

iPath: Path Inference in Wireless Sensor Networks

Yi Gao, *Student Member, IEEE*, Wei Dong, *Member, IEEE*, Chun Chen, *Member, IEEE*,
Jiajun Bu, *Member, IEEE, ACM*, Wenbin Wu, and Xue Liu, *Member, IEEE*

Abstract—Recent wireless sensor networks (WSNs) are becoming increasingly complex with the growing network scale and the dynamic nature of wireless communications. Many measurement and diagnostic approaches depend on per-packet routing paths for accurate and fine-grained analysis of the complex network behaviors. In this paper, we propose iPath, a novel path inference approach to reconstructing the per-packet routing paths in dynamic and large-scale networks. The basic idea of iPath is to exploit high path similarity to iteratively infer long paths from short ones. iPath starts with an initial known set of paths and performs path inference iteratively. iPath includes a novel design of a lightweight hash function for verification of the inferred paths. In order to further improve the inference capability as well as the execution efficiency, iPath includes a fast bootstrapping algorithm to reconstruct the initial set of paths. We also implement iPath and evaluate its performance using traces from large-scale WSN deployments as well as extensive simulations. Results show that iPath achieves much higher reconstruction ratios under different network settings compared to other state-of-the-art approaches.

Index Terms—Measurement, path reconstruction, wireless sensor networks.

I. INTRODUCTION

WIRELESS sensor networks (WSNs) can be applied in many application scenarios, e.g., structural protection [1], ecosystem management [2], and urban CO₂ monitoring [3]. In a typical WSN, a number of self-organized sensor nodes report the sensing data periodically to a central sink via multihop wireless.

Recent years have witnessed a rapid growth of sensor network scale. Some sensor networks include hundreds even thousands of sensor nodes [2], [3]. These networks often employ dynamic routing protocols [4]–[6] to achieve fast adaptation to the dynamic wireless channel conditions. The growing network scale and the dynamic nature of wireless channel make WSNs become increasingly complex and hard to manage.

Reconstructing the routing path of each received packet at the sink side is an effective way to understand the network's com-

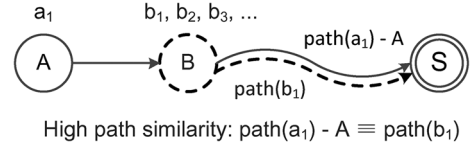


Fig. 1. Example to illustrate the basic idea of iPath.

plex internal behaviors [7], [8]. With the routing path of each packet, many measurement and diagnostic approaches [9]–[13] are able to conduct effective management and protocol optimizations for deployed WSNs consisting of a large number of unattended sensor nodes. For example, PAD [10] depends on the routing path information to build a Bayesian network for inferring the root causes of abnormal phenomena. Path information is also important for a network manager to effectively manage a sensor network. For example, given the per-packet path information, a network manager can easily find out the nodes with a lot of packets forwarded by them, i.e., network hop spots. Then, the manager can take actions to deal with that problem, such as deploying more nodes to that area and modifying the routing layer protocols. Furthermore, per-packet path information is essential to monitor the fine-grained per-link metrics. For example, most existing delay and loss measurement approaches [9], [14] assume that the routing topology is given *a priori*. The time-varying routing topology can be effectively obtained by per-packet routing path, significantly improving the values of existing WSN delay and loss tomography approaches.

A straightforward approach is to attach the entire routing path in each packet. The problem of this approach is that its message overhead can be large for packets with long routing paths. Considering the limited communication resources of WSNs, this approach is usually not desirable in practice.

In this paper, we propose iPath, a novel path inference approach to reconstruct routing paths at the sink side. Based on a real-world complex urban sensing network with all node generating local packets, we find a key observation: It is highly probable that a packet from node i and one of the packets from i 's parent will follow the same path starting from i 's parent toward the sink. We refer to this observation as *high path similarity*. Fig. 1 shows a simple example where S is the sink node. a_1 denotes a packet from A, and b_1, b_2, b_3 denotes packets from B (A's parent). *High path similarity* states that it is highly probable that a_1 will follow the same path (i.e., $\text{path}(a_1) - A$, which means the subpath by removing node A from $\text{path}(a_1)$) as one of B's packet, say b_1 , i.e., $\text{path}(a_1) = (A, \text{path}(b_1))$.

The basic idea of iPath is to exploit high path similarity to iteratively infer long paths from short ones. iPath starts with a known set of paths (e.g., the one-hop paths are already known) and performs path inference iteratively. During each iteration,

Manuscript received February 19, 2014; revised August 18, 2014; accepted October 28, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Ramasubramanian. Date of publication December 05, 2014; date of current version February 12, 2016. This work was supported by the National Science Foundation of China under Grant No. 61472360, the Fundamental Research Funds for the Central Universities, National Key Technology R&D Program under Grant 2012BAI34B01, and the Demonstration of Digital Medical Service and Technology in Destined Region. (Corresponding author: Wei Dong.)

Y. Gao, W. Dong, C. Chen, J. Bu, and W. Wu are with the College of Computer Science, Zhejiang University, Hangzhou 310027, China (e-mail: dongw@zju.edu.cn).

X. Liu is with the School of Computer Science, McGill University, Montreal, QC H3A 2A7, Canada.

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TNET.2014.2371459

it tries to infer paths one hop longer until no paths can be inferred. In order to ensure correct inference, iPath needs to verify whether a short path can be used for inferring a long path. For this purpose, iPath includes a novel design of a lightweight hash function. Each data packet attaches a hash value that is updated hop by hop. This *recorded hash value* is compared against the *calculated hash value* of an inferred path. If these two values match, the path is correctly inferred with a very high probability. In order to further improve the inference capability as well as its execution efficiency, iPath includes a fast bootstrapping algorithm to reconstruct a known set of paths.

iPath achieves a much higher reconstruction ratio in networks with relatively low packet delivery ratio and high routing dynamics.

The contributions of this work are the following.

- We observe high path similarity in a real-world sensor network. Based on this observation, we propose an iterative boosting algorithm for efficient path inference. We propose a lightweight hash function for efficient verification within iPath. We further propose a fast bootstrapping algorithm to improve the inference capability as well as its execution efficiency.
- We propose an analytical model to calculate the successful reconstruction probability in various network conditions such as network scale, routing dynamics, packet losses, and node density.
- We implement iPath and evaluate its performance using traces from large-scale WSN deployments as well as extensive simulations. iPath achieves higher reconstruction ratio under different network settings compared to states of the art.

The rest of this paper is organized as follows. Section II discusses the related works. Section III gives the results of a measurement study on two deployed networks. Section IV describes the network model and assumptions made in this paper. Section V describes the design of iPath. Section VI formally analyzes the reconstruction performance of iPath and two related works. Section VII evaluates iPath's performance compared to existing works in a trace-driven study. Section VIII reveals more system insights by extensive simulations, and Section IX concludes this paper.

II. RELATED WORK

In wired IP networks, fine-grained network measurement includes many aspects such as routing path reconstruction, packet delay estimation, and packet loss tomography. In these works, probes are used for measurement purpose [15]–[18]. Traceroute is a typical network diagnostic tool for displaying the path multiple probes. DTrack [18] is a probe-based path tracking system that predicts and tracks Internet path changes. According to the prediction of path changes, DTrack is able to track path changes effectively. FineComb [15] is a recent probe-based network delay and loss topography approach that focuses on resolving packet reordering. In fact, a recent work [19] summarizes the design space of probing algorithms for network performance measurement. Using probes, however, is usually not desirable in WSNs. The main reason is that the wireless dynamic

is hard to be captured by a small number of probes, and frequent probing will introduce high energy consumption.

A recent work [20] investigates the problem of identifying per-hop metrics from end-to-end path measurements, under the assumption that link metrics are additive and constant. Without using any active probe, it constructs a linear system by the end-to-end measurements from a number of internal monitors. Path information is assumed to exist as prior knowledge to build the linear system. Therefore, this work is orthogonal to iPath, and combining them may lead to new measurement techniques in WSNs.

