Reconstructing the Routing Topology in Dynamic and Large Scale Networks

S.Dharani
Student,Departmentof Information Technology
M.Kumarasamy College of Engineering,Karur
sridharanidgl@gmail.com

S.Kalaiyarasi
Student,Departmentof Information Technology
M.Kumarasamy College of Engineering,Karur
kalaiyarasi.kssakthivel@gmail.com

V.Jeevitha
Student,Departmentof Information Technology
M.Kumarasamy College of Engineering,Karur
jeevithamkce@gmail.com

K.Megala
Student,Departmentof Information Technology
M.Kumarasamy College of Engineering,Karur
megalak22@gmail.com

***Abstract*— In wireless sensor networks(WSNs), sensor nodes are usually self-organized, delivering data to a central sink in a multi-hop manner. Reconstructing the per-packet routing path enables fine-grained diagnostic analysis and performance optimizations of the network. The performances of existing path reconstruction approaches, however, degrade rapidly in large scale networks with lossy links. This paper presents Pathfinder, a robust path reconstruction method against packet losses as well as routing dynamics. 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 sink side, Pathfinder infers packet paths from the compressed information and employs intelligent path speculation to reconstruct the packet paths with high reconstruction ratio. We propose an analytical model to analyze the performance of Pathfinder. We further compared Pathfinder with two most related approaches using traces from a large scale deployment and extensive simulations. Results show that Pathfinder outperforms existing approaches, achieving both high reconstruction ratio and low transmission cost.**

***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 CO_2 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.

Reconstructing the routing path of each received packet at the sink side is an effective way to understand the network's complex 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, most existing delay and loss measurement approaches [9], [14] assume that the routing topology is given as *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.

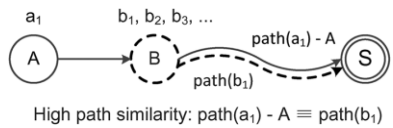

Fig. 1. Example to illustrate the basic idea of iPath.

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., *path* a_1 - *A* , which means the subpath by removing node A from *path* a_1))as one of B's packet, say b_1 , i.e., *path* (a_1) = *(A, 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, 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 net- work 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 ac- curacy. However, as described in Section V-C, MNT is vulner- able 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 informa- tion. 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

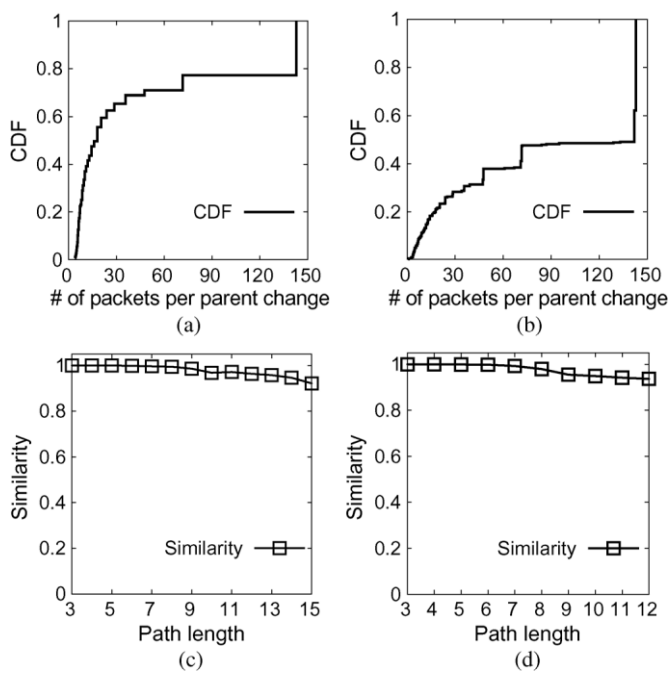

Fig. 2. Path similarity and routing dynamics in two large-scale deployed sensor networks. (a) Dynamic of CitySee. (b) Dynamic of GreenOrbs. (c) Similarity of CitySee. (d) Similarity of GreenOrbs.

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 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 iter- ative 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.

