Universal Path Tracing for Large-Scale Sensor Networks

Yi Gao, Wei Dong*, Xiaoyu Zhang, Wenbin Wu
College of Computer Science, Zhejiang University, China.
Zhejiang Provincial Key Laboratory of Service Robot, China.
Email: {*gaoyi, dongw, zhangxy1991, wuwb*}@*zju.edu.cn*

***Abstract*—Most sensor networks employ dynamic routing protocols so that the routing topology can be dynamically optimized with environmental changes. The routing behaviors can be quite complex with increasing network scale and environmental dynamics. Knowledge on the routing path of each packet is certainly a great help in understanding the complex routing behaviors, allowing effective performance diagnosis and efficient network management. We propose PAT, a *universal* sensornet path tracing approach. PAT includes an intelligent path encoding scheme that allows efficient decoding at the PC side. To make PAT more scalable, we propose techniques to *accurately* estimate the degree information by exploiting timing information, allowing more *compact* path encoding. Moreover, we employ subpath concatenation to infer excessively long paths with a high recovery probability. We carefully evaluate PAT's performance using testbed experiments and and extensive simulations with up to 4,000 nodes. Results show that PAT significantly outperforms existing approaches.**

I. Introduction

As an emerging technology that bridges cyber systems and the physical world, wireless sensor networks (WSNs) are envisioned to support numerous applications such as military surveillance, environmental monitoring, infrastructure protection, etc [1, 2]. In these networks, large numbers of tiny, low-power wireless sensing devices are self-organized, collecting the sensing data to a central sink in a multihop manner.

Most sensor networks employ dynamic routing protocols so that the routing topology can be dynamically optimized with environmental changes. The TinyOS CTP protocol [3] is an instance of dynamic routing protocol with which each node regularly estimates the expected number of transmissions (ETX) [4] to the sink and dynamically selects the next-hop forwarder with a minimum ETX along the path. Due to the dynamic routing scheme, a node could forward different packets to different next hops, although these packets have the same destination, i.e., the sink node.

The routing behaviors can be quite complex with increasing network scale and environmental dynamics: packets from a distant sensor node may follow numerous routing paths to the sink. Knowledge on the routing path of each packet is very useful in the following ways.

- It is important for understanding the dynamic routing topology, revealing the complex routing behaviors. Such a knowledge is essential for topology control, routing improvement, and load balance, enabling effective management and optimized operations for deployed WSNs consisting of a large number of unattended wireless sensor nodes [5].
- It is essential for deriving important system metrics, e.g., per-link loss ratio and per-link delays. Most existing works on link loss and delay monitoring [5–9] assume that the routing topology is given a priori. Most of them are restricted to *static* routing tree estimation, which is unrealistic and problematic in real-world WSN deployments where routing topology is time-varying due to wireless channel dynamics such as fading and interference [5]. The investigation into realistic and dynamic routing topology can significantly improve the values of the works on WSN loss/delay tomography.
- It is very useful for effective network diagnosis and analysis. Most existing works heavily rely on the obtained network topology for inference, allowing further performance diagnosis and analysis [10–13]. For example, PAD [10] relies on dynamic network topology for maintaining the inference model which encodes the internal dependencies among different network elements, for online diagnosis of an operational sensor network system. MAP [11] also relies on the dynamic network topology for defining the impact scope of a network event, allowing accurate measurement on the root causes of packet losses. Clearly, obtaining the accurate per-packet path in a lightweight manner will greatly improve the accuracy of the measurement and diagnosis results.

Attaching each packet with the routing path hop-by-hop is not an attractive approach due to the large message overhead for packets with long routing paths (perhaps with routing loops). It is hence desirable to instrument a small message overhead to each packet, and reconstruct the routing path at the sink with a high recovery ratio.

We expect that a good path tracing approach should ideally meet the following two requirements.

- **Universal**: Since a sensor network can be usually classified into periodic monitoring (where all nodes generate data packets, e.g., environmental monitoring), event-triggered (where only a few nodes generate data packets, e.g.,

This work is supported by the National Science Foundation of China (No. 61502417 and No. 61472360), Zhejiang Provincial Natural Science Foundation of China (No. LY16F020006), Zhejiang Provincial Key Research and Development Program (No. 2017C02044), CCF-Tencent Open Research Fund, and Fundamental Research Funds for the Central Universities (2016QNA5012, 2016FZA5010). *Wei Dong is the corresponding author.

tracking), and observer-initiated (where only the requested nodes generate data packets, e.g., user query) according to [14], a good path tracing approach should well support all these networks for different applications.
- **Scalable**: A good path tracing approach should have a high path recovery ratio (the fraction of packets with correctly recovered paths) for large-scale networks. Otherwise, a simple approach of attaching the node IDs would not incur too much overhead.

Over the recent years, path tracing (or path reconstruction) attracted much research attention and many efficient approaches have been proposed for sensor networks [15–18]. However, to the best of our knowledge, no approaches simultaneously satisfy the above requirements.

MNT [15] is neither universal nor scalable: it assumes a periodic monitoring sensor network and all nodes in the network periodically transmit data packets to the sink node; it is also very sensitive to packet losses and routing dynamics, impairing its scalability in large-scale networks. PathZip [16] is universal, but it is not scalable since the time complexity of its path recovery algorithm grows exponentially with increasing path length. Pathfinder [17] is scalable to networks upto 900 nodes [17], but it is not universal since it assumes a periodic monitoring sensor network.

The main contribution of this paper is a sensornet path tracing approach that satisfy both the above requirements. We present a *novel* path tracing approach called PAT (PAth Tracing) which includes several versions with increasing recovery capability. The basic version (bPAT) attaches to each data packet a field called pathvalue which is updated hop-by-hop and encodes path information towards the sink. When a packet is received at the sink, the attached pathvalue can be efficiently decoded to recover the routing path. The pathvalue, however, can overflow (e.g., when the routing path is excessively long), impairing PAT's effectiveness. This problem is addressed from two perspectives. The dynamic version (tPAT) *accurately* estimates each node's in-degree (which is time-varying) by exploiting timing information, allowing more *compact* path encoding. The extended versions (xbPAT and xtPAT) try to infer a long path by concatenating two short subpaths. The key insight of PAT for inferring such a path (e.g., A to S) is to let an intermediate node on the long path (e.g., B) generate a local packet towards the sink. It is highly probable that the two local packets will follow the same path towards the sink since the routing topology keeps stable during a short period in most cases. PAT can thus obtain the long path by concatenating two short subpaths (i.e., A to B and B to S).

