of the edge from node t to its parent node $parent$ $l_{t,parent}$, and the measurement of its parent node y_{parent}. If measurement y_t matches the computing result, it means the routing path from source node t follows the routing path of its parent node $parent$ and edge $e_{t,parent}$ will be added to the dependent map $dependentMap$. Otherwise, it indicates there is a path change so this packet needs to be added to set $leftPackets$ and its routing path will be recovered by function buildATrees later.

Function buildATrees($tree, leftPackets$), as shown in Fig. 3, tries to recover the paths of packets left in $leftPackets$. All the A-tree solutions will be put in the global variable $TPSet$ which is initially an empty set. If $leftPackets$ is empty, it means that all routing paths have been recovered and $TPSet$ could be updated by joining $\{\{tree\}$,TPSet$\}$. Note, there may be already a same topology tree in set $TPSet$ so the group function is used to remove the duplicates here. If $leftPackets$ is not empty, we check from the first packet in $leftPackets$. If no matched path is found for this packet, move it to the end of the $packets$ set and check the next packet. If one or more paths matched the measurement could be found by function $findMatchedPaths$, update the current A-Tree with each path to get new A-Tree(s). Each new A-tree $newTree$ is passed with the rest packets $newLeftPackets$ to call function buildATrees recursively.

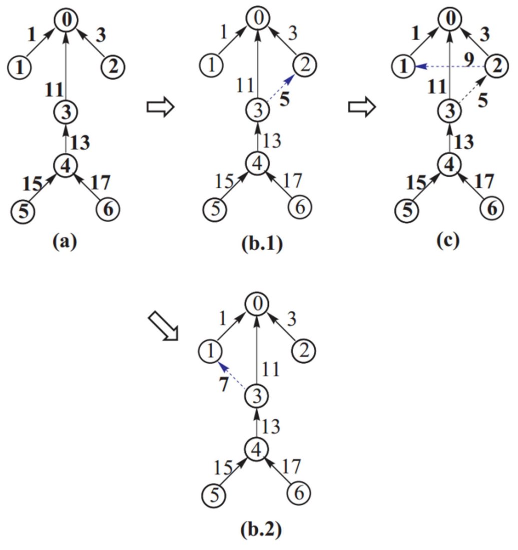

Fig. 4. An example of NS-RTR.

Example 1: Fig. 4 illustrates how the devised NS-RTR algorithm works with a network of 7 nodes. We use a notation of $\{$source, parent, hop count, $\{$measurement value1, value2$\}\}$ to represent a packet. In this network, the sink is node 0; packets received at the sink would be $Packets$=$\{\{1,0,1,\{1,1\}\},\{2,0,1,\{3,3\}\},\{3,0,1,\{11,11\}\},\{4,3,3,\{21,11\}\},\{5,4,4,\{36,4\}\},\{6,4,5,\{45,17\}\}\}$. The order of the packets in the $Packets$ does not matter. Fig. 4(a) shows the static tree $staticTree$ built from function buildStaticTree. At this step, the corresponding set $leftPackets$ is $\{\{4,3,3,\{21,11\},\{6,4,5,\{45,17\}\}\}\}$ and the dependent map $dependentMap$ is $\{0\rightarrow\{1,2,3\},4\rightarrow\{5\}\}$. Node 5 is the dependent child node of node 4 which means it follows the routing path of node 4. So the packet from node 5 is not included in the set $leftPackets$. When the routing path from node 4 is recovered, the path from node 5 could be easily found by checking the dependent map. Then function buildATrees ($staticTree, leftPackets$) is used to recover the paths of the packets in $leftPackets$. If packet $\{6,4,5,\{45,17\}\}$ is checked first, there will be no matched path and this packet will be moved to the end of the $Packet$ If packet $\{4,3,3$, $\{21,11\}\}$ is checked first, there will be two matched paths found: $\{e_{4,3}, e_{3,2}, e_{2,0}\}$ and $\{e_{4,3}, e_{3,1}, e_{1,0}\}$. A tie situation occurs here. So the static tree could be updated to a new A-tree shown in either Fig. 4(b.1) or Fig. 4(b.2). These two new A-trees are used to recover packet $\{6,4,5,\{45,17\}\}$ by calling function buildATrees again. The routing path of the packet originated from node 6 $\{e_{6,4}, e_{4,3}$, $e_{3,2}, e_{2,1}, e_{1,0}\}$ could only be recovered based on the A-tree in Fig. 4(b.1). So the A-tree shown in Fig. 4(c) is the finally reconstructed routing topology in the solution set of this example.