There are several recent path reconstruction approaches for WSNs [7], [8], [10], [21]. PAD is a diagnostic tool that includes a packet marking scheme to obtain the network topology. PAD [10] assumes a relatively static network and uses each packet to carry one hop of a path. When the network becomes dynamic, the frequently changing routing path cannot be accurately reconstructed. MNT [8] first obtains a set of reliable packets from the received packets at sink, then uses the reliable packet set to reconstruct each received packet's path. When the network is not very dynamic and the packet delivery ratio is high, MNT is able to achieve high reconstruction ratio with high reconstruction accuracy. However, as described in Section V-C, MNT is vulnerable to packet loss and wireless dynamics. PathZip [7] hashes the routing path into an 8-B hash value in each packet. Then, the sink performs an exhaustive search over the neighboring nodes for a match. The problem of PathZip is that the search space grows rapidly when the network scales up. Pathfinder [21] assumes that all nodes generate local packets and have a common interpacket interval (i.e., IPI). Pathfinder uses the temporal correlation between multiple packet paths and efficiently compresses the path information into each packet. Then, at the PC side, it can infer packet paths from the compressed information. Compared to PathZip, iPath exploits high path similarity between multiple packets for fast inference, resulting in much better scalability. Compared to MNT, iPath has much less stringent requirements on successful path inference: In each hop, iPath only requires *at least* one local packet following the same path, while MNT requires a set of consecutive packets with the same parent (called reliable packets). Compared to Pathfinder, iPath does not assume common IPI. iPath achieves higher reconstruction ratio/accuracy in various network conditions by exploiting path similarity among paths with different lengths.

III. MEASUREMENT STUDY

In order to quantify the path similarity in real-world deployment, we conduct a measurement study on two deployed networks—CitySee [3] and GreenOrbs [2]. The CitySee project is deployed in an urban area for measuring carbon emission. All nodes are organized in four subnets. Each subnet has one sink node, and sink nodes communicate to the base station through 802.11 wireless links. We collect traces from one sink of a subnet with 297 nodes. The GreenOrbs project includes 383 nodes in an forest area for measuring the carbon absorbance.

These two networks use the Collection Tree Protocol [4] as its routing protocol. In order to reduce the energy consumption and prolong the network lifetime, all nodes except the sink node

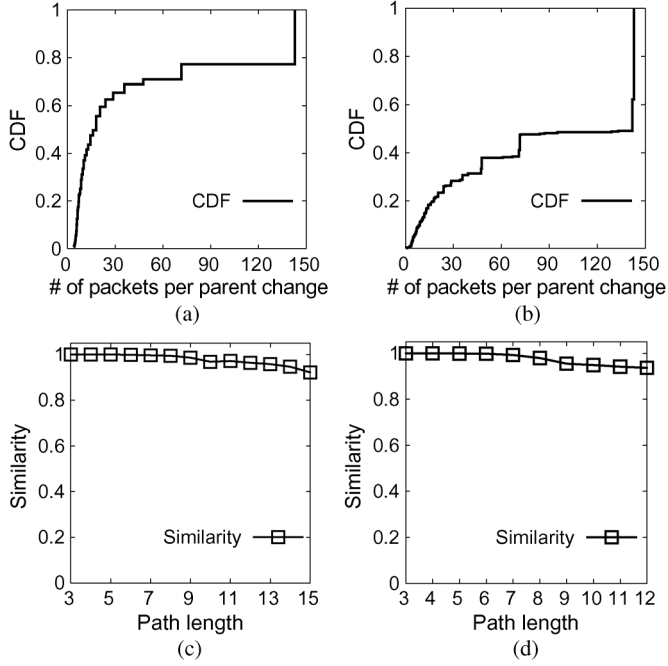


Fig. 2. Path similarity and routing dynamics in two large-scale deployed sensor networks. (a) Dynamic of CitySee. (b) Dynamic of GreenOrbs. (c) Similarity of CitySee. (d) Similarity of GreenOrbs.

work at low-power listening states. The wakeup interval of the low power setting is 512 ms. Each node reports data packets to a sink with a period of 10 min. Each data packet carries the routing path information directly for offline analysis.

We first look at the routing dynamics of the networks. We measure a quantity λ that is defined to be the average number of periods (i.e., local packets) between two parent changes by a node. It is simply the inverse of the number of parent changes per period at a node. A smaller λ means more frequent parent changes. Fig. 2(a) and (b) shows the cumulative distribution function (CDF) of λ for all nodes in the two networks. We can see that these two network have different degrees of routing dynamics. On average, there is a parent change every 46.9 periods in CitySee and 89.1 periods in GreenOrbs. As a comparison, the MNT paper [8] reports a parent change every $88.2 \sim 793.3$ periods of the networks tested, which have less frequent parent changes. We see that CitySee and GreenOrbs have high routing dynamics, making per-packet path inference necessary for reasoning about complex routing behaviors.

On the other hand, we observe high path similarity in the networks, i.e., it is highly probable that a packet from node i and one of the packets from i 's parent will follow the same path starting from i 's parent toward the sink. To quantitatively measure path similarity, we define $sim(len)$ such that among all packets with path length len , there are $sim(len)$ ratio of packets that follow the same path as at least one $(len - 1)$ hop packet. Fig. 2(c) and (d) shows the $sim(len)$ values with varying len . We see that the values of $sim(len)$ are close to 1, indicating that a high path similarity in both the CitySee network and GreenOrbs network. Note that the paths shown in these two figures include more than 99% of the total packet paths in these two traces. Therefore, the path similarity observation is not biased.

The above results show that although there are severe routing dynamics, the path similarity can still be very high. This key observation gives us important implications for efficient path inference: If a similar short path is known, it can be used to reconstruct a long path efficiently.

IV. NETWORK MODEL

In this section, we summarize the assumptions made and data fields in each packet.

We assume a multihop WSN with a number of sensor nodes. Each node generates and forwards data packets to a single sink. In multisink scenarios, there exist multiple routing topologies. The path reconstruction can be accomplished separately based on the packets collected at each sink.

In each packet k , there are several data fields related to iPath. We summarize them as follows.

- The first two hops of the routing path, origin $o(k)$ and parent $p(k)$. Including the parent information in each packet is common best practice in many real applications for different purposes like network topology generation or passive neighbor discovery [8], [22].
- The path length $len(k)$. It is included in the packet header in many protocols like CTP [4]. With the path length, iPath is able to filter out many irrelevant packets during the iterative boosting (Section V-A).
- A hash value $h(k)$ of packet k 's routing path. It can make the sink be able to verify whether a short path and a long path are similar. The hash value is calculated on the nodes along the routing path by the PSP-Hashing (Section V-B).
- The global packet generation time $t_g(k)$ and a parent change counter $pc(k)$. These two fields are *not required* in iPath. However, with this information, iPath can use a fast bootstrapping algorithm (Section V-C) to speed up the reconstruction process as well as reconstruct more paths.

V. IPATH DESIGN

The design of iPath includes three parts: iterative boosting, PSP-Hashing, and fast bootstrapping. The iterative boosting algorithm is the main part of iPath. It uses the short paths to reconstruct long paths iteratively based on the path similarity. PSP-Hashing provides a path similarity preserving hash function that makes the iterative boosting algorithm be able to verify whether two paths are similar with high accuracy. When the global generation time and the parent change counter are included in each packet, a fast bootstrapping method is further used to speed up the iterative boosting algorithm as well as to reconstruct more paths.

A. Iterative Boosting

iPath reconstructs unknown long paths from known short paths iteratively. By comparing the *recorded hash value* and the *calculated hash value*, the sink can verify whether a long path and a short path share the same path after the short path's original node. When the sink finds a match, the long path can be reconstructed by combining its original node and the short path.