PAT is *universal* (i.e., can better support periodic monitoring, event-triggered, and obverver-initiated sensor networks) since it relies on *compact* encoding and decoding for path reconstruction, unlike MNT and Pathfinder which exploit *inter-packet* correlation and heavily depend on the existence of locally generated packets at the forwarder nodes. PAT *scales* to larger networks than previous approaches since it adopts subpath concatenation for long path inference when pathvalue overflows.

The rest of this paper is structured as follows. Section II describes the related work. Section III presents the design of our approach. Section IV shows the evaluation results, and finally, Section V concludes this paper.

II. Related Work

Sensornet path tracing attracts significant research attentions. PAD [10] adds to each packet a two-byte field which is used to store one forwarder along the path. For an h hop path, PAD needs h packets to assemble the entire path. Although PAD is lightweight, it cannot accurately assemble the routing path with high routing dynamics.

PathZip [16] adds a fix-cost hash value to each packet, containing compressed path information. In order to reconstruct the routing path for each packet, (1) the set of neighboring nodes for each node must be known a priori; (2) the PC side algorithm performs exhaustive search over neighbors for all nodes along the path for a match with the hash value in the packet. The computational overhead grows exponentially with increasing path length. In other words, given a limited computation time for each packet, the recovery ratio of per-packet path will be low for large-scale networks. Compared with PathZip, PAT employs an efficient encoding scheme that allows fast decoding at the sink. PAT additionally employs subpath concatenation, making it much more scalable than PathZip.

MNT [15] exploits information in the packet header (e.g., the first-hop forwarder or parent for short) to reconstruct the routing path with the assumption that all nodes in the network generate local traffic. The key insight of MNT is to utilize the parent information in packets originated from a given node to infer routing paths of packets passing the node. MNT essentially exploits *inter-packet* correlation for path recovery, i.e., a forwarded packet and its adjacent packets locally generated at the forwarder are transmitted to the same next hop. MNT has a low message overhead. However, the recovery ratio degrades in large-scale networks with decreasing packet delivery ratio and increasing routing dynamics (i.e., frequent parent changes). MNT requires all network nodes to periodically generate local packets, hence is not suitable for event-triggered or observer-initiated networks. Different from MNT, PAT exploits efficient and compact path encoding and decoding, instead of inter-packet correlation. Therefore, PAT does not require periodic traffic from all network nodes.

Pathfinder [17] is a robust path reconstruction method. At the node side, Pathfinder exploits temporal correlation between a set of packet paths and efficiently compresses the path information using path difference. At the PC side, Pathfinder infers packet paths from the compressed information and employs intelligent path speculation to reconstruct the packet paths with high reconstruction ratio. Similar to MNT, Pathfinder also exploits inter-packet correlation. Different from MNT, Pathfinder tolerates certain degree of differences in *inter-packet* correlation by *explicitly* recording the path difference. PAT differs from Pathfinder in two important ways. First, PAT is *universal* while Pathfinder is only suitable for periodic monitoring sensor networks since Pathfinder requires all nodes to periodically generate local packets. Second, PAT performs better than Pathfinder in large networks with high routing dynamics. In such networks, the path difference between two packets is inherently large, exceeding the tolerance capacity in Pathfinder.

CSPR [18] employs a compressive sensing based approach for path reconstruction in sensor networks. CSPR first classifies packets into different groups with each group containing the packets traveling the same path. It then decodes the path using

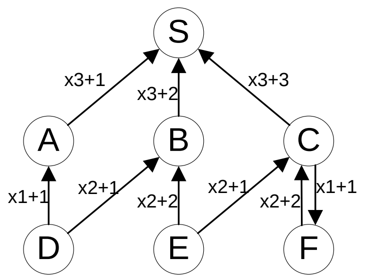


Fig. 1: PAT's basic idea. The pathvalue is initialized to 0 at the original node. It is then updated using multiplication and addition according to the rule associated with each in-edge.

compressive sensing when a sufficient number of packets in the group are accumulated. CSPR's classification accuracy highly depends on the size of the bloom filter for path recovery. For successful recovery, CSPR requires a sufficient number of packets in each group, i.e., $ck\log(N/k)$, where $c = 1.5$, N is the total network size, and k is the path length [18]. This number can be quite large for large-scale networks, which means that the path cannot be recovered if it is less frequently taken. PAT does not suffer from the above problem, and performs much better than CSPR, especially in large-scale network with long routing paths.

III. Design

We assume a sensor network where some nodes generate local packets to a central sink. We would like to trace the routing paths of those data packets with a small and bounded message overhead.

PAT contains a series of versions with increasing recovery capability at the cost of slightly increasing message overhead. We will describe different versions in the following subsections.

A. bPAT: the basic version

Figure 1 shows the basic idea of PAT. With PAT, data packets are instrumented with a field called pathvalue which is updated at each forwarder along the routing path towards the sink. Each forwarder updates the pathvalue according to its in-edges. If there are n in-edges and packets are coming from the i-th edge ($1 \le i \le n$), the pathvalue is updated to: pathvalue $\times n + i$. In Fig. 1, we annotate each edge with an update rule "$\times n + i$".

For example, we consider routing paths from node D. The pathvalue is initialized as 0 at D. If the routing path is DAS, the pathvalue is updated as 1 at A, and is updated as 4 at the sink S. If the routing path is DBS, the pathvalue is updated as 1 at B, and is updated as 5 at S.

The update rules can not only work for acyclic paths as illustrated above, but also work for cyclic paths. Consider the routing paths from node F. If the routing path is FCS, the pathvalue is updated as 2 at C, and is updated as 9 at S. If the routing path is FCFCS, the pathvalue is updated subsequently as 2 (at C), 3 (at F), 8 (at C), and finally 27 at S. We see that the different pathvalues can differentiate different paths in this example.

In general, we can formally prove the following theorem.

Theorem 1: PAT ensures a unique pathvalue for each routing path.

Algorithm 1 Algorithm for decoding a pathvalue

```
1: procedure DECODE(sink, pathvalue)
2:     P=(sink)
3:     k=sink
4:     while pathvalue > 0 do
5:         i=(pathvalue-1) % n(k) + 1
6:         k=f(i)
7:         P=P ++ k // ++ means concatenation
8:         pathvalue = (pathvalue-i) / n(k)
9:     return reverse P
```

Proof: We consider two different paths p and p':

$$\begin{aligned} p &: n_k \to \ldots \to n_{i+1} \to n_i \to \ldots \to n_0 = S \\ p' &: n'_j \to \ldots \to n'_{i+1} \to n_i \to \ldots \to n_0 = S \end{aligned} \tag{1}$$

The two paths share a common sub-path starting from n_i ($i \ge 0$). The previous hop nodes of n_i are different for the two paths. There are two cases: (a) Either $k = i$ or $j = i$ (but not both since in this case two paths are the same) (b) $k \ge i+1$, $j \ge i+1$, and $n_{i+1} \ne n'_{i+1}$. We will prove that for both cases, the pathvalues at n_i are different, i.e., $v_i \ne v'_i$.