Example 2: Fig. 5 further illustrates an NS-RTR example for loopy path reconstruction in a network of 6 nodes. The

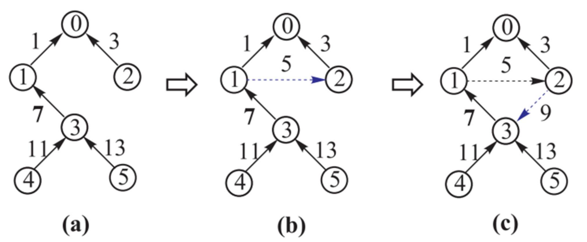

Fig. 5. An example of NS-RTR recovering loopy path.

packets received at the sink are $Packets$=$\{\{1,0,1,\{1,1\}\}$, $\{2,0,1,\{3,3\}\}, \{3,1,2,\{8,6\}\}, \{4,3,4,\{26,10\}\}, \{5,3,6,$ $\{42,0\}\}\}$. Fig. 5(a) shows the $staticTree$, along with which the dependent map $dependentMap$ is {0→{1,2},1→{3}}. The corresponding set $leftPackets$ at this moment is {{4,3,4, {26,10}},{5,3,6,{42,0}}}. The function buildATrees will find the matched path $\{e_{4,3}, e_{3,1}, e_{1,2}, e_{2,0}\}$ for the first left packet {4,3,4,{26,10}} and update $staticTree$ to a new A-Tree with the new shortcut $e_{1,2}$ as shown in Fig. 5(b). Then the path $\{e_{5,3}, e_{3,1}, e_{1,2}, e_{2,3}, e_{3,1}, e_{1,0}\}$ will be found for the next packet {5,3,6,{42,0}}. There is a loop $\{e_{3,1}, e_{1,2}, e_{2,3}\}$ in the routing path for node 5. NR-RTR algorithm is able to recover such loopy path cases with the help of the given hop count information.

3) Fast NS-RTR (FNS-RTR) Algorithm: A solution set obtained from NS-RTR algorithm could contain more than one A-tree solutions due to potential tie situations. However, a tie situation rarely occurs when both SUM_m and XOR are adopted for path measurement encoding. We hence devise a fast version of NS-RTR algorithm, referred to as the FNS-RTR algorithm, which attempts to give a unique true solution with very high probability.

The FNS-RTR algorithm returns the first found solution A-tree and then stops searching. The merit of FNS-RTR algorithm is its speedup since FNS-RTR is likely to save the effort of trying to find either non-existent or duplicated solution(s). The main algorithm structure of FNS-RTR is very similar to that of NS-RTR except that function buildATree is used instead of function buildATrees. The main differences between function buildATree and function buildATrees are marked by underlines in Fig. 6. When a path is found by function findMatchedPath, this path will be used to update the current A-Tree for the left packets. So, there will be only one A-tree reconstructed by function buildATree.

Example 3: Recover the routing topology in Example 1 using FNS-RTR. First, the static tree $staticTree$ built from function buildStaticTree is same as in Fig. 4(a). Also, the corresponding set $leftPackets$ is still {{4,3,3,{21,11},{6,4,5, {45,17}}}} and the dependent map $dependentMap$ is {0→ {1,2,3},4→{5}}. When function buildATree($staticTree$, $leftPackets$) is used to check packet {4,3,3,{21,11}}, one matched path, either $\{e_{4,3}, e_{3,2}, e_{2,0}\}$ or $\{e_{4,3}, e_{3,1}, e_{1,0}\}$, will be found. If $\{e_{4,3}, e_{3,2}, e_{2,0}\}$ is obtained, the static tree could be updated to the new A-tree shown in Fig. 4(b.1) and then

```
Notation
findMatchedPath($packet, tree$): find the first path
matched the measurement in $packet$ with at most
one newly introduced shortcut in the tree $tree$;
updateATree($tree, p$): update A-Tree $tree$ by
adding path $p$ and its dependent children in
$dependentMap$ if there is any.
Function buildATree($tree, leftPackets$)
 1:  if ($leftPackets = \{\}$)
 2:  then $TPSet \leftarrow \{tree\}$; return;
 3:  $packets \leftarrow leftPackets$;
 4:  for ($i \leftarrow 1; i \leq getSize(leftPackets); i++$)
 5:    $path \leftarrow$ findMatchedPath($leftPackets[i], tree$);
 6:    if ($path \neq null$)
 7:    then
 8:      $newTree \leftarrow$ updateATree($tree, p$);
 9:      $newLeftPackets \leftarrow$
             $packets[i+1$, getSize($packets$)];
10:      buildATree($newTree$, $newLeftPackets$);
11:    else
12:      $packets \leftarrow packets[i+1$,
             getSize($packets$)] $\cup \{packets[i]\}$;
13:  end for
```