Algorithm 1: The iterative boosting algorithm

Input: An initial set of packets P_{init} whose paths have been reconstructed and a set of other packets P_x

Output: The routing paths of packets

```

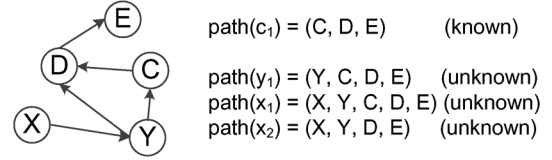
1: Procedure ITERATIVE-BOOSTING
2:    $P_n \leftarrow P_{init}$ 
3:   while  $P_n \neq \{\}$  do
4:      $P_{nn} \leftarrow \{\}$ 
5:     for each packet  $k$  in  $P_n$  do
6:       for each packet  $i$  in  $P_x$  do
7:          $res = \text{RECOVER}(k, i)$ 
8:         if  $res \equiv \text{True}$  then
9:            $P_{nn} \leftarrow P_{nn} \cup i$ 
10:           $P_x \leftarrow P_x - i$ 
11:     $P_n \leftarrow P_{nn}$ 
12: procedure RECOVER( $k, i$ )
13:   if  $len(i) - len(k) \notin \{1, 2\}$  then
14:     return False
15:   if  $len(i) - len(k) \equiv 2$ 
16:     if  $hash(o(i), p(i), path(k)) \equiv h(i)$  then
17:        $path(i) \leftarrow (o(i), p(i), path(k))$  // Case 2
18:       return True
19:     return False
20:   if  $len(i) - len(k) \equiv 1$  then
21:     if  $hash(o(i), path(k)) \equiv h(i)$ 
22:        $path(i) \leftarrow (o(i), path(k))$  // Case 1
23:       return True
24:     if  $hash(o(i), p(i), path(k) - o(k)) \equiv h(i)$  then
25:        $path(i) \leftarrow (o(i), p(i), path(k) - o(k))$  // Case 3
26:       return True
27:     return False

```

Algorithm 1 gives the complete iterative boosting algorithm. There are two procedures, the *Iterative-Boosting* procedure (line 1) and the *Recover* procedure (line 12). The *Iterative-Boosting* procedure includes the main logic of the algorithm that tries to reconstruct as many as possible packets iteratively. The input is an initial set of packets P_{init} whose paths have been reconstructed and a set of other packets P_x . During each iteration, P_n is a set of newly reconstructed packet paths. The algorithm tries to use each packet in P_n to reconstruct each packet's path in P_x (lines 5 ~ 10). The procedure ends when no new paths can be reconstructed (line 3).

The *Recover* procedure tries to reconstruct a long path with the help of a short path. Based on the high path similarity observation, the following cases describe how to reconstruct a long path.

Case 1 (Lines 21 ~ 23): The sink uses the hash value in packet k with length len and packet i with length $(len + 1)$ to verify whether the path of k is similar with i 's path. The verification is simply to check whether $hash(o(i), path(k))$ equals $h(i)$ (line 21). If the verification passes, packet i 's path is reconstructed as $(o(i), path(k))$. Fig. 3 shows an example: A packet with path (C, D, E) can reconstruct a packet's path that is (Y, C, D, E).



Case 1: $hash(Y, path(c_1)) \equiv h(y_1) \rightarrow path(y_1) = (Y, C, D, E)$

Case 2: $hash(X, Y, path(c_1)) \equiv h(x_1) \rightarrow path(x_1) = (X, Y, C, D, E)$

Case 3: $hash(X, Y, path(c_1) - C) \equiv h(y_2) \rightarrow path(y_2) = (X, Y, D, E)$

Fig. 3. Example to illustrate three cases of reconstructing long paths based on short paths in the iterative boosting algorithm. X, Y, etc., are nodes, and x_1, y_1 , etc., are packets originated from different nodes.

In practice, the first hop receiver (i.e., parent) node is also included in each packet. With this parent information in each packet, for a reconstructed path of packet k with length len in P_{new} , it can help reconstruct more paths with length $(len + 1)$ and $(len + 2)$. Specifically, there are two additional cases to reconstruct long paths.

Case 2 (Lines 15 ~ 19): The second case is similar with the first one. Since packet k carries its first two hops $o(k)$ and $p(k)$, the sink can reconstruct path with length $(len + 2)$. The sink checks whether $hash(o(i), p(i), path(k))$ equals $h(i)$ (line 16). If the verification passes, packet i 's path is $(o(i), p(i), path(k))$. For example, a packet with path (C, D, E) can reconstruct a packet's path that is (X, Y, C, D, E).

Case 3 (Lines 24 ~ 26): The third case is to verify whether $hash(o(i), p(i), path(k) - o(k))$ equals $h(i)$ (line 24). We use $(path(k) - o(k))$ to denote the path of packet k without its origin node $o(k)$. If the verification passes, packet i 's path is $(o(i), p(i), path(k) - o(k))$. For example, a packet with path (C, D, E) can reconstruct a packet's path that is (X, Y, D, E).

Since all the three cases require the difference of the two packets' path lengths to be one or two, the procedure can return false immediately if this constraint does not hold (lines 13 and 14). Then, the three cases try to reconstruct the long path (lines 15 ~ 27). The *Recover* procedure outputs the reconstructed path if one of the three cases successfully finds a match (lines 16, 21, and 24).

When the input trace is relatively large, iPath divides the trace into multiple time-windows. When a trace with N packets is divided into w windows evenly, the worst-case time complexity of the algorithm is $O(N^2/w)$. Details about the time complexity analysis can be found in the technical report [23] of this work. Note that this time complexity represents the number of procedure *Recover* executed in Algorithm 1. The overall time consumption also depends on the time complexity of the hash function.

In order to make the iterative boosting efficient and effective, two problems need to be addressed. The first is how to design a lightweight hash function that can be calculated efficiently on each sensor node. iPath uses PSP-Hashing, a novel lightweight path similarity preserving hash function, to make the sink be able to verify two similar paths efficiently (Section V-B). Second, each iteration of the iterative boosting needs a set of reconstructed paths. Therefore, how to obtain the initial set of paths is important. The basic initial set is all paths with length

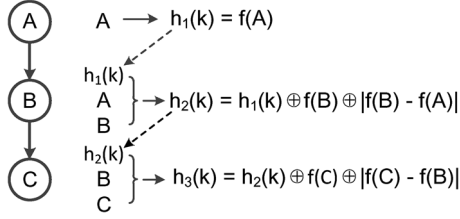


Fig. 4. PSP-Hashing function. Each node in the routing path takes three inputs and updates the hash value $h_i(k)$ in packet k . Note that we use $h_i(k)$ to denote the hash value in packet k at the i th hop, instead of a function defined on packet k itself.

one and two since these paths can be directly reconstructed by its origin and parent. iPath further uses a fast bootstrapping algorithm to obtain a larger initial set (Section V-C).

B. PSP-Hashing

As mentioned in the iterative boosting algorithm, the PSP-Hashing (i.e., path similarity preserving) plays a key role to make the sink be able to verify whether a short path is similar with another long path. There are three requirements of the hash function.

- The hash function should be lightweight and efficient enough since it needs to be run on resource-constrained sensor nodes.
- The hash function should be order-sensitive. That is, $\text{hash}(A, B)$ and $\text{hash}(B, A)$ should not be the same.
- The collision probability should be sufficiently low to increase the reconstruction accuracy.

Traditional hash functions like SHA-1 are order-sensitive. However, they are not desirable due to their high computational and memory overhead. For example, an implementation [24] of SHA-1 on a typical sensor node TelosB takes more than 4 kB program flash and longer than 5 ms to hash 20 B of data. Note that this memory overhead is about 10% of the total program flash of a TelosB node, and 5 ms computational overhead nearly doubles the forwarding delay in a typical routing protocol [4]. In order to design an efficient and lightweight hash function, efficient operations, such as bitwise XOR operation, are preferred. Since XOR operation is not order-sensitive, the order information should be explicitly hashed into the hash value.

We propose PSP-Hashing, a lightweight path similarity preserving hash function to hash the routing path of each packet. PSP-Hashing takes a sequence of node ids as input and outputs a hash value. Each node along the routing path calculates a hash value by three pieces of data. One is the hash value in the packet that is the hash result of the subpath before the current node. The other two are the current node id and the previous node id. The previous node id in the routing path can be easily obtained from the packet header. Fig. 4 shows this chained hash function along the routing path.