(a) Without loss of generality, we assume $k = i$. In this case, $v_i = 0$ since the packet originates from n_i. On the other hand, $v'_i > 0$ since the pathvalue is updated at least once. Therefore, $v_i \ne v'_i$.

(b) We show that $v_i \ne v'_i$ by contradiction. Assume the number of inedges for n_i is n. $n_{i+1} \to n_i$ is the m-th edge while $n'_{i+1} \to n_i$ is the m'-th edge ($1 \le m, m' \le n$). If $v_i = v'_i$, $v_i - 1 \equiv v'_i - 1 \pmod{n}$. According to the update rules, $v_i = v_{i+1} \times n + m$ and $v'_i = v'_{i+1} \times n + m'$ ($1 \le m, m' \le n$). Therefore, $v_i - 1 \equiv m - 1$ (mod n) and $v'_i - 1 \equiv m' - 1 \pmod{n}$. We can get $m - 1 \equiv m' - 1 \pmod{n}$. Since $1 \le m, m' \le n$, it must be the case that $m = m'$ which contradicts with the fact that $m \ne m'$.

Starting from n_i, p and p' have the same sequence of nodes. Therefore, the pathvalues are updated according to the same set of update rules. We will prove that $v_0 \ne v'_0$ given that $v_i \ne v'_i$.

1) $v_i \ne v'_i$ is already proved.
2) We will show that $v_{i-1} \ne v'_{i-1}$. Suppose the update rule from n_i to n_{i-1} is "xh+l". $v_{i-1} = h \times v_i + l$ and $v'_{i-1} = h \times v'_i + l$. Therefore, $v_{i-1} - v'_{i-1} = h \times (v_i - v'_i) \ne 0$, i.e., $v_{i-1} \ne v'_{i-1}$.
3) From 1 and 2, $v_0 \ne v'_0$.

Hence, the theorem holds. ■

The decoding process at the PC side needs two kinds of information: 1) The number of in-edges of each node (i.e., in-degree). We use $n(k)$ to denote the in-degree of node k. 2) For each node, there exists a one-to-one mapping between the i-th in-edge and the corresponding incoming node ID. We use $f(i)$ to denote the node ID corresponding to the i-th edge and $f^{-1}(k)$ to denote the index of the in-edge corresponding to node k.

Algorithm 1 shows the decoding algorithm. We will use an example to show the working details of the algorithm. Suppose at the sink node, we get a pathvalue of 27. The path P is initialized to (S).

- k=S. We would like to determine which node forwards the packet to the current node k=S. The index is calculated as (27-1) % 3 + 1 = 3 which corresponds to node C. Therefore, k=C, P=(S,C), and pathvalue= (27-3)/3=8.
- k=C. Again we would like to determine C's previous hop. The index is calculated as (8-1) % 2 + 1 = 2 which corresponds to node F. Therefore, k=F, P=(S,C,F), and

pathvalue=(8-2)/2=3.

- k=F. The index is (3-1) % 1 + 1 = 1 which corresponds to node C. Therefore, k=C, P=(S,C,F,C), and pathvalue=(3-1)/1=2.
- k=C. The index is (2-1) % 2 + 1 = 2 which corresponds to node F. Therefore, k=F, P=(S,C,F,C,F), and pathvalue=(2-2)/2=0.
- The while loop terminates since pathvalue is no longer larger than 0, and the decoded path is obtained by reversing P. The resulting path is hence FCFCS.

The encoding and decoding scheme of PAT has some nice properties: 1) The update rules can be locally determined by a node. 2) A pathvalue can uniquely determine a routing path. In contrast to PathZip which uses exhaustive search in the entire space of possible routing paths, PAT has an efficient decoding algorithm (constant overhead per hop), enabling fast packet path tracing.

There are, however, two challenging issues concerning PAT's scalability. First, how to estimate the degree and index information and how it can be reliably collected at the sink? Second, how to handle the case when the pathvalue overflows? bPAT addresses these two issues in simple ways.

First, node k can locally obtain its degree information. A network usually employs broadcasting for routing initialization and maintenance. Each node k can find all its direct neighbors that k can hear. The i-th found neighbor is assigned with index i. Each node can locally obtain the above information and transfer the information to the central sink after the network initialization phase. There are chances that the message drops along the path towards the sink. We employ NAK mechanism at the sink: if the sink does not receive the information from a node, it will transmit an NAK to the node, using the *reverse* path which can be setup in all nodes along the collection path when a packet is sent from the node to the sink. Therefore, the node can be informed to retransmit the information again till the sink receives the information.

Second, bPAT uses the most significant bit in pathvalue to record whether the pathvalue overflows. bPAT cannot recover path of a data packet with an overflown pathvalue. In the initialization phase, pathvalues are intentionally marked as overflown.

There are indeed more intelligent ways to tackle these two problems at the cost of slightly increasing (but still small and bounded) overhead. Section III-B introduces tPAT which exploits packet timing information to get a more accurate estimate of node in-degrees. Section III-C introduces xPAT (including xbPAT and xtPAT) which uses subpath concatenation for inferring excessively long path.

B. tPAT: the dynamic version exploiting timing information

As mentioned in the previous subsection, we need two kinds of information for each node k: (1) $n(k)$: node k's in-degree. (2) $f(i)$: a mapping from the index of the in-edge to the corresponding node ID.

In bPAT, each node k estimates $n(k)$ as the number of all its direct neighbors that k can hear. It is not *accurate* since it considers all direct neighbors that k can hear, some of which never forward packets to k. For example, in Fig. 1, node B's direct neighbors include all other nodes shown in the figure. However, only nodes D and E forward packets to B. If B takes into account in all neighbors, the number of in-edges of B will be 6 while the actual number of in-edges is 2. Overestimating the in-degree of a node will waste the encoding space, causing the pathvalue to overflow more easily.