Fig. 6. Function buildATree in FNS-RTR.

the path of packet {6,4,5,{45,17}} will be recovered later as shown in Fig. 4(c). FNS-RTR will return the solution A-tree in Fig. 4(c). If $\{e_{4,3}, e_{3,1}, e_{1,0}\}$ is found, the new A-tree will be as shown in Fig. 4(b.2) and the routing path originated from node 6 cannot be recovered. Then the FNS-RTR algorithm will not find any solution A-tree and return null. Note, while it is possible for the FNS-RTR algorithm to not find any solution A-tree, the possibility of such situation is very low from our observation, when the effective labeling function given in Section III.C is employed.

4) Complexity Analysis: We examine the complexity of the FNS-RTR algorithm, while the complexity analysis of NS-RTR algorithm is omitted due to the page limit. As shown in subsection IV.A.3, the complexity of the FNS-RTR algorithm is the complexity of function buildStaticTree plus the complexity of function buildATree. With the given parent node information in each packet, the complexity of function buildStaticTree is pretty straightforward. For a WSN of size n, the complexity of function buildStaticTree is $O(n)$.

To analyze the complexity of function buildATree, we first check the complexity of its core function findMatchedPath (line 5 of Fig. 6). We give the following Theorem 2.

Theorem 2. *Given an A-Tree with at most $r = |E^+| < K$ shortcuts, the maximum number of all possible routing paths for any node without loop in this A-Tree is $O(1)$.*

Proof. Let PN denote the number of all possible paths towards the sink for a node in a given A-Tree. The best case is no shortcut along the path for the node, $PN = 1$. The worst case is all shortcuts are along the path: $PN = \prod_{i=1}^{h}(1+k_i)$ where k_i is the number of shortcuts for each node i along the path and h is the hop number of the path. It will not affect the value of PN if we remove or add a factor $(1+k_i)$ when $k_i = 0$. So if $h > r$, we can remove $(h-r)$ factors of $(1+k_i)$ with $k_i = 0$; if $h < r$, we can add $(r-h)$ such factors. Then we can get $PN = \prod_{i=1}^{r}(1+k_i)$ and $\sum_{i=1}^{r} k_i \leq r$ since there are at most r shortcuts in the A-Tree.

Also since k_i should be non-negative integer number, based on AM-GM inequality (inequality of arithmetic and geometric means), $\prod_{i=1}^{r}(1+k_i) \leq (\frac{\sum_{i=1}^{r}(1+k_i)}{r})^r = (\frac{\sum_{i=1}^{r}1+\sum_{i=1}^{r}k_i}{r})^r \leq (\frac{r+r}{r})^r = 2^r$. Therefore, $PN \leq 2^r = O(1)$ since r is a given constant integer. □

Since, according to Theorem 2, the total number of routing path candidates is bounded by a constant for each shortcut candidate to be checked, the complexity of function findMatchedPath depends on the number of shortcut candidates to check. A possible start node of a shortcut for a given left packet could be any node along a possible routing path from the source node's parent node except the sink. As the hop count of any path is bounded by the size of WSN n (without loops), the number of possible start nodes of a shortcut is then $O(n)$. Similarly, a possible end node for a shortcut could be any node in the network, which means the number of possible end nodes of a shortcut is also $O(n)$. Thus, the complexity of the function findMatchedPath, the total number of shortcut candidates to check, is $O(n^2)$.

The complexity of function buildATree depends on how many times the function findMatchedPath will be called. The best case is the shortcuts introduced by individual left packets are independent, in which function findMatchedPath only needs to be called once for each left packet. Since there are $j < n$ packets left initially in any collection cycle, the complexity of function buildATree in the best case is $O(j) = O(n)$. On the other hand, the routing path of one packet may include the shortcut introduced by another packet. Thus, the worst case is that in every round of the **for** loop at line 4 in Fig. 6, all left packets have to be checked before a routing path of the last packet checked is found. So function findMatchedPath will be called $\sum_{i=1}^{j} i$ times in such case, which is $O(j^2) = O(n^2)$. In conclusion, the complexity of function buildATree is $O(n^4)$, and therefore the complexity of the FNS-RTR algorithm is also $O(n^4)$.