The following equation describes the calculation of each step of PSP-Hashing:

$$h_i(k) = \begin{cases} f(n_1), & \text{if } i = 1 \\ h_{i-1}(k) \oplus f(n_i) \oplus |f(n_i) - f(n_{i-1})|, & \text{if } i > 1 \end{cases} \quad (1)$$

where n_i is the node id of the i th node of packet k 's routing path and $f(n_i)$ is a mapping function that maps the node id to a

number with the same length as the hash value. The following equation gives a simple mapping function with input node id A :

$$f(A) = (A \cdot \alpha \lll \lceil \log_2 A \rceil) \bmod 2^m + A \quad (2)$$

where α is a prime number larger than 2^m and m is the number of bits of the hash value that is configured as 32. In practice, 4 B hash value is sufficient to achieve high reconstruction accuracy. In Section VIII, we will further show the impact of different lengths of the hash value.

For example, the hash value of a path (A, B, C) can be calculated as follows. 1) At the origin node A, the hash value is $f(A)$. 2) At node B, the hash value becomes $f(A) \oplus f(B) \oplus |f(B) - f(A)|$. 3) At node C, the final hash value is $f(A) \oplus f(B) \oplus |f(B) - f(A)| \oplus f(C) \oplus |f(C) - f(B)|$.

PSP-Hashing has several good features.

- 1) On each node along the routing path, the calculation is simple and can be executed very efficiently. As is given in (1), each node only needs to do several bit operations and arithmetic operations. On TelosB sensor node, the computational overhead of PSP-Hashing is negligible ($< 1 \mu\text{s}$). The memory overhead of the PSP-Hashing is less than 10 B.
- 2) Another good feature is that PSP-Hashing is an order-sensitive hash function. That is, $\text{hash}((A, B, C, D))$ is different with $\text{hash}((C, B, A, D))$. In practice, paths are easy to have different orders with the same nodes due to parent changes. Therefore, this feature is important to achieve low error ratio. For example, assume $\text{hash}((A, B, C, D))$ equals to $\text{hash}((C, B, A, D))$. When the sink has reconstructed the path (B, A, D) and tries to reconstruct another path path_x that is actually (A, B, C, D) , it will find that $\text{hash}(o(\text{path}_x), (B, A, D))$ equals $\text{hash}(\text{path}_x)$ since $\text{hash}((A, B, C, D))$ equals $\text{hash}((C, B, A, D))$. Then, the sink reconstructs path_x to be (A, B, A, D) , which is not correct. Since PSP-Hashing is order-sensitive, it can exclude the above error case and improve the accuracy of iPath.

C. Fast Bootstrapping

The iterative boosting algorithm needs an initial set of reconstructed paths. In addition to the one/two-hop paths, the fast bootstrapping algorithm further provides more initial reconstructed paths for the iterative boosting algorithm. These initial reconstructed paths reduce the number of iterations needed and speed up the iterative boosting algorithm.

The fast bootstrapping algorithm needs two additional data fields in each packet k , parent change counter $pc(k)$ and global packet generation time $t_g(k)$. The parent change counter records the accumulated number of parent changes, and the global packet generation time can be estimated by attaching an accumulated delay in each packet [12]. For packet k , there are an upper bound Δ_k^u and a lower bound Δ_k^l of the difference between the estimated packet generation time $\hat{t}_g(k)$ and the real value $t_g(k)$.

The basic idea is to reconstruct a packet's path by the help of the local packets at each hop. For each node, we can obtain its *stable periods* by the parent change counter attached in each of

Algorithm 2: The fast bootstrapping algorithm

Input: All received packets and a packet k whose path is being reconstructed

Output: $path(k)$: the routing path of packet k

```

1: procedure FAST-BOOTSTRAPPING
2:    $path(k) \leftarrow (o(k), p(k))$ 
3:    $n \leftarrow p(k)$ 
4:   while  $n \neq sink$  do
5:      $u \leftarrow \arg \max_x \hat{t}_g(x) + \Delta_x^u$  for all  $x : o(x) \equiv n$ 
6:      $\cap \hat{t}_g(x) + \Delta_x^u < \hat{t}_g(k) - \Delta_k^l$ 
7:      $v \leftarrow \arg \min_x \hat{t}_g(x) + \Delta_x^v$  for all  $x : o(x) \equiv n$ 
8:      $\cap \hat{t}_g(x) - \Delta_x^l < t_s(k)$ 
9:     if  $u \equiv \{\}$  or  $v \equiv \{\}$  or  $pc(u) \neq pc(v)$  then
10:      break
11:      $path(k) \leftarrow path(k) \cup p(n)$ 
12:      $n \leftarrow p(n)$ 
13:   return  $path(k)$ 

```

its local packet. A stable period of a node is a period of time in which the node does not change its parent. If a packet is forwarded by this node in one of its *stable periods*, we can safely reconstruct the next-hop of that forwarded packet to be the parent of its local packet in the same stable period.

In order to determine whether a packet is in its forwarders' stable periods, we use the packet generation time and the parent change counter in each packet. Specifically, a packet k 's forwarding node F 's stable period can be identified at the sink by two of its local packets u and v where $pc(u) \equiv pc(v)$ ($\hat{t}_g(u) < \hat{t}_g(v)$). We call these two packets *indicating packets* that are used to indicate one or multiple consecutive stable periods. If a packet k 's generation time $\hat{t}_g(k)$ is later than u 's generation time $\hat{t}_g(u)$ and it arrives at the sink before packet v 's sink time $t_s(v)$, k must have arrived at node F in its stable period.

Algorithm 2 shows the fast bootstrapping algorithm that reconstructs an initial set of packets for the iterative boosting algorithm. For each input packet k , it first initializes its path to be $(o(k), p(k))$ (line 2). Then, the algorithm locates the two indicating packets u and v at each hop of k 's routing path (lines 5 ~ 8). If the two indicating packets both exist and have the same parent change counter, packet k must have arrived at the forwarder in a stable period, and its next-hop can be safely reconstructed (lines 9 ~ 12). Given N packets and path length len on average, the time complexity of the fast bootstrapping algorithm is $O(N \cdot len)$ since the reconstruction process is done hop by hop.

Compared to MNT, the fast bootstrapping algorithm is more loss resilient. Fig. 5 shows an example. When two packets are lost, the stable periods of the fast bootstrapping algorithm are not affected. The reason is that the parent change counters in the first and last packets can still indicate the stable periods. The stable periods of MNT, however, are affected in case of packet losses. The reason is that MNT requires the stable periods to be indicated by *consecutive* received packets. In this example, the four periods around the two lost packets are not stable periods in MNT. In the fast bootstrapping algorithm, the forwarded

packet is still in stable periods, and its next-hop can be safely reconstructed.

VI. ANALYSIS

In order to quantify the reconstruction performance of iPath and two related approaches, we analyze these approaches by a novel analytical model. Here, the performance means the probability of a successful reconstruction, which is the most important metric. We use the following definitions for analysis.

- Local packet generation period t . iPath does not require all nodes have the same local packet generation period. In order to simplify the presentation, we assume all nodes have the same packet generation period in this analysis section.
- Routing dynamics δ , which is the number of parent changes in a single period t . On average, there is one parent change every $\lambda = 1/\delta$ local packets. We call these λ consecutive periods as one *cycle* for analysis.
- Packet delivery ratio PDR of packet k . It can be calculated as the product of the packet reception ratios (PRR) along the routing path of packet k , $\prod_{z \in path(k)} PPR(z)$.
- The average node degree d .
- As mentioned in the fast bootstrapping algorithm, a *stable period* of a node is a period t in which the node does not change parent.

Then, we use these parameters to model the probability of a successful reconstruction by MNT and iPath. The analysis of PathZip can be found in the technical report [23] of this work.