Instead of relying on *control-plane* broadcasting packets to find all incoming neighbors, tPAT exploits *data-plane* packets: each time a node receives a data packet from a new direct neighbor, it adds the new neighbor into its neighbor table. In-degree estimation based on data-plane packets can avoid the overestimation issue, resulting in more efficient use of the encoding space.

A further issue is that the convergence time can be long since there is no explicit initialization phase to explore all possible in-edges in existing collection protocols (e.g., CTP). To address this issue, tPAT encodes path information based on the *current status* of the node, and further exploits timing information to recover as many as possible packet paths during the convergence time.

When a data packet is received, a node updates the pathvalue of the forwarded packet using the current neighbor table size and the neighbor's index. The degree and index information at each node is hence *time-varying*. In order to estimate the time-varying information at the sink side, tPAT needs timing information of data packets' arrivals as well as timing information of how the neighbor table evolves at each node.

tPAT needs to record the following additional information:

- For each data packet d, the arrival time at the sink $t_s(d)$ is measured on a reference clock and known for any packet d. We assume an estimate of the packet generation $\hat{t}_g(d)$ is accessible for all packets. For example, we can attach an additional field in the data packet, recording the network sojourn time of the packet $s(d)$. The packet generation time can thus be estimated as $t_s(d) - s(d)$. The error of this estimate is bounded by $\hat{t}_g(d) - \Delta^l(d) \leq \hat{t}_g(d) \leq \hat{t}_g(d) + \Delta^u(d)$. Here $\Delta^l(d), \Delta^u(d)$ denote the upper and lower bounds on the error of the time-stamping mechanism used. They are not transmitted as part of a packet d, but assumed to be derived from an analysis, e.g., [19].
- When a neighbor is added, the node generates an update message $c = \langle \text{origin, node, index, } \hat{t}_g(c) \rangle$, where origin denotes the node generating the update message, node denotes the added neighbor, index denotes the index of the added neighbor, and $\hat{t}_g(c)$ is the estimate of the time when the neighbor is added. Similar to data packets, we assume the estimation error of $\hat{t}_g(c)$ is bounded by $\hat{t}_g(c) - \Delta^l(c) \leq \hat{t}_g(c) \leq \hat{t}_g(c) + \Delta^u(c)$.

There are some important points worth mentioning here. First, we note that the update message is only generated when a data packet from a *new* neighbor is received. If a node repeatedly receives data packets from a given neighbor already in the neighbor table, no update message will be generated except for the first one. Second, the update message does not need to be immediately transferred to sink. As long as the update message can eventually be received with timestamp $\hat{t}_g(c)$, we can infer the node status for a given time. Third, the update message format can be further extended to handle neighbor deletions, e.g., when a node fails in a network. Hence, tPAT is more suitable for dynamic networks with node additions and deletions. Finally, the transmissions of update messages can be aggregated to reduce transmission overhead. However, we separately consider each update message for the clarity of presentation.

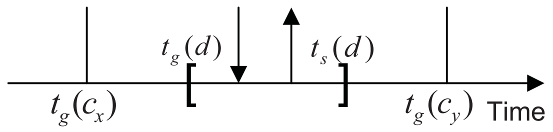


Fig. 2: Method for inferring time-varying degree and index information at a node k.

Figure 2 illustrates how our method can infer the time-varying degree and index information. The figure shows the packet generation and reception activities at a node, say k. The x-axis denotes the time with respect to sink. Node k generates two update messages c_x, c_y (where the subscripts denote the sequence numbers starting from 1) with the $\hat{t}_g(c)$ values indicated. During the two update messages, node k also receives a data packet d. Although we do not know the *exact time* when packet d arrived at node k (except that k is the sink), we do know that the arrival time is bounded between the packet's generation time at the original node ($t_g(d)$) and its arrival time at the sink ($t_s(d)$). There are two cases where we can determine the degree and index information applied to packet d.

- c_x is the last update message from k (i.e., node k never issues c_y) and $t_g(c_x) < t_g(d)$. Considering that we only know the estimated times, a further sufficient condition is: $\hat{t_g}(c_x) + \Delta^u(c_x) < \hat{t_g}(d) - \Delta^l(d)$.
- There exists $c_y : y = x+1$ and $t_g(c_x) < t_g(d) < t_s(d) < t_g(c_y)$. Considering that we only know the estimated generation times, a further sufficient condition is: $\hat{t_g}(c_x) + \Delta^u(c_x) < \hat{t_g}(d) - \Delta^l(d)$ and $t_s(d) < \hat{t_g}(c_y) - \Delta^l(c_y)$.

In both cases, we can guarantee that the degree and index information applied to packet d at node k can be inferred accurately by considering c_1 to c_x. We say that the packet d's path can be decoded at node k. If packet d's path can be decoded at all nodes involved in the decoding process, d's entire routing path can be decoded.

Suppose a data packet's path is currently decoded at node k (it has been decoded backward from the sink), we need to determine the degree and index information at k that has applied to d's pathvalue in order to determine k's previous hop. As discussed above, if *either* of the two conditions holds, we can determine the corresponding $n(k)$ and $f(i)$ by assembling c_1 to c_x.

If c_1 to c_x are all correctly received at the sink, $n(k) = c_x$.index and $f(i) : f(c_j.\text{index}) = c_j.\text{node}, 1 \le j \le x$.

However, if there are losses, there might be ambiguities in determining $n(k)$ and $f(i)$. Our approach to addressing this problem is to employ a reliable transfer mechanism. Basically speaking, tPAT employs an NAK mechanism like bPAT: the sink will transmit a NAK if losses are detected, informing the node to retransmit the information again till the sink receives the information.

C. xbPAT and xtPAT: the extended versions employing subpath concatenation

So far, we have only considered the case when the pathvalue never overflows. Both bPAT and tPAT cannot decode overflown pathvalue. Indeed, for long path with high in-degree forwarders, it is possible that the pathvalue overflows, causing decoding failures at the sink.

We can extend the pathvalue field to accommodate complete information about the path. However, the pathvalue field might be very long for some packets. It is favorable to devise an approach with small and bounded overhead, and it can also recover very long path with a high probability.

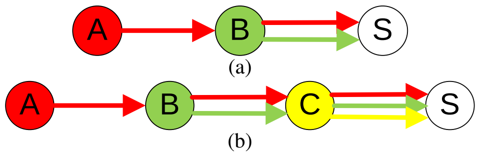


Fig. 3: The idea of subpath concatenation.