Algorithm 1: The iterative boosting algorithm

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

Output: The routing paths of packets

Procedure ITERATIVE-BOOSTING
  $P_n \leftarrow P_{\text{init}}$
  while $P_n \neq \{\}$ **do**
    $P_{nn} \leftarrow \{\}$
    for each packet k in P_n **do**
      for each packet i in P_x **do**
        res = RECOVER(k, i)
        if $res \equiv True$ **then**
          $P_{nn} \leftarrow P_{nn} \cup i$
          $P_x \leftarrow P_x - i$
    $P_n \leftarrow P_{nn}$
procedure RECOVER (k, i)
  if $len(i) - len(k) \notin \{1, 2\}$ **then**
    return **False**
  if $len(i) - len(k) \equiv 2$
    if $hash(o(i), p(i), path(k)) \equiv h(i)$ **then**
      $path(i) \leftarrow (o(i), p(i), path(k))$ //Case2
      return **True**
    return **False**
  if $len(i) - len(k) \equiv 1$ **then**
    if $hash(o(i), path(k)) \equiv h(i)$
      $path(i) \leftarrow (o(i), path(k))$//Case1
      return **True**
    if $hash(o(i), p(i), path(k) - o(k)) \equiv h(i)$ **then**
      $path(i) \leftarrow (o(i), p(i), path(k) - o(k))$ //Case3
      return **True**
    return **False**

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 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_x 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 '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).

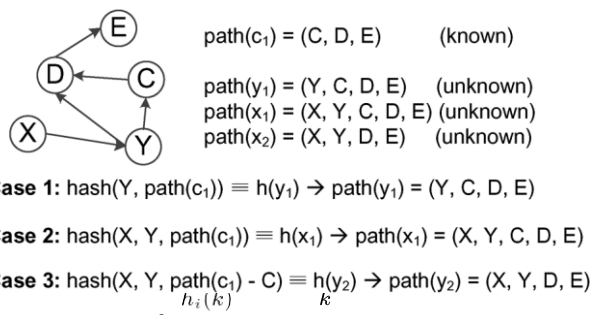

Fig. 3. Example to illustrate three cases of reconstructing long paths based on short paths in the iterative boosting algorithm. X, Y, etc., are nodes, and x1, y1, etc., are packets originated from different nodes.

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 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 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 light- weight 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 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).

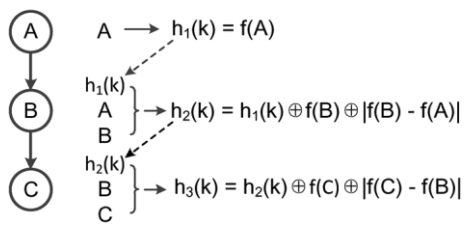

Fig. 4. PSP-Hashing function. Each node in the routing path takes three inputs and updates the hash value in packet . Note that we use $h_i(k)$ to denote the hash value in packet at the th hop, instead of a function defined on packet K itself.

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, hash(A, B) and 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 me. However, they are not desirable due to their high computational and 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 light weight 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 ith 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:

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

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 μs) . 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, $hash((A, B, C, D))$ is different with $hash((C, B, A, D))$. In practice, paths are 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 $hash((A, B, C, D))$ equals to $hash((C, B, A, D))$. 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 $hash(o(path_x), (B, A, D))$ equals $hash(path_x)$ since $hash((A, B, C, D))$equals $hash((C, B, A, D))$. 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 recon- structed 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 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.

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

```
1: procedure FAST-BOOTSTRAPPING
2:     path(k) ← (o(k), p(k))
3:     n ← p(k)
4:     while n ≠ sink do
5:         u ← arg max_x t̂_g(x) + Δ_x^u for all x : o(x) ≡ n
6:             ∩ t̂_g(x) + Δ_x^u < t̂_g(k) − Δ_k^l
7:         v ← arg min_x t̂_g(x) + Δ_x^v for all x : o(x) ≡ n
8:             ∩ t̂_g(x) − Δ_x^l < t_s(k)
9:         if u ≡ {} or v ≡ {} or pc(u) ≠ pc(v) then
10:            break
11:    path(k) ← path(k) ∪ p(n)
12:    n ← p(n)
13: return path(k)
```

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.

a. 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.

b. 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.

c. 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, $\prod_{z \in \text{path}(k)} \text{PPR}(z)$.

d. The average node degree .

e. As mentioned in the fast bootstrapping algorithm, a *stable period* of a node is a period in which the node does not change parent.

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 recon- struction 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:

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

Where $S_{MNT}(j)$ is the ratio of stable periods of node j. Then, we calculate by dividing the expected number of stable periods in one cycle by the total periods λ in one cycle

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 bootstrap- ping algorithm reconstructs the routing path of a packet hop by hop. When the sink reconstructs the path of a packet to a for- warder F, it can reconstruct the next-hop only when the packet is in one of Fs stable periods. Therefore, the probability of a is successful reconstruction of the fast bootstrapping algorithm 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 $S(j)$.

VII. 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.