A. Performance of MNT

MNT [8] reconstructs the whole path hop by hop. Since there is no parent change within a number of consecutive stable periods, we can calculate the probability of a successful reconstruction by the product of the ratios of stable periods on all forwarding nodes. The following equation describes the probability of a successful reconstruction of MNT:

$$R_{MNT}(k) = \prod_{j \in path(k) - \{o(k), sink\}} S_{MNT}(j) \quad (3)$$

where $S_{MNT}(j)$ is the ratio of stable periods of node j . Then, we calculate $S_{MNT}(j)$ by dividing the expected number of stable periods in one cycle by the total periods λ in one cycle. Fig. 6 gives an example. A cycle includes λ periods in which there are $(\lambda - 1)$ stable periods. When there is no packet loss, the ratio can be easily calculated as $S_{no-loss}(j) = \frac{\lambda-1}{\lambda}$.

When there are packet losses, some stable periods will be broken, and the number of stable periods will be less. Specifically, one packet loss will break one or two stable periods. In Fig. 6's example, when the first packet is lost, the number of stable periods in the first cycle will be two. When the second packet is lost, however, the number of stable periods will be only one. The reason is that MNT requires consecutive local packets to indicate stable periods. The following equation describes the calculation of the ratio of stable periods $S_{MNT}(j)$:

$$S_{MNT}(j) = \frac{1}{\lambda} \sum_{l=\max(0, \lambda-2x_j-1)}^{\lambda-x_j-1} l \cdot \frac{\binom{\lambda-x_j-1}{\lambda-x_j-1-l} \binom{x_j+1}{\lambda-l-x_j}}{\binom{\lambda}{x_j}}. \quad (4)$$

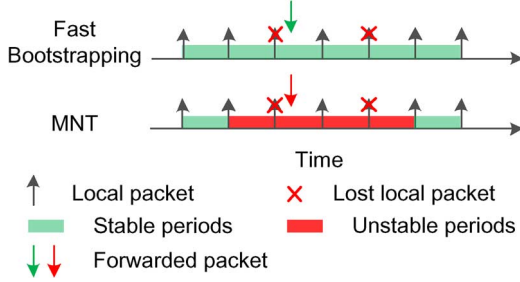


Fig. 5. Example to illustrate the difference of the fast bootstrapping algorithm and MNT. When the same two packets are lost, the forwarded packet is still in a stable period in the fast bootstrapping algorithm, while it is not in any stable period in MNT.

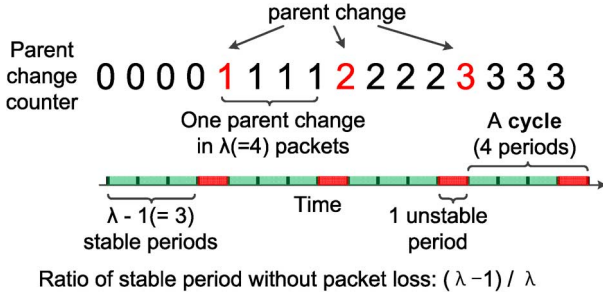


Fig. 6. Example to illustrate how to calculate the ratio of stable periods without any packet losses. In λ periods, there is one parent change that introduces one unstable period.

where x_j is the number of lost packets in the stable periods of one cycle. The number of packet losses x_j can be calculated as $\lambda \cdot (1 - \text{PDR}_j)$. Details about the calculation of $S_{\text{MNT}}(j)$ can be found in the technical report of this work [23].

B. Performance of iPath

The fast bootstrapping algorithm reconstructs an initial set of paths for the iterative boosting algorithm. Therefore, we first analyze the performance of the fast bootstrapping algorithm.

1) *Performance of Fast Bootstrapping*: The fast bootstrapping algorithm reconstructs the routing path of a packet hop by hop. When the sink reconstructs the path of a packet to a forwarder F , it can reconstruct the next-hop only when the packet is in one of F 's stable periods. Therefore, the probability of a successful reconstruction of the fast bootstrapping algorithm is the product of the ratios of stable periods on all forwarding nodes

$$R_{fb}(k) = \prod_{j \in \text{path}(k) - \{o(k), \text{sink}\}} S(j) \quad (5)$$

where $S(j)$ is the ratio of stable periods of node j . Then, we need to model the ratio of stable periods $S(j)$.

When a packet loss happens at the start or end of consecutive stable periods, it will cause the stable periods to be less. However, when a packet loss happens inside consecutive stable periods, the length of the stable periods stays the same. In the example shown in Fig. 6, if the first packet with parent change counter zero is lost, the number of stable period in the first four periods will become two. However, if the second packet with parent change counter zero is lost, the number of stable periods left is still three. The reason is that in the fast bootstrapping algorithm, a forwarded packet is able to identify itself in a

stable period if the two local packets before and after it have the same parent change counter. It does not matter whether these two packets have consecutive sequence numbers. Compared to MNT, in which a packet loss always break one or two stable periods, the fast bootstrapping algorithm has more stable periods left. Therefore, the fast bootstrapping algorithm is able to reconstruct more paths than MNT.

In one cycle's stable periods of node j , the number of packet losses x_j is $\lambda \cdot (1 - \text{PDR}_j)$. We want to calculate the expected number of stable periods left with x_j packet losses in the stable periods. Assume there are l stable periods left after x_j packet losses in the λ packets of one stable period. We calculate the number of different cases as follows.

Since l consecutive stable periods are left, the first and last packets of the left stable periods should not be lost, and packets outside the left stable periods are all lost. There are $(\lambda - l)$ different cases. Then, inside the left stable periods, there are $(l - 1)$ positions for $(\lambda - x_j - 2)$ packets not lost. Therefore, there are $(\lambda - l) \binom{l-1}{\lambda-x_j-2}$ different cases in total. Then, the probability that l stable periods are left with x_j packet losses is $(\lambda - l) \binom{l-1}{\lambda-x_j-2} / \binom{\lambda}{x_j}$. Therefore, the expected number of stable periods left is $\sum_{l=\lambda-x_j}^{\lambda} l (\lambda - l) \binom{l-1}{\lambda-x_j-2} / \binom{\lambda}{x_j}$.

Since one packet loss will break zero or one stable period, the range of the number of the left stable periods is $[\lambda - x_j - 1, \lambda - 1]$. Therefore, the following equation gives the ratio of stable periods with packet losses:

$$S(j) = \frac{1}{\lambda} \sum_{l=\lambda-x_j-1}^{\lambda-1} l \cdot \frac{(\lambda - l) \binom{l-1}{\lambda-x_j-2}}{\binom{\lambda}{x_j}}. \quad (6)$$

2) *Performance of Iterative Boosting*: The iterative boosting algorithm reconstructs long paths based on short paths. Specifically, a path p_1 with length len can be reconstructed by another path p_2 with length $(len - 1)$ or $(len - 2)$ (three cases in Section V-A). That path p_2 can be reconstructed by another even shorter path p_3 . A path can be successfully reconstructed only when the path helping to reconstruct it can also be successfully reconstructed. In other words, a path can be successfully reconstructed only when at each of its hops, there exists at least one shorter path that can help reconstruct that hop. For simplicity, we only consider the second case (Section V-A) of the iterative boosting in the performance analysis. For example, a path (A, B, C, D, E, F) can be reconstructed by path (C, D, E, F), which can be reconstructed by path (E, F). From another point of view, path (E, F) and path (C, D, E, F) both contribute two hops in reconstructing the path (A, B, C, D, E, F).