B. INS-RTR Algorithm

Our presented NS-RTR algorithms above are for reliable WSNs. However, packets can be lost in real-world WSNs. A source sensor node is called a missing node if its packet did not arrive at the sink in that collection cycle. We further develop a new NS-RTR algorithm for lossy WSNs, referred to as the INS-RTR algorithm, to recover the routing paths with incomplete packet set received. The challenge of the INS-RTR algorithm is to recover any path from a source node that may traverse one or more missing nodes.

```
Notation
getMissingNodes($AllNodes, Packets$): get the nodes from the set $AllNodes$ that do not have a packet received in $Packets$.
addVirtualLinks($virtualTP, n$): add virtual links for missing node $n$ to the topology $virtualTP$, and return new topology with the new virtual links.
removeVirtualLinks($virtualTP$): remove virtual links from the topology $virtualTP$, and return topology with only recovered wireless links.

Function INS-RTR($Packets$)
 1: $MissingNodes \leftarrow$ getMissingNodes($AllNodes$, $Packets$)
 2: $TPSet \leftarrow \{\}$; $staticTree \leftarrow \{\}$; $dependentMap \leftarrow \{\}$; /*initializing variables*/
 3: $\{staticTree, leftPackets, dependentMap\} \leftarrow$ buildStaticTree($Packets, root$)
 4: $virtualTP \leftarrow staticTree$
 5: for all $n \in MissingNodes$ do
 6:   $virtualTP \leftarrow$ addVirtualLinks($virtualTP, n$);
 7: end for
 8: buildATree($virtualTP, leftPackets$)
 9: $TP \leftarrow$ removeVirtualLinks($virtualTP$)
10: return $TP$
```

Fig. 7. INS-RTR algorithm.

1) Assumptions: First, to deal with any lossy WSN, all node IDs of the WSN are known in advance to identify any missing nodes in any collection cycle. Here we assume that the total number of missing nodes is bounded by a given constant in any data collection cycle. Second, while we still assume that each node will not introduce more than one new shortcut link in its route towards the sink, the total number of the shortcuts in an A-Tree now does not need to be bounded by a constant in any collection cycle. Finally, it is assumed that any missing node will only introduce one link, as we attempt to obtain the sparsest solutions by our INS-RTR algorithm for lossy WSNs.

2) Algorithm description: The main challenge is how to infer the routing path of a received packet that has been forwarded by some missing node(s). To address this problem, the basic idea of INS-RTR is to tentatively add virtual links attached to each missing node, with which we are able to apply the similar method as used in NS-RTR algorithms to recover the routing paths of received packets. The devised INS-RTR algorithm is shown in Fig. 7. If there are any intermediate nodes missing when a static tree $staticTree$ is built based on the received packets, the built 'static tree' will not be a connected spanning tree but rather a forest. Some edges are missing due to these missing intermediate nodes. The received packets originating from their children nodes will be put in the set $leftPackets$. Then virtual links are added for each missing node in $MissingNodes$, so that each missing node will connect to every other node in a 'virtual static tree'. Finally, according to the actual links found in

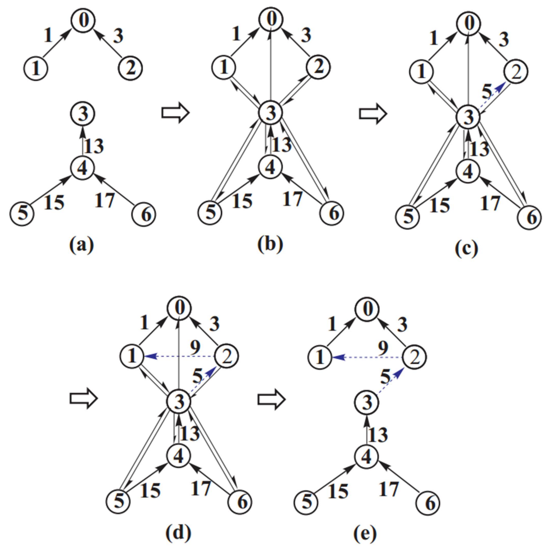

Fig. 8. An example of INS-RTR.

function buildStaticTree and the virtual links added for the missing nodes, function buildATree will be used to recover the packets in $leftPackets$. We could use either function buildATrees in the NS-RTR algorithm to get a set of solutions or function buildATree to get only one solution. The INS-RTR algorithm given in Fig. 7 uses function buildATree described in Fig. 6. The unused virtual links need to be removed if they are not being recovered as actual links/shortcuts in function buildATree. The solution of routing topology will only contain the wireless links along the recovered routing paths for the received packets.