Figure 3 illustrates cases in which the pathvalue from A to S will overflow. The basic idea of our approach is to let an intermediate node on the path, e.g., B, generate a local packet. If path from A to B's previous hop can be recovered and path B to S can be recovered, path A to S can also be recovered by concatenating the two paths if packets from A and B follow the same path to the sink. Note that we do not require that path B to S is directly recovered via pathvalue, it can also be recovered using another intermediate node. Figure 3(a) illustrates the case in which B to S can be directly decoded. Figure 3(b) illustrates the case in which the recovery of B to S depends on another intermediate node, C. Hence the above recovering process can be performed in an iterative manner, resulting in excellent scalability.

In order to realize the above idea, three key issues need to be addressed.

(1) Which intermediate node is the helper node for generating a local packet?

When a data packet is forwarded in the network, PAT updates the pathvalue hop-by-hop. If the pathvalue never overflows, no helper node is needed. Otherwise, the first node that experiences overflow is identified as the helper node for generating a local packet.

(2) How to recover the path from origin to the previous hop of the helper node?

The helper node performs the following tasks on the data packet from origin: (a) keeps the non-overflown pathvalue unchanged (i.e., do not update pathvalue). (b) records an additional field called anchor which is the previous hop of the helper node for the forwarded data packet. Using these two fields, the path from origin to anchor can be recovered by performing decoding starting from anchor. For example, we can invoke DECODE (shown in Algorithm 1) with anchor and pathvalue as parameters to recover the partial path from origin to anchor.

(3) How to verify that packet from the helper node follows the same path as the original node towards the sink?

The helper node will attach to the data packet from origin a crc field which will be used to record the CRC value of the subpath from the helper node to the sink. It will be used to verify that the local packet from the helper node follows the same path. Consider A is the origin node. Let Path(A..S) denote the path of packet from A, and Path(helper..S) denote the path of packet from the helper. We consider Path(A..S) = Path(A..anchor) ++ Path(helper..sink) if the recorded crc field in A's packet equals to CRC(Path(helper..sink))—the calcuated CRC value of the known path. Note that Path(helper..sink) can be recovered either directly or indirectly relying on another helper node.

Algorithm 2 shows the algorithm for recovering a pathvalue which will overflow with subpath concatenation. We will use

Algorithm 2 Inferring path with overflown pathvalue

```
1:  procedure INFER(pathvalue, anchor, crc)
2:      P1=DECODE(anchor, pathvalue)
3:      for pkt in time window do
4:          if pkt non-overflow then
5:              P2=DECODE(sink, pkt.pathvalue)
6:          else
7:              P2=INFER(pkt.pathvalue, pkt.anchor, pkt.crc)
8:          if crc == CRC(P2) then
9:              return P1++P2 // ++ means concatenation
10:     // cannot recover the full path
```

the example shown in this section to show its working details.

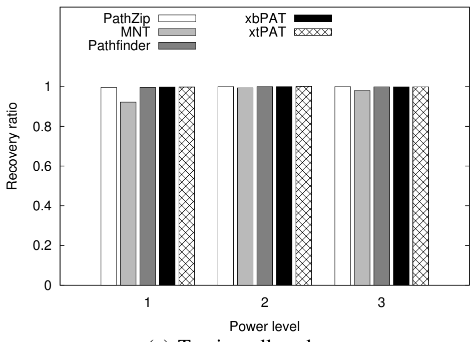
In Figure 3(a), A generates a data packet. The pathvalue in A's packet will be updated hop-by-hop towards the sink. An intermediate node on the path, B, detects that the pathvalue will overflow if it is further updated at B. B keeps the pathvalue unchanged, and at the same time, attaches two fields to A's packet: the anchor field records B's previous hop while the crc field will be the CRC value of the subpath starting from B to S. With the anchor field and the non-overflown pathvalue field, PAT can recover the subpath from A to B's previous hop. B will also generate a local packet towards the sink. It is assumed that if the transmission times of the two packets (from A and from B) are close, B's packet will follow the same path as A's packet with a high probability. The path from B to S can be directly recovered since B's packet does not experience overflow in pathvalue. PAT verifies if the recorded crc in A's packet equals to the calculated CRC values of the known path from B to S. If it is the same, PAT considers the path from A to S is the concatenation of the path from A to anchor (i.e., B's previous hop) and the path from B to S.

Figure 3(b) shows a more complicated case in which the path from B to S cannot be directly recovered. In this case, the local packet from B will trigger another helper node, C, to generate a local packet. The path from B to S can be recovered in a process similar to the one we have described above. If the path from B to S is recovered, the path from A to S can be recovered if the recorded crc in A' packet equals to the calculated CRC value of the known path from B to S.

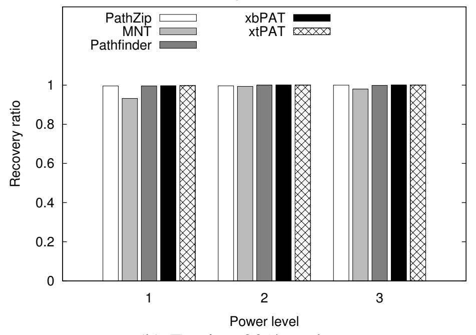
We note here a few important features of the extended versions: 1) Two fields (anchor and crc) are attached to the data packet when the pathvalue will overflow without subpath concatenation. It also requires the helper nodes to generate local packets. However, the attached overhead to each data packet is still small and bounded. 2) The use of subpath concatenation achieves a much stronger path recovery capability. First, it can recover long path recursively. Second, even if the full path cannot be recovered (e.g., the crc values do not match), it can at least recover a partial path from origin to anchor.

IV. EVALUATION

In this section, we evaluate the performance of PAT with state-of-the-art approaches. Section IV-A shows the evaluation results based on a 80-node indoor testbed. In order to see the performance of PAT with various network parameters, we perform TOSSIM simulations in Section IV-B. Since TOSSIM does not run efficiently for a large scale network exceeding 1,000 nodes, we perform a simulation study based on our own simulator in Section IV-C. By using this simulator, we evaluate the performance of various approaches in networks with up to 4,000 nodes.



(a) Tracing all nodes



(b) Tracing 20% nodes

Fig. 4: Recovery ratios of the testbed experiments.

A. *Testbed experiment*

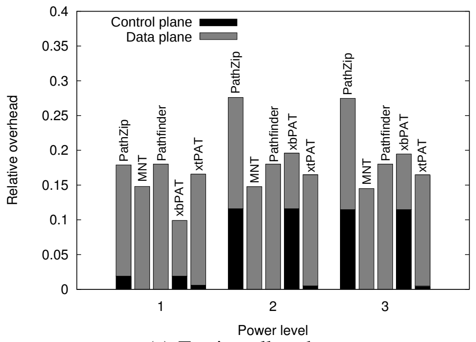
We implement PAT on TelosB nodes and use a 80-node testbed to validate its design. In order to see the impact of different network densities, we configure the transmission power level of each node to be 1, 2, 3 which correspond to -32.5dBm, -28.7 dBm, and -25 dBm, respectively.