Based on the above analysis, we can calculate the probability of a successful reconstruction by multiplying the probabilities there exists at least one shorter helper path at several hops. We call these hops *anchor hop sequence AH*, which can be obtained as follows. First, all hops in the path except the origin, parent, and sink are in sequence AH . Second, one hop is removed in every two hops since *Case 2* of the iterative boosting algorithm can reconstruct two hops at once. Third, if the path of one packet originated from one hop in the left hops in sequence AH has been reconstructed by the fast boosting algorithm, remove all hops after that hop from the anchor hop sequence AH . In the previous example, the anchor hop sequence AH is (C, E) after

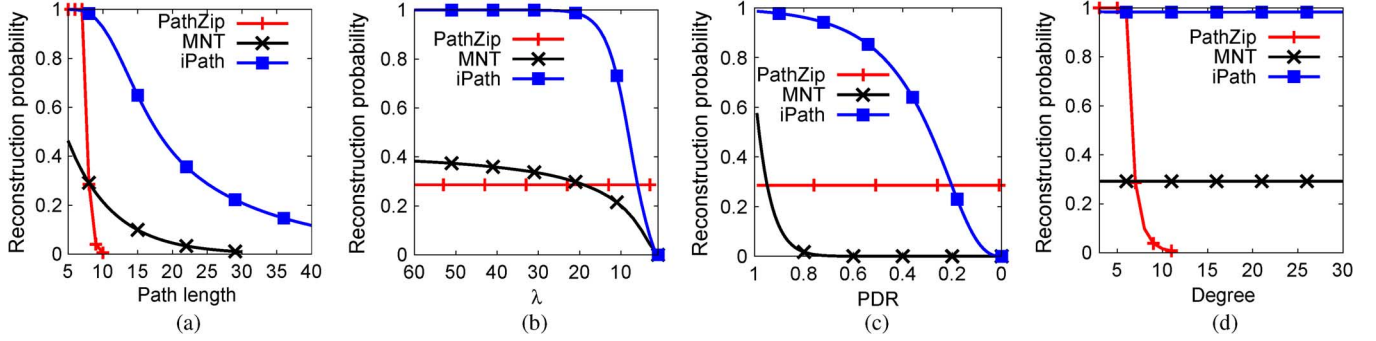


Fig. 7. Analytical reconstruction probabilities of the three approaches when one of the four parameters varies. (a) Scale. (b) Dynamic. (c) PDR. (d) Density.

the first two steps. This means if there exists at least one helper packet originated from C and E, the sink can reconstruct path (A, B, C, D, E, F). If path (C, D, E, F) has been reconstructed by the fast bootstrapping algorithm, path (E, F) is not required in reconstructing path (A, B, C, D, E, F). Therefore, the final anchor hop sequence AH is (C).

Let the probability of the helper path existing at hop j be $E(j)$, then the probability of a successful reconstruction of the iterative boosting algorithm can be calculated as the following equation:

$$R_{ib}(k) = \prod_{j \in AH(k)} E(j). \quad (7)$$

Since $E(j)$ is the probability that there exists at least one helper path, it depends on the number of paths in a time window. The larger the time window is, the larger $E(j)$ is. Let N_j be the number of node j 's local packets in a time window. The following equation gives $E(j)$ at each hop:

$$E(j) = 1 - \prod_{u=1}^{N_j} (1 - \text{PDR}_j \cdot e(j, k_u)) \quad (8)$$

where $e(j, k_u)$ is the probability that packet k_u originated from node j can be the forwarded packet k 's helper packet. Packet k_u can be packet k 's helper packet only when their paths are similar, which means their paths should be the same after node j .

We first calculate the probability that a forwarded packet and packet k_u have the same next-hop. Let the two packets leave the forwarder with $\Delta t(k_u)$ time difference. When $\Delta t(k_u) \leq (\lambda - 1)t$, the same next-hop probability is $\frac{(\lambda - 1)t - \Delta t(k_u)}{\lambda t}$, where t is the local packet generation period and $(\lambda - 1)t$ is the number of consecutive stable periods in a cycle. When $\Delta t(k_u) > (\lambda - 1)t$, the two packets cannot stay in the same stable period. Then, the same next-hop probability is $\frac{1}{d}$, where d is the node degree.

Then, we extend the probability analysis from the same next-hop to the same path. For simplicity, let the time differences of packet k_u and the forwarded packet on the following hops be still $\Delta t(k_u)$. Therefore, the following equation gives the calculation of $e(j, k_u)$:

$$e(j, k_u) = \begin{cases} \left(\frac{1}{d}\right)^{|\text{path}(k_u)|-1}, & \text{if } \Delta t(k_u) > (\lambda - 1)t \\ \left(\frac{(\lambda - 1)t - \Delta t(k_u)}{\lambda t}\right)^{|\text{path}(k_u)|-1}, & \text{otherwise.} \end{cases} \quad (9)$$

The proposed analytical model is validated by comparing the reconstruction results to the modeled results in the technical report [23] of this work.

C. Analysis Results

We then present the analysis results according to the above analyses. There are four factors that can affect the performance of these approaches: network scale, routing dynamic, packet loss, and network density. Specifically, the network scale affects the path length, the routing dynamic affects the number of local packets λ in which there is a parent change, the packet loss affects the PDR, and the network density affects the degree d . Fig. 7 plots the reconstruction probabilities of the four approaches when one of the four parameters varies.

In Fig. 7(a), when the path length becomes long, the reconstruction ratio of each approach degrades. PathZip is able to achieve good performance in small-scale networks, but fails to reconstruct most of the paths when the path length becomes large. The reason is that the search space of PathZip grows exponentially. MNT's performance drops immediately when the path length becomes long. iPath achieves high reconstruction ratio in small-scale networks and is able to reconstruct much more paths than the other three approaches when the network becomes large.

λ is the number of local packets in which there is one parent change on average. When λ becomes smaller, it means the network becomes more dynamic and there are more parent changes. As shown in Fig. 7(b) and (c), PathZip's performance does not depend on λ or PDR. The performance of the other two approaches degrades when the network becomes dynamic or lossy. However, iPath can still achieve higher reconstruction ratio than MNT under different settings.

As shown in Fig. 7(d), the average degree does not affect MNT, Pathfinder, or iPath. PathZip's performance degrades rapidly when the degree increases. The reason is similar as the path length's case, which is the search space, grows rapidly when the degree increases.

The above analytical results show the broad applicability of iPath. For example, the parameters of the networks tested in MNT (TWIST [25] testbed, FlockLab [26] testbed) are shown as follows. The path length is three or four, the dynamic (λ) is from 88.2 to 793.3, and the PDR is from 98.3% to 99.9%. In these networks, we believe iPath is able to achieve very high reconstruction ratio according to the results of Fig. 7. In Section VII,

TABLE I
PERFORMANCE COMPARISON IN THE TRACE-DRIVEN STUDY

Approach / Metric		CitySee	GreenOrbs
PathZip	Reconstruction Ratio	21.03%	60.00%
	Error Ratio	0%	0.73%
MNT	Reconstruction Ratio	24.56%	71.06%
	Error Ratio	0.16%	0.19%
Pathfinder	Reconstruction Ratio	91.2%	93.4%
	Error Ratio	1.2%	1.1%
iPath	Reconstruction Ratio	98.26%	98.71%
	Error Ratio	0%	0%

we will give a trace-driven study from traces of two large-scale deployed networks. The results of the trace-driven study can further validate the broad applicability of iPath.

VII. TRACE-DRIVEN STUDY

In this section, we evaluate the performance of iPath by a trace-driven study from two large-scale deployed networks [2], [3]. As a comparison, we also implement three related approaches (i.e., MNT, PathZip, Pathfinder) and compare their performance to iPath.

A. Methodology

iPath is implemented in TinyOS2.1. In the trace-driven study, we use traces collected from the CitySee [3] project and the GreenOrbs project [2]. As mentioned in Section III, CitySee is a large-scale deployed network in an urban area for monitoring the carbon emission. GreenOrbs is a large-scale sensor network for forest monitoring. A customized Collection Tree Protocol [4] is used as the routing protocol in these two projects.

The CitySee and GreenOrbs traces include the first 10 hops in each packet for further offline analysis. Therefore, in the trace-driven study, we can use the collected routing information to reproduce the local operations on each node for each approach. Take PathZip as an example, we calculate the hash value according to the path included in each received packet at the sink side. Then, we run PathZip's algorithm to reconstruct paths and compare them to the collected ones to calculate the error ratio.

B. Results

We then show the path reconstruction performance of the four approaches in two traces.

Table I gives the reconstruction ratios and error ratios of the four approaches in two traces. PathZip performs better in the GreenOrbs trace; the reason is that the CitySee network has longer paths, which enlarges the search space. MNT also performs better in the GreenOrbs trace since MNT is vulnerable to packet losses and the CitySee trace has more packet losses. Pathfinder performs better than PathZip and MNT in both of the traces. iPath achieves the highest reconstruction ratio.