Example 4: To illustrate, the same WSN of Example 1 is re-examined where the packet from node 3 is lost in the given collection cycle this time, resulting in the incomplete packet set $Packets = \{\{1, 0, 1, \{1, 1\}\}, \{2, 0, 1, \{3, 3\}\}, \{4, 3, 3, \{21, 11\}\}, \{5, 4, 4, \{36, 4\}\}, \{6, 4, 5, \{45, 17\}\}\}$. The static tree $staticTree$ built by function buildStaticTree based on the received packets is shown in Fig. 8(a). The edge started from node 3 is missing in $staticTree$ since the packet for node 3 is missing. Set $leftPackets$ and dependent map $dependentMap$ are {{4,3,3,{21,11},{6,4,5,{45,17}}}} and {0→{1,2},4→{5}} respectively. Static tree $staticTree$ is initially expanded to $virtualTP$, in which all the potential virtual links for the missing node 3 are added as shown in Fig. 8(b). Then function buildATree($virtualTP, leftPackets$) is used to check the packets in $leftPackets$. If path $\{e_{4,3}, e_{3,2}, e_{2,0}\}$ is found as the matched path for packet {4,3,3,{21,11}}, the topology will be updated as in Fig. 8(c). Fig. 8(d) shows the topology after recovering routing path $\{e_{6,4}, e_{4,3}, e_{3,2}, e_{2,1}, e_{1,0}\}$ for packet {6,4,5,{45,17}}. Any unused virtual links are then removed and the solution topology is given in Fig. 8(e).

V. Performance Evaluation

We conducted thorough simulations of lossy WSNs in TOSSIM [19], the standard network simulator in TinyOS [17], to evaluate our approach versus other state-of-the-art methods MNT, Pathfinder, and CSPR. TOSSIM utilizes the popular Meyer Heavy noise trace to provide realistic noise model during the simulation [20]. Two network sizes of 200 and 500 uniformly distributed nodes were both simulated for 100 data collection cycles. We keep the density of the sensor nodes unchanged when the network size increases.

TABLE I
Simulation Statistics

WSN Size	200	500
Total packets	19424	42641
Packet delivery ratio	97.12%	85.28%
Total cycles	100	100
Total different path groups	8520	35771
Longest path (hops)	16	25
Avg. shortcuts per cycle	19.24	189.40
Stdev. of shortcuts per cycle	12.12	65.04
No. of path group ties	12	24

Table I illustrates the statistics of the simulations for two network sizes, with the packet delivery ratio being 97.12% and 85.28%, respectively. A $pathgroup$ is a set of packets which are transmitted following the same path [18]. The number of different path groups indicates the number of different routing paths and hence reflects the routing dynamics in the network. For 200-node WSN simulation, there are totally 8520 path groups; for 500-node WSN simulation, the number of path groups has been increased to 35771. According to the last row of Table I, the probability of path group tie is very small, 12/8250 (0.145%) and 24/35771 (0.067%) for 200-node network and 500-node network, respectively, indicating that modular sum and XOR are effective path encoding metrics.

We compare our INS-RTR algorithm with MNT [2], PathFinder [11], and CSPR [18], the three most related works of WSN path inference. We focus not only on per-packet path reconstruction but also on path group reconstruction. For CSPR, a path group, and hence all the packets in the path group, cannot be recovered if it contains insufficient number of packets even after data collections for many cycles. In fact, due to the dynamic nature of the simulated WSNs, it is observed that 92.11% of the path groups in 200-node simulation and 98.95% of the path groups in 500-node simulation contain less than 5 packets.

Fig. 9 shows the successful ratios of the per-packet path recovery and the path group recovery separately. Regarding per-packet path recovery, for 200-node simulation study, INS-RTR has successfully recovered the paths of 91.23% packets, whereas MNT has recovered 46.50%, Pathfinder has recovered 58.32%, and CSPR has recovered 26.98%. Increasing the network size has degraded the performance of all the algorithms. For 500-node simulation study, CSPR has recovered the paths of only 6.90% packets, whereas MNT has recovered 11.37%, and Pathfinder recovered 16.28%. In contrast, INS-RTR has achieved the path reconstruction ratio of 50.65%, still performing much better than the other approaches/algorithms.