Each node employs the CTP protocol [3] to transmit data packets to the sink node with a period of 1 min. The payload length of the data packet fixed at 70 bytes. We obtain the ground truth by analyzing the logs collected from each node's external flash.

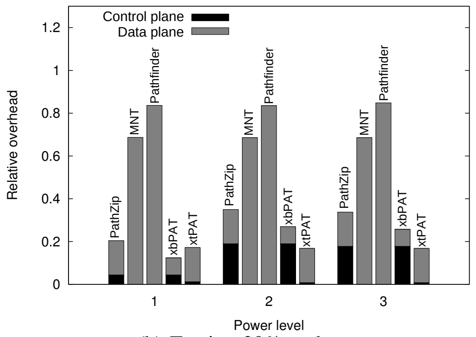
We conduct two sets of experiments. The first set of experiments are used to evaluate the performance of different approaches for tracing paths from all nodes, i.e., all nodes generate local data packets. The second set of experiments are used to evaluate the performance of different approaches for tracing only 20% nodes, i.e., only 20% nodes generate local data packets. We note that both MNT and Pathfinder are *not* suitable for event-triggered or observer-initiated networks where only a small number of nodes generate local data packets. In order to let MNT and Pathfinder work, all nodes must explicitly generate packets periodically. The control information is piggybacked in locally generated packets whenever possible. If there are no locally generated data packets, separate control packets are transferred to the sink node. Each experiment lasts for one hour.

Recovery ratios of testbed experiments. Figure 4(a) shows the recovery ratio for tracing all nodes with power levels 1, 2, and 3. and Figure 4(b) shows the recovery ratio for tracing 20% nodes with power levels 1, 2, and 3. We can see that the recovery ratios for all approaches (MNT, PathZip, Pathfinder, xbPAT and xtPAT) are high since network scale of the testbed is relatively small.

Overhead. Existing path tracing approaches may introduce two kinds of overhead: *data-plane overhead* which is the



(a) Tracing all nodes



(b) Tracing 20% nodes

Fig. 5: Relative overhead of the testbed experiments.

overhead added to each data packet. *control-plane overhead* which is the overhead carried in separate control packets, e.g., update and helper packets in xtPAT.

Analysis of data-plane overhead. We assume each packet already includes the CTP header (including origin, sequence number, hop count). We assume two bytes are used to represent a node ID. The message overhead of MNT is six bytes, including the parent ID, and the 4-byte timestamp. PathZip has an overhead of 8-byte MD5-like hash value. Pathfinder has a variable overhead with a maximum overhead of 9 bytes including parent node ID (2 bytes), XOR-byte (1 byte), bit vector ($\leq$2 bytes) and container ($\leq$4 bytes).

For PAT, the pathvalue field is 4 bytes, the anchor field is 2 bytes, and the crc field is 2 bytes. PAT's basic version, bPAT, requires the smallest message overhead, i.e., 4 bytes of pathvalue. The full-fledged version, xtPAT, requires the largest message overhead. For data packets without overflow, the overhead is 8 bytes, including 4 bytes of pathvalue and 4 bytes of timestamp. It is also mentioning that the 4-byte timestamp is useful for accurate delay measurement [20].

Relative overhead of testbed experiments. We define the *relative overhead* as the percentage of the generated data-plane overhead and control-plane overhead over all generated packet overhead (including useful data payload). Figure 5(a) shows the relative overhead for tracing all nodes. We can see that (1) The overhead of PAT is comparable to existing approaches, e.g., Pathfinder. (2) In dense networks (e.g., power levels 2 and 3), the control-plane overhead of xbPAT is much larger since each node regards many nodes as its neighbors. (3) The data-plane overhead of xtPAT is much larger than xbPAT as xtPAT additionally attaches a 4-byte timestamp field. However, xtPAT's control-plane overhead is much smaller than xbPAT since each node only reports possible child nodes (i.e., previous hop nodes) to the sink node.

Figure 5(b) shows the relative overhead for tracing 20% nodes. In this scenario, MNT and Pathfinder still require all nodes to transfer control packets (without useful data payload). Otherwise, they cannot decode the packet paths. Therefore, the relative overheads of MNT and Pathfinder are high. PathZip and PAT have much smaller relative overheads. xtPAT's relative overhead is the smallest since it generates the smallest control-plane overhead.

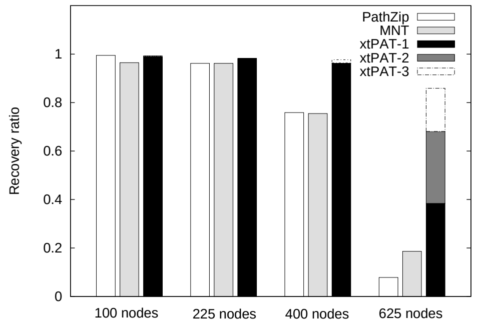


Fig. 6: Recovery ratios in TOSSIM simulation.

B. TOSSIM simulations

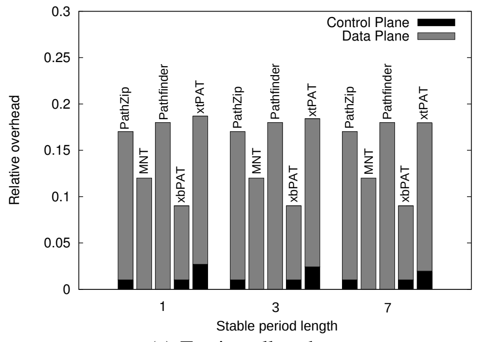
In this section, we use the TOSSIM simulator [21] to study the performance of various approaches with varying network parameters. TOSSIM is a high-fidelity simulator which can reuse the TinyOS code.

In the simulation, each node employs the CTP protocol to send data packets to a central sink. Different approaches attach different fields to data packets in order to perform path recovery at the PC side. The ground truth is obtained by inspecting the `dbg()` output from each node. We use the LinkLayerModel utility in the TinyOS distribution to generate different network configurations.