MNT and PathZip both have a small error ratio. The reason of PathZip's error reconstruction is clear since there are collisions during the exhaustive search. In MNT, it is assumed that a forwarded packet must be forwarded to the same next-hop if

TABLE II
OVERHEAD COMPARISON

Overhead/Approach	iPath	MNT	Pathfinder	PathZip
Message (bytes)	four	four	seven	eight
Computational (node)	low	low	low	low
Computational (PC)	low	modest	modest	high

its two anchor packets are sent to the same next-hop. We looked into the traces and found that there are a small number of violations of this assumption. In iPath, the fast bootstrapping uses a parent change counter to reconstruct a hop, which is safer than MNT's assumption. Pathfinder has a larger error ratio in both of the two traces, compared to other approaches.

We also evaluate the path reconstruction performance of the fast bootstrapping algorithm described in Section V-C. In the CitySee trace, the fast bootstrapping algorithm successfully reconstructs 23.94% of all paths. In the GreenOrbs trace, it successfully reconstructs 85.12% of all paths. These results show that the fast bootstrapping algorithm can reconstruct a large number of paths that are used as the initial set of paths for the iterative boosting method.

C. Overhead

Since sensor nodes are highly resource-constrained [27], we then present the message overhead and computational overhead of iPath and three related methods. Table II shows the results.

For message overhead, iPath introduces a hash value in each packet. In the current implementation, we use 4 B to store the hash value. In Section VIII-B, we will study the impact of hash value length and show that 4 B length is sufficient to achieve high reconstruction accuracy. MNT needs the path delay of each packet that can be recorded by a 4-B timestamp [12]. The message overhead of Pathfinder is 7 B, including one XOR byte, a 2-B bit-vector, and a 4-B path container. PathZip has an overhead of 8 B MD5-like hash value.

In iPath, the computational overhead at the node side is negligible since there are only several arithmetic operations. MNT, Pathfinder, and Pathzip do not require high computational overhead at the node side either. At the PC side, the time complexity of iPath is polynomial (detailed analysis can be found in the technical report of this work [23]). MNT needs to calculate a conflict free set for each node, which is NP-hard. Since there exists an approximate algorithm to improve the efficiency, it has modest computational overhead. Pathfinder includes a path speculation method that enumerates possible paths originated from a source node to find an XOR match. Since the number of possible paths from a source node could be large in a large-scale network, it has modest computational overhead at the PC side. PathZip does exhaustive search on neighboring nodes. Its computational overhead at the PC side could be high in large-scale networks.

VIII. SYSTEM INSIGHT

In this section, we conduct extensive simulations in TOSSIM [28], a standard simulator for TinyOS programs, to reveal more system insights. Specifically, we evaluate the reconstruction performance of iPath and three related works

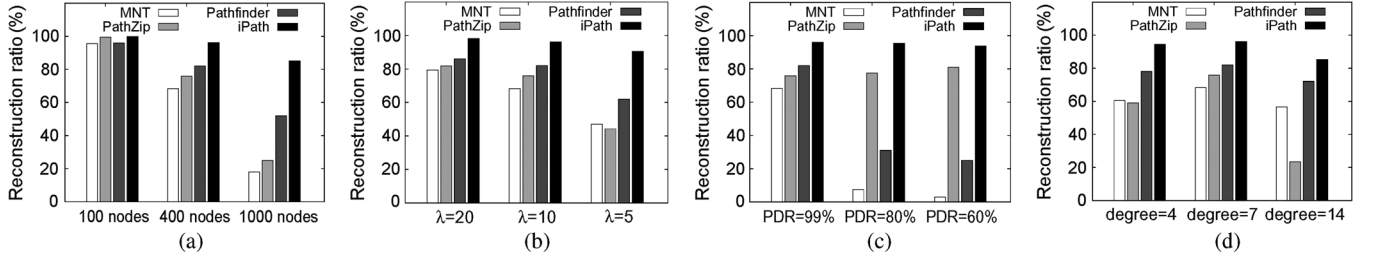


Fig. 8. Impact of (a) network scale, (b) routing dynamic, (c) packet delivery ratio, and (d) node degree.

in networks with different configuration settings such as path length, routing dynamic, packet delivery ratio, and degree. A number of networks with up to 1000 nodes are used in the simulations. We also evaluate the impact of length of the hash value, which is the key parameter in the design of iPath. At the end of this section, we will show a visualization of the reconstruction process in a network with 400 nodes.

A. Performance in Different Network Settings

Different numbers of nodes are uniformly distributed in simulations. We study the impacts of different network configuration parameters to the performance of different approaches. When we change one parameter, we keep the other parameters at their default values. The average default values of the path length, routing dynamic λ , PDR, and degree are 8, 10, 99%, and 7, respectively. In total, nine different network configurations are used in the simulations.

Fig. 8(a) shows the reconstruction ratios of three approaches in networks with different scales. All approaches achieve lower reconstruction ratio when the network has longer path length. However, iPath achieves much higher ratio compared to other approaches, especially in the large-scale network. Fig. 8(b) shows the reconstruction ratios of three approaches in networks with different routing dynamics (we use the number of local packets per parent change λ to measure the routing dynamics). The results are similar with the cases of the impact of path length. iPath achieves high reconstruction in a network with severe routing dynamics, while the reconstruction ratios of other three approaches drop significantly. Fig. 8(c) shows the reconstruction ratios of three approaches. The reconstruction ratio of MNT drops significantly when the PDR decreases. This result fits the analysis result [Fig. 7(c)] and shows that MNT is vulnerable to packet losses. The reconstruction ratio of PathZip does not depend on the PDR, so its performance remains stable in three networks with different PDR. iPath achieves the highest reconstruction ratio in all PDR settings. This result shows that iPath is more loss resilient compared to MNT. Fig. 8(d) shows the reconstruction ratios of three approaches. MNT's reconstruction ratio does not depend on the degree. PathZip cannot achieve high reconstruction ratio in sparse network. The reason is that we keep the network size at 400 nodes and tune the degree. Then, the path length is longer when the degree is smaller, making PathZip unable to achieve high reconstruction ratio in a sparse network. iPath can achieve high reconstruction ratio in both sparse and dense networks.

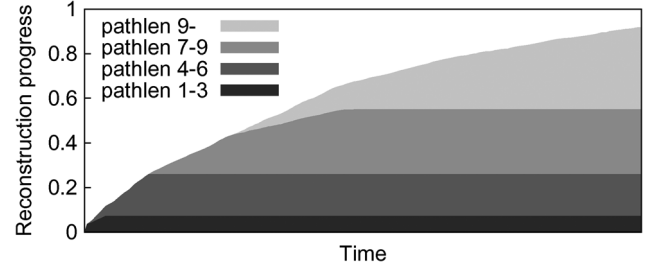


Fig. 9. Reconstruction progress during the whole reconstruction process. Different colors represent the reconstruction progress of paths with different lengths.

TABLE III
IMPACT OF HASH VALUE LENGTH

Hash value length	Correct	Incorrect	Unreconstructed
1 byte	75.3%	20.7%	4.0%
2 bytes	93.0%	0.2%	6.8%
4 bytes	95.2%	0%	4.8%
8 bytes	95.2%	0%	4.8%

B. Impact of Hash Value Length

Furthermore, we use different lengths of the hash value in iPath to evaluate how hash collision affects the overall reconstruction performance. Specifically, we configure the hash value length to be 1, 2, 4, and 8 B and keep the network at its default configuration.

Table III shows the reconstruction results under different hash value length settings. In total, there are 22 187 packet paths that need to be reconstructed. When the hash value is only 1 B, 21 302 paths are reconstructed, and 16 701 of them are correctly reconstructed. When we use 2 B hash value, much more paths are correctly reconstructed. When we use 4 or 8 B of hash value, there is no incorrectly reconstructed path. These results show that 4 B hash value is sufficient to achieve high accuracy. A nice feature is that when a hash collision happens, it will cause one incorrect path, but this incorrect path is not likely to affect other longer paths. The reason is that the hash function will take the incorrect node ID into calculation for longer path verification. It is unlikely that another hash collision happens at that verification. Therefore, the damage of each hash collision is limited.