The recovery of path groups can provide us with more insights into the different approaches. The number of path

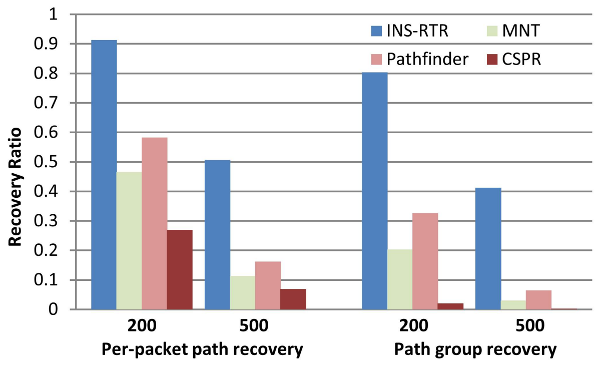

Fig. 9. Comparison among INS-RTR, MNT, Pathfinder, and CSPR.

TABLE II
COMPARISON BETWEEN CSPR AND OUR APPROACH.

	Our Approach	CSPR
Path representation	Link vector	Node vector
Path measurement	Modular SUM&XOR	Modular SUM
Packet overhead	4 or 8 bytes	$\geq$ 8 bytes
reconstruction	Based on the packets in a collection cycle	Based on the packets in a path group
Major constraints	–	Many path groups can not collect enough packets

groups indicates the degree of routing dynamics in a given WSN. Thus, it can be used as an important metric to demonstrate the ability of each approach to really catch routing dynamics. As shown in Fig. 9, INS-RTR significantly outperforms all the other algorithms. It has successfully recovered 80.37% of the path groups in the 200-node simulation and 41.30% path groups in the 500-node simulation. CSPR performs the worst due to insufficient packets in most of the path groups. For 200-node simulation, CSPR only recovered 2.09% of the path groups, whereas for 500-node simulation, it only recovered 0.30% path groups. MNT and Pathfinder have recovered 20.32% and 32.67% of the path groups in the 200-node simulation, respectively. For 500-node simulation, MNT recovered 3.00% of the path groups, whereas Pathfinder recovered 6.45%.

In summary, INS-RTR has significantly outperformed the MNT, Pathfinder, and CSPR on both per-packet path reconstruction and path group recovery. In particular, the evaluation results profoundly reveal that due to their different problem formulations, our approach and CSPR exhibit drastic difference in their respective performances although both are CS-inspired approaches. Table II further summarizes the major differences between CSPR and our approach. The undesirable reconstruction performances of CSPR are mainly caused by its drawback of unable to collect a sufficient number of packets for many highly dynamic paths that do not occur frequently. Thus, CSPR fails to recover those paths even after hundreds of data collection cycles.

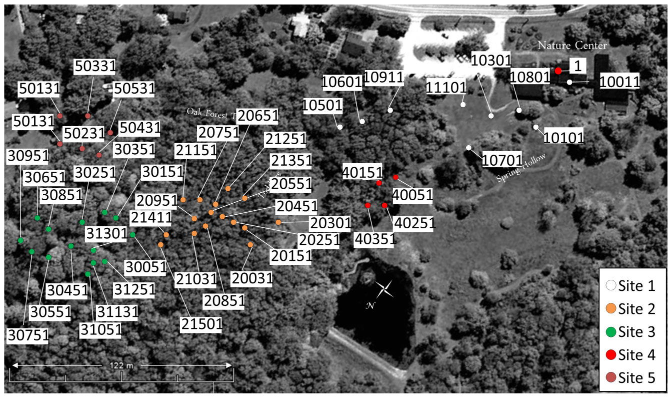

Fig. 10. An illustration of the WSN testbed deployed in a forested nature reserve at ASWP.

VI. Real-World WSN Validation

A. WSN testbed

A real-world outdoor multi-hop WSN testbed is used to further validate and evaluate our approach. This WSN testbed used in our experiments has been deployed in a forested nature reserve at the Audubon Society of Western Pennsylvania (ASWP), Pennsylvania, collecting ground-based data for calibrating and validating scientific models in hydrology research [15, 16]. Two types of nodes, MICAz and IRIS, are deployed running a data collection application developed using CTP in TinyOS 2.1.2. Over 50 sensor nodes are deployed in the monitoring area, As shown in Fig. 10.

B. In-network processing

We developed a lightweight in-network processing layer in node's network stack to encode the piggy-back path information of each packet along the path towards the sink. The in-network processing layer is implemented in TinyOS 2.1.2, between the network layer and the link layer, providing transparent in-network processing service to all upper layers.