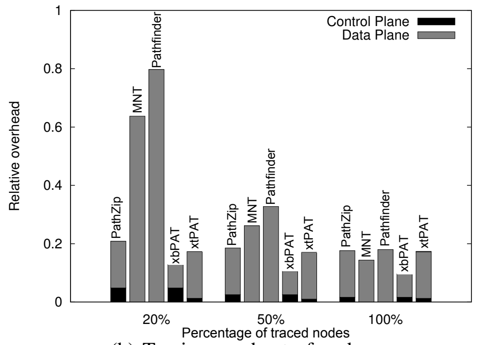
Recovery ratios. We compare performance of four existing approaches using these four network configurations with 100, 225, 400, and 625 nodes, from the simplest to the most complex. The number of nodes increases from 100 to 625. The average hop count increases from 3.4 to 11.9. The average node degree increases from 3.5 to 9.8. The average stable period length decreases from 37.6 to 1.5. The average link PRR decreases from 99.8% to 94.0%.

Figure 6 shows the recovery ratios of four approaches. We can see that for the first two simple cases, all approaches perform well. However, for the last two complex cases, xtPAT performs significantly better than the others. For simulation trace C with 400 nodes, xtPAT can recover 95.44% packet paths, and, for simulation trace D with 625 nodes, xtPAT can recover 73.35% full packet paths, with 36.37% decoded in the first phase (xtPAT-1) and 36.98% inferred in the second phase (xtPAT-2). In addition, 11.35% packets have partial paths recovered (xtPAT-3). All approaches have a zero error ratio in our simulations.

Overhead. Figure 7(a) shows the relative overhead for the TOSSIM experiments. The simulation consists of 225 nodes and we configure the simulator to exhibit different network dynamics (stable period length equals to 1,3,7). We can see that xtPAT's control-plane overhead increases in more dynamic networks (with smaller stable period length). This is because in more dynamic networks, each node has more child nodes, resulting in more update messages. For xbPAT, the relative overhead does not change since it has a separate initialization phase to discover all neighbors. We also compare different



(a) Tracing all nodes



(b) Tracing a subnet of nodes

Fig. 7: Relative overhead of TOSSIM experiments.

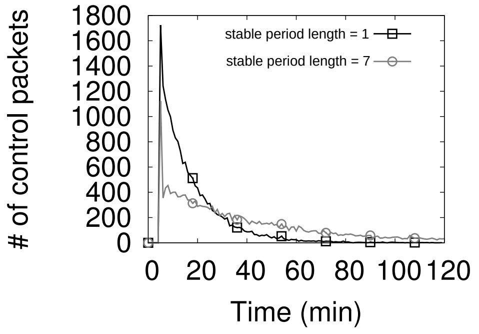


Fig. 8: Control-plane overhead of xtPAT.

approaches for tracing a subnet of nodes in a 225-node network with a stable period length of 3. Figure 7(b) shows the relative overhead of different approaches for tracing 20%, 50% and 100% nodes. We can observe both MNT and Pathfinder have high overheads, making them unsuitable for event-triggered and observer-initiated networks where only a small subnet of nodes generate local data packets. On the other hand, both PathZip and PAT incur small overhead.

Figure 8 shows the distribution of xtPAT's control packets over time (for tracing all nodes). We can that in more dynamic networks, the number of control packets is high in the initial phase. However, it becomes low afterwards. This is reasonable because with high routing dynamics, xtPAT can find more child nodes in the initial phase and it quickly converges afterwards.

In order to see PAT's overhead in networks with node deletions and additions, we perform an experiment based on a 225-node network for 4 hours. At the end of each hour, we delete 10% nodes and add 10% new nodes to the network. This operation represents regular network maintenance where a small set of nodes are replaced with new nodes. Figure 9 shows the number of control packets varied with time. We can see

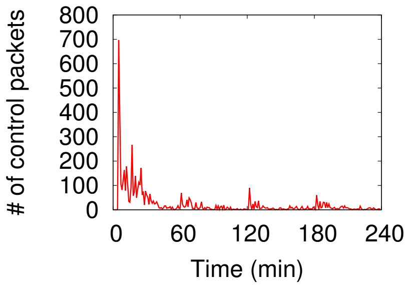


Fig. 9: xtPAT's overhead under node deletions and additions.

that node deletions/additions introduce extra overhead to xtPAT since additional update messages are required. This overhead is relatively small as long as the number of replaced nodes is small. Moreover, the control overhead quickly decreases when the network becomes steady again.

C. Large-scale simulations

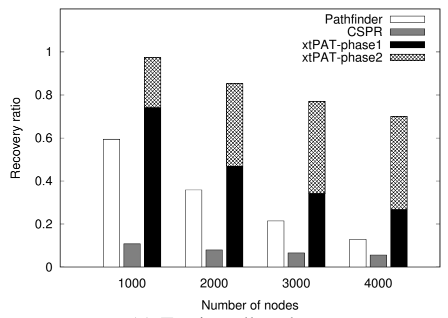
Since TOSSIM cannot efficiently support a large scale network exceeding 1000 nodes, we write our own simulator in C++ to study the performance of PAT in large scale networks. For comparison, we also implement Pathfinder and CSPR, the state-of-the-art path tracing approaches.

The data packet payload is fixed at 50 bytes. We vary the network scale to study its impact to the path recovery ratio and relative overhead. The stable period length is 3 in the simulation. The data packet generation period is 1min.

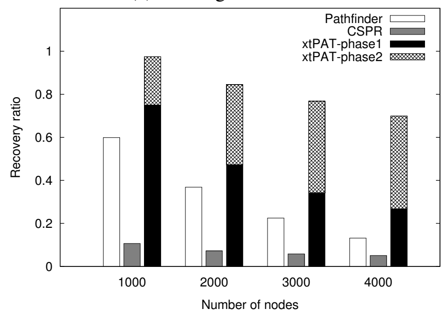
Figure 10(a) shows the recovery ratios for tracing all nodes with network sizes 1000, 2000, 3000 and 4000. We can see that xtPAT performs consistently better than Pathfinder and CSPR. The path concatenation phase plays a more important role in larger networks, e.g., it recovers 23.3%, 38.3%, 42.8%, 43.2% of all packet paths for network sizes of 1000, 2000, 3000 and 4000. Figure 10(b) shows the recovery ratios for tracing 20% nodes. Similarly, we can see that xtPAT performs much better than Pathfinder and CSPR.