C. Visualization of Reconstruction Process

We also visualize the path reconstruction process in one simulation with 400 nodes. Fig. 9 shows how all the paths are reconstructed from a temporal perspective. The x -axis is the

time, and y -axis is the reconstruction progress. The reconstruction speed is slightly faster at first. The reason is that verifying short paths requires less computation. Furthermore, different colors in the figures represent the reconstruction process of different path lengths. We can see that longer paths are reconstructed after the short ones, visualizing the iterative boosting process in iPath.

IX. CONCLUSION

In this paper, we propose iPath, a novel path inference approach to reconstructing the routing path for each received packet. iPath exploits the path similarity and uses the iterative boosting algorithm to reconstruct the routing path effectively. Furthermore, the fast bootstrapping algorithm provides an initial set of paths for the iterative algorithm. We formally analyze the reconstruction performance of iPath as well as two related approaches. The analysis results show that iPath achieves higher reconstruction ratio when the network setting varies. We also implement iPath and evaluate its performance by a trace-driven study and extensive simulations. Compared to states of the art, iPath achieves much higher reconstruction ratio under different network settings.

REFERENCES

- [1] M. Ceriotti *et al.*, "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *Proc. IPSN*, 2009, pp. 277–288.
- [2] L. Mo *et al.*, "Canopy closure estimates with GreenOrbs: Sustainable sensing in the forest," in *Proc. SenSys*, 2009, pp. 99–112.
- [3] X. Mao *et al.*, "CitySee: Urban CO2 monitoring with sensors," in *Proc. IEEE INFOCOM*, 2012, pp. 1611–1619.
- [4] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. SenSys*, 2009, pp. 1–14.
- [5] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. MobiCom*, 2003, pp. 134–146.
- [6] Z. Li, M. Li, J. Wang, and Z. Cao, "Ubiquitous data collection for mobile users in wireless sensor networks," in *Proc. IEEE INFOCOM*, 2011, pp. 2246–2254.
- [7] X. Lu, D. Dong, Y. Liu, X. Liao, and L. Shanshan, "PathZip: Packet path tracing in wireless sensor networks," in *Proc. IEEE MASS*, 2012, pp. 380–388.
- [8] 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. SenSys*, 2012, pp. 15–28.
- [9] 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 Trans. Ind. Electron.*, vol. 58, no. 6, pp. 2126–2137, Jun. 2011.
- [10] Y. Liu, K. Liu, and M. Li, "Passive diagnosis for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 18, no. 4, pp. 1132–1144, Aug. 2010.
- [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 Trans. Netw.*, 2013, to be published.
- [12] J. Wang, W. Dong, Z. Cao, and Y. Liu, "On the delay performance analysis in a large-scale wireless sensor network," in *Proc. IEEE RTSS*, 2012, pp. 305–314.
- [13] Y. Liang and R. Liu, "Routing topology inference for wireless sensor networks," *Comput. Commun. Rev.*, vol. 43, no. 2, pp. 21–28, 2013.
- [14] Y. Gao *et al.*, "Domo: Passive per-packet delay tomography in wireless ad-hoc networks," in *Proc. IEEE ICDCS*, 2014, pp. 419–428.
- [15] M. Lee, S. Goldberg, R. R. Kompella, and G. Varghese, "Fine-grained latency and loss measurements in the presence of reordering," in *Proc. ACM SIGMETRICS*, 2011, pp. 329–340.
- [16] Y. Shavitt and U. Weinsberg, "Quantifying the importance of vantage points distribution in internet topology measurements," in *Proc. IEEE INFOCOM*, 2009, pp. 792–800.
- [17] M. Latapy, C. Magnien, and F. Oudraogo, "A radar for the internet," in *Proc. IEEE ICDMW*, 2008, pp. 901–908.
- [18] I. Cunha, R. Teixeira, D. Veitch, and C. Diot, "Predicting and tracking internet path changes," in *Proc. SIGCOMM*, 2011, pp. 122–133.
- [19] A. D. Jaggard, S. Kopparty, V. Ramachandran, and R. N. Wright, "The design space of probing algorithms for network-performance measurement," in *Proc. SIGMETRICS*, 2013, pp. 105–116.
- [20] L. Ma, T. He, K. K. Leung, A. Swami, and D. Towsley, "Identifiability of link metrics based on end-to-end path measurements," in *Proc. IMC*, 2013, pp. 391–404.
- [21] Y. Gao *et al.*, "Pathfinder: Robust path reconstruction in large scale sensor networks with lossy links," in *Proc. IEEE ICNP*, 2013, pp. 1–10.
- [22] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. SenSys*, 2003, pp. 14–27.
- [23] Y. Gao *et al.*, "iPath: Path inference in wireless sensor networks," Tech. Rep., 2014 [Online]. Available: <http://www.emnets.org/pub/gaoyi/tech-ipath.pdf>
- [24] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. IPSN*, 2008, pp. 245–256.
- [25] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "TWIST: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks," in *Proc. REALMAN*, 2006, pp. 63–70.
- [26] R. Lim, C. Walser, F. Ferrari, M. Zimmerling, and J. Beutel, "Distributed and synchronized measurements with FlockLab," in *Proc. SenSys*, 2012, pp. 373–374.
- [27] Z. Li, M. Li, and Y. Liu, "Towards energy-fairness in asynchronous duty-cycling sensor networks," *Trans. Sensor Netw.*, vol. 10, no. 3, pp. 38:1–38:26, 2014.
- [28] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. SenSys*, 2003, pp. 126–137.



networks.



networks.



Yi Gao (S'09) received the B.Eng. degree in software engineering from Zhejiang University, Hangzhou, China, in 2009, and is currently pursuing the Ph.D. degree in computer science at Zhejiang University.

From 2008 to 2009, he worked with the Information System College, Singapore Management University, Singapore, as an exchange student. From 2011 to 2012, he worked with McGill University, Montreal, QC, Canada, as a joint training research student. His research interests include network protocols design and measurement in wireless sensor

Wei Dong (S'08–M'11) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 2005 and 2010, respectively.

He was a Postdoctoral Fellow with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, from 2010 to 2011. He is currently an Associate Professor with the College of Computer Science, Zhejiang University. His research interests include networked embedded systems and wireless sensor

Chun Chen (M'13) received the Bachelor of Mathematics degree from Xiamen University, Xiamen, China, in 1981, and the M.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1984 and 1990, respectively.

He is a Professor with the College of Computer Science and the Director of the Institute of Computer Software, Zhejiang University. His research interests include embedded system, image processing, computer vision, and CAD/CAM.



Jiajun Bu (M'06) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1995 and 2000, respectively.

He is a Professor with the College of Computer Science and the Deputy Director of the Institute of Computer Software, Zhejiang University. His research interests include embedded system, mobile multimedia, and data mining.

Prof. Bu is a member of the Association for Computing Machinery (ACM).



Wenbin Wu received the B.S. degree from Zhejiang University of Technology, Hangzhou, China, in 2012 and is currently a graduate student with the College of Computer Science, Zhejiang University.

His research interests include fine-grained measurement in wireless sensor networks and network tomography techniques in modern communication networks.



Xue Liu (S'99–M'06) received the B.S. degree in Mathematics and M.S. degree in automatic control from Tsinghua University, Beijing, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2006.

He is an Associate Professor with the School of Computer Science, McGill University, Montreal, QC, Canada. He has also worked as the Samuel R. Thompson Associate Professor with the University of Nebraska–Lincoln, Lincoln, NE, USA, and HP Labs, Palo Alto, CA, USA. He has been granted one US patent and filed four other US patents and published more than 150 research papers in major peer-reviewed international journals and conference proceedings. His research interests are in computer networks and communications, smart grid, real-time and embedded systems, cyber-physical systems, data centers, and software reliability.

Dr. Liu received the 2008 Best Paper Award from the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS and the First Place Best Paper Award of the ACM Conference on Wireless Network Security (WiSec 2011).