A few bytes are added into each packet to carry the compressed measurement of the packet path up to the current receiving node, using either SUM_m alone or SUM_m and XOR of the labels of the traversed links. The use of SUM_m alone is for smaller WSNs to further reduce the packet overhead, whereas the use of both SUM_m and XOR is for large-scale WSNs where path measurement ties may occur. In TinyOS, a node ID is an unsigned 16-bit integer, and hence a link label is 32 bits. Thus the compressed path measurement adds totally four bytes (SUM_m alone) or eight bytes (SUM_m and XOR) overhead to a packet. This overhead is similar to other approaches: eight bytes in CSPR and PathZip, four bytes in MNT, and maximum seven bytes in Pathfinder. The hop counter in CTP is adopted. A source node initially reserves the space of the needed fixed-size measurement overhead to a packet, whereas the path encoding (i.e., SUM_m and XOR) is implemented at each receiver of the packet, as the packet has completed its link communication on this hop once successfully received by a receiver. We note that our approach needs little node resource beyond the four or eight bytes of packet overhead, because the path encoding of a packet can be very easily performed by each forwarder. In contrast, other approaches (e.g., CSPR and PathZip) require a lot node resources in addition to their packet overheads. For example, CSPR requires 200 bytes in each node for storing its dictionary. For the purpose of validation, each packet's actual path is recorded hop by hop in each packet up to the maximum 10 hops for our WSN testbed, which is used as the ground truth. We note that the 20 bytes of path recoding will not be necessary in regular WSN deployments.

C. Testbed Results And Analyses

Each packet received at the WSN sink includes source node ID, parent node ID, the hop count of path, and path measurements. Such information will be used to recover the routing path for each received packet. Every packet also records its full path to validate the recovered path and thus to verify the correctness of our algorithm. A timestamp is added for each packet at the sink to record its arrival time.

We first conducted some preprocessing of received packets at the sink. According to their time stamps, packets are partitioned into different collection cycles. Our INS-RTR algorithm for lossy WSN was applied for path reconstruction due to packet drops in the testbed data collection.

Two tests during two periods of [2013-11-19, 2013-12-04] and [2014-02-21, 2014-03-19] under different WSN dynamics, with totally more than 200 thousands of packets received, are examined in our evaluation. The total number of packets generated in the testbed during a test period can be computed based on packet sequence number assigned at each source node. Detailed information of the two tests and their path reconstruction results are given in Table III. Packet delivery ratios were 90.52% and 87.84% for test 1 and test 2, respectively. Using both SUM_m and XOR in path encoding, path recovery ratios were 99.98% and 99.99% for test 1 and test 2, respectively, whereas path group recovery ratios were 98.78% and 99.26% for test 1 and test 2, respectively. In particular, we observed that even using SUM_m measurement alone INS-RTR algorithm had the same or slightly lower path recovery and path group recovery ratios as those using both SUM_m and XOR, as shown in the last two rows of Table III, indicating that the use of SUM_m measurement alone in our approach could be sufficient for successful routing topology monitoring in WSNs of small and moderate sizes.

TABLE III
TESTBED PACKETS AND PATH RECONSTRUCTION RESULTS

	Test 1	Test 2
Collection Time	2013-11-19 00:00 2013-12-04 24:00	2014-02-21 00:00 2014-03-19 24:00
Total packets received	71536	135458
Packet delivery ratio	90.52%	87.84%
Total cycles	1536	2588
Total path groups	1069	1494
Path recovery ratio using SUM_m and XOR	71520/71536 (99.98%)	135411/135458 (99.97%)
Path group recovery ratio using SUM_m and XOR	1053/1069 (98.50%)	1473/1494 (98.59%)
Path recovery ratio using SUM_m alone	71520/71536 (99.98%)	135407/135458 (99.96%)
Path group recovery ratio using SUM_m alone	1053/1069 (98.50%)	1469/1494 (98.33%)

VII. CONCLUSIONS

We present a novel approach to WSN tomography for dynamic routing topology from piggy-back measurements. Formulated as a novel and interesting optimization problem, our approach is general and systematic, particularly suited for WSN deployments at harsh environments with severe resource constraints at sensor nodes. To the best of the authors knowledge, our work, originally presented in [21] and substantially extended in this paper, provides the first CS-inspired approach to address dynamic WSN path reconstruction. We devise a suite of algorithms to reconstruct per-packet routing path at the sink for both reliable and lossy non-synchronized WSNs. One unique strength of our algorithms is their capability to reconstruct loops in per-packet paths, which would be very helpful for WSN diagnosis and performance analysis of routing protocols. Extensive simulations of lossy WSNs with drastic routing dynamics are conducted in comparison with the recent methods MNT, PathFinder, and CSPR. The results reveal that our INS-RTR algorithm significantly outperforms all the three state-of-the-art methods. Furthermore, our approach and INS-RTR algorithm are thoroughly validated in a real-world outdoor WSN testbed for months with more than 200 thousand received packets, achieving successful path (group) reconstruction ratios of higher than 98%. In our future work, we plan to further extend our algorithms to deal with more complex routing dynamic where multiple new shortcuts are introduced in an individual packet routing path.