Figure 11(a) shows the relative overhead with network sizes 1000, 2000, 3000 and 4000. xtPAT's overhead is larger than Pathfinder and CSPR, and the overhead due to helper packets increases with larger network scale. However, as Figure 11(b) shows, the relative overhead is much smaller than Pathfinder when only 20% nodes are required to be traced since Pathfinder requires all network nodes to periodically generate control packets for path reconstruction. CSPR introduces constant overhead in each packet. Therefore, its relative overhead does not change in different networks. However, as shown in Figure 10, the recovery ratio of PAT (above 70%) is much higher than CSPR (below 10%) in large scale networks.

It is worth noting that Pathfinder requires all network nodes to generate packets with a *common* period for achieving high recovery accuracy. PAT does not have this requirement. In order to compare the performance of PAT with Pathfinder when nodes have different inter-packet intervals (IPIs), we conduct a series of simulations with different levels of IPI randomness. IPIs are set in a random interval $[t-r, t+r]$ (seconds) where $t = 120$s in our simulation and r represents the randomness. When $r = 0$, all nodes have the same IPI. Table 1 shows the reconstruction results with varying IPI randomness controlled by r. When there is no IPI randomness, both PAT and Pathfinder achieve high reconstruction ratio. When the IPI

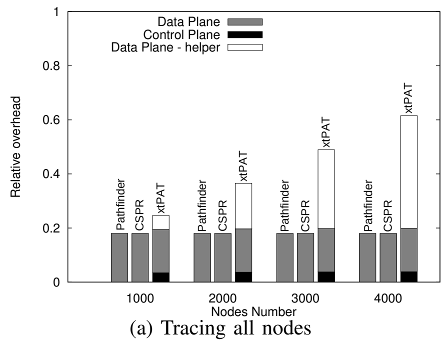


(a) Tracing all nodes

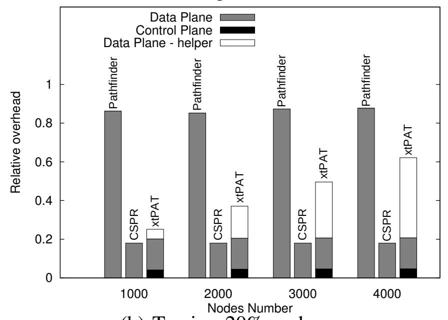


(b) Tracing 20% nodes

Fig. 10: Recovery ratios in large-scale simulations.



(a) Tracing all nodes



(b) Tracing 20% nodes

Fig. 11: Relative overhead in large-scale simulations.

Table 1: Path reconstruction ratios of Pathfinder and PAT when there exist IPI randomness.

r	Pathfinder	PAT
0 second	86.1%	97.5%
20 seconds	83.0%	98.8%
60 seconds	60.2%	99.1%
100 seconds	26.7%	98.3%

randomness increases, the reconstruction ratio of Pathfinder decreases rapidly. The reason is that in Pathfinder, each path is reconstructed with the help of a reference packet at each hop. When nodes have different IPIs, those reference packets cannot be accurately located, resulting in decreased performance.

V. Conclusion

In this paper, we propose PAT, a universal sensornet path tracing approach. PAT includes an intelligent path encoding scheme that allows efficient decoding at the PC side. To make PAT more scalable, we propose techniques to *accurately* estimate the degree information by exploiting timing information, allowing more *compact* path encoding. Moreover, we employ subpath concatenation to infer excessively long paths with a high recovery probability. We carefully evaluate PAT's performance using testbed experiments and extensive simulations. Results show that PAT significantly outperforms existing approaches.

References

[1] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and Yield in a Volcano Monitoring Sensor Networks," in *Proc. of USENIX OSDI*, 2006.

[2] A. Hasler, I. Talzi, C. Tschudin, and S. Gruber, "Wireless sensor networks in permafrost research - concept, requirements, implementation and challenges," in *Proc. 9th Int'l Conf. on Permafrost*, 2008.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proc. of ACM SenSys*, 2009.

[4] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-throughput Path Metric for Multi-hop Wireless Routing," in *Proc. of ACM MobiCom*, 2003.

[5] Y. Liang and R. Liu, "Routing topology inference for wireless sensor networks," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 2, pp. 22–27, 2013.

[6] H. X. Nguyen and P. Thiran, "Using end-to-end data to infer lossy links in sensor networks." in *Proc. of IEEE INFOCOM*, 2006.

[7] G. Hartl and B. Li, "Loss inference in wireless sensor networks based on data aggregation," in *Proc. of ACM/IEEE 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," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 4, pp. 820–829, 2005.

[9] L. Ma, T. He, K. K. Leung, D. Towsley, and A. Swami, "Efficient identification of additive link metrics via network tomography," in *Proc. of IEEE ICDCS*, 2013.

[10] K. Liu, M. Li, Y. Liu, M. Li, Z. Guo, and F. Hong, "Passive diagnosis for wireless sensor networks," in *Proceedings of SenSys*, 2008.

[11] W. Dong, Y. Liu, Y. He, T. Zhu, and C. Chen, "Measurement and analysis on the packet delivery performance in a large-scale sensor network," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 1952–1963, 2014.

[12] U. Javed, I. Cunha, D. R. Choffnes, E. Katz-Bassett, T. Anderson, and A. Krishnamurthy, "PoiRoot: Investigating the Root Cause of Interdomain Path Changes," in *Proc. of ACM SIGCOMM*, 2013.

[13] M. Keller, J. Beutel, and L. Thiele, "The Problem Bit," in *Proc. of DCOSS*, 2013.

[14] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless micro-sensor network models," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 2, pp. 28–36, 2002.

[15] M. Keller, J. Beutel, and L. Thiele, "How Was Your Journey? Uncovering Routing Dynamics in Deployed Sensor Networks with Multi-hop Network Tomography," in *Proc. of ACM SenSys*, 2012.

[16] X. Lu, D. Dong, X. Liao, and S. Li, "PathZip: Packet Path Tracing in Wireless Sensor Networks," in *Proc. of IEEE MASS*, 2012.

[17] Y. Gao, W. Dong, C. Chen, J. Bu, and X. Liu, "Towards reconstructing routing paths in large scale sensor networks," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 281–293, 2016.

[18] Z. Liu, Z. Li, M. Li, W. Xing, and D. Lu, " Path Reconstruction in Dynamic Wireless Sensor Networks Using Compressive Sensing," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–1, 2015.

[19] M. Keller, L. Thiele, and J. Beutel, "Reconstruction of the correct temporal order of sensor network data," in *Proc. of IPSN*, 2011.

[20] J. Wang, W. Dong, Z. Cao, and Y. Liu, "On the delay performance analysis in a large scale wireless sensor network," in *Proc. of IEEE RTSS*, 2012.

[21] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *SenSys*, 2003.