ACKNOWLEDGMENT

This work is supported in part by U.S. National Science Foundation under grants CNS-1252066 and CNS-1320132.

REFERENCES

[1] Y. Liu, K. Liu, and M. Li. *Passive diagnosis for wireless sensor networks*. IEEE/ACM Transactions on Networking, Vol. 18, No. 4, 2010.

[2] M. Keller, J. Beutel, and L. Thiele. *How was your journey?: uncovering routing dynamics in deployed sensor networks with multi-hop network tomography*. Proceedings of SenSys, 2012.

[3] J. Zhao and R. Govindan. *Understanding packet delivery performance in dense wireless sensor networks*. Proceedings of Sensys, 2003.

[4] Y. Yang, Y. Xu, X. Li, and C. Chen. *A loss inference algorithm for wireless sensor networks to improve data reliability of digital ecosystems*. IEEE Transactions on Industrial Electronics, vol. 58, no. 6, pp. 2126-2137, 2011.

[5] H. Nguyen and P. Thiran. *Using end-to-end data to infer lossy links in sensor networks*. Proceedings IEEE INFOCOM, 2006.

[6] Y. Lin, B. Liang, and B. Li. *Passive loss inference in wireless sensor networks based on network coding*. Proceedings IEEE INFOCOM, pages 1809–1817, 2009.

[7] G. Hartl and B. Li. *Loss inference in wireless sensor networks based on data aggregation*. Proceedings of IPSN, 2004.

[8] Y. Mao, F. R. Kschischang, B. Li, and S. Pasupathy. *A factor graph approach to link loss monitoring in wireless sensor networks*. IEEE J. Selected Area in Comm., 23(4):820–829, 2005.

[9] V. Shah-Mansouri and V. W. S. Wong. *Link loss inference in wireless sensor networks with randomized network coding*. Proceedings IEEE GLOBECOM, pages 1–6, 2010.

[10] X. Lu, D. Dong, Y. Liu, X. Liao, and L. Shanshan. *Pathzip: Packet path tracing in wireless sensor networks*. Proceedings of MASS, 2012.

[11] Y. Gao, W. Dong, C. Chen, J. Bu, G. Guan, X. Zhang, and X. Liu. *Pathfinder: robust path reconstruction in large scale sensor networks with lossy links*. The 21st IEEE International Conference on Network Protocols (ICNP), 2013.

[12] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis. *CTP: an efficient, robust, and reliable collection tree protocol for wireless sensor networks*. ACM Transactions on Sensor Networks (TOSN), vol. 10, no. 3, 2013.

[13] E. Candes, J. Romberg, and T. Tao. *Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information*. IEEE Transactions on Information Theory, 52(2):489–509, 2006.

[14] D. L. Donoho. *Compressed sensing*. IEEE Transactions on Information Theory, 52(4):1289–1306, 2006.

[15] M. Navarro, T. Davis, Y. Liang, and X. Liang. *A study of long-term WSN deployment for environmental monitoring*. Proceedings of PIMRC, September, 2013.

[16] M. Navarro, T. Davis, G. Villalba, Y. Li, X. Zhong, N. Erratt, X. Liang, and Y. Liang. *Towards long-term multi-hop WSN deployments for environmental monitoring: an experimental network evaluation*. Journal of Sensor and Actuator Networks, 2014.

[17] TinyOS [Online]. Available: http://www.tinyos.net.

[18] Z. Liu, Z. Li, M. Li, W. Xing, and D. Lu. *Path reconstruction in dynamic wireless sensor networks using compressive sensing.* In Proc. of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), 2014.

[19] P. Levis, N. Lee, M. Welsh, and D. Culler. *TOSSIM: accurate and scalable simulation of entire tinyos applications.* In IEEE SenSys, 2003.

[20] H.J. Lee, A. Cerpa, and P. Levis. *Improving wireless simulation through noise modelling.* Proceedings of IPSN, 2007.

[21] Y. Liang and R. Liu. *Routing topology inference for wireless sensor networks.* ACM Computer Comm. Review, Vol. 43, No. 2, pp. 22-27, April 2013.