A Practical Approach for Provenance Transmission in Wireless Sensor Networks[☆]

S. M. Iftekharul Alam

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 47907

Sonia Fahmy

Department of Computer Science, Purdue University, West Lafayette, IN, 47907

Abstract

Assessing the trustworthiness of sensor data and transmitters of this data is critical for quality assurance. Trust evaluation frameworks utilize *data provenance* along with the sensed data values to compute the trustworthiness of each data item. However, in a sizeable multi-hop sensor network, provenance information requires a large and variable number of bits in each packet, resulting in high energy dissipation due to the extended period of radio communication. In this paper, we design energy-efficient provenance encoding and construction schemes, which we refer to as Probabilistic Provenance Flow (PPF). Our work demonstrates the feasibility of adapting the Probabilistic Packet Marking (PPM) technique in IP traceback to wireless sensor networks. We design two bit-efficient provenance encoding schemes along with a complementary vanilla scheme. Depending on the network size and bit budget, we select the best method based on mathematical approximations and numerical analysis. We integrate PPF with provenance-based trust frameworks and investigate the trade-off between trustworthiness of data items and transmission overhead. We conduct TOSSIM simulations with realistic wireless links, and perform testbed experiments on 15-20 TelosB motes to demonstrate the effectiveness of PPF. Our results show that the encoding schemes of PPF have identical performance with a low bit budget ($\sim$ 32-bit), requiring 33% fewer packets and 30% less energy than PPM variants to construct provenance. With a two-fold increase in bit budget, PPF with the selected encoding scheme reduces energy consumption by 46-60%.

Keywords: provenance; trust framework; energy-efficiency; sensor networks

1. Introduction

New micro sensors have enabled wireless sensor networks (WSNs) to gather real-time data from the physical world [1, 2]. Planet-wide sensor networks [3, 4], sensor networks for large-scale urban environments [5], and physical infrastructure systems [6] indicate potential deployments of multi-hop networks consisting of hundreds of sensor nodes. In such networks, data produced by the sensors are collected at the base station and made available to decision makers for further analysis. As the quality of decision making is critically dependent on the quality of transmitted information [5], trustworthiness of information and information-transmitting nodes is important [7]. In a multi-hop network, *provenance* includes knowledge of the originator and processing path of data since its generation. While a few provenance-based trust evaluation frameworks have been proposed [8, 9], they do not consider *energy dissipation* due to provenance transmission.

Provenance of a data item can be represented by a tree that is embedded as meta-data with the item, and updated along the path used to forward the item to the base station [9]. In this case, every intermediate node carries provenance of length proportional to the hop count between that node and the originator of the data item. In a network with a large diameter (hop count), this increased meta-data length results in an extended period of radio communication and energy dissipation at every intermediate node. We consider a real deployment of a 46-hop network [10] in our simulations, and observe that aggregated energy dissipation of the network

[☆]This work has been sponsored in part by NSF grants CNS-0964294 and CNS-0964086.

Email addresses: alams@purdue.edu (S. M. Iftekharul Alam), fahmy@cs.purdue.edu (Sonia Fahmy)

increases by 27% when a traditional trust framework is employed. Although large networks can be hierarchically organized [11], they still require a significant number of hops [12], with non-negligible energy usage for provenance transmission.

Provenance encoding and construction is similar in nature to the well-known *IP traceback* problem [13, 14]. IP traceback aims to determine the forwarding paths of spoofed packets in the Internet. Among the many proposed solutions to this problem, Probabilistic Packet Marking (PPM) can most easily be adapted to WSNs [15]. We have shown that direct application of PPM to WSNs is infeasible since it embeds a single node identifier in each packet, and hence requires a large number of packets to construct the forwarding path [16]. Instead, we propose a new approach, Probabilistic Provenance Flow (PPF), where a connected subgraph of the forwarding path is probabilistically embedded into a packet. PPF includes three new bit-efficient provenance encoding schemes that quickly construct provenance of an arbitrarily large multi-hop network.

We integrate a simple but robust scheme into PPF to handle topological changes. Since encoded provenance is matched against previously constructed provenance graphs at the base station, we can reduce decoding errors to negligible levels and speed up convergence. We also integrate PPF with provenance-based trust frameworks and explore how trust scores evolve faster with data items having dissimilar provenance than with the items having shared provenance. This study exposes the trade-off between trustworthiness or provenance dissimilarity of data items and transmission overhead: making provenance more dissimilar increases transmission overhead. We investigate this trade-off and propose a solution to provide decision makers with a tunable parameter to control the extent of provenance dissimilarity and transmission overhead.

We perform extensive simulations using TOSSIM to demonstrate the performance of PPF in a highly dynamic and asymmetric network. We further evaluate PPF using a testbed consisting of 15-20 TelosB motes in different settings. Our simulation and testbed results show that PPF with the selected encoding scheme can consume up to 46-60% less energy and converge with 45% fewer packets than the traditional approach, which significantly increases the network life-time.

The remainder of this paper is organized as follows. We formulate the problem of energy-efficient provenance transmission in Section 2. Section 3 discusses related work. Section 4 explains three different encoding schemes for PPF. We discuss the corresponding approaches to decode and construct provenance in Section 5. Section 6 discusses integration of PPF with provenance-based trust frameworks. In Section 7, we examine the parameter selection for one of the encoding methods. We analyze the bit requirements for embedding provenance using all encoding schemes in Section 8. Section 9 and 10 present simulation and testbed results, respectively. Finally, Section 11 concludes the paper.

2. Problem Formulation

2.1. Network Model

We consider a multi-hop WSN where changes in topology due to failure or mobility can occur, but are infrequent. We make the following assumptions regarding the network and traffic:

(1) A Base Station (BS) acts as a central command authority and the root of a routing tree. It has no resource constraints and cannot be compromised by an attacker.

(2) Sensor nodes monitor their surroundings and periodically report to the base station or their designated cluster head (if any).

(3) Multiple sensors are used to monitor an event. Within a particular time window, independent observations obtained at cluster heads (if any) or the base station from different sensors are concerned with the same event.

(4) A provenance-based trust management method such as [8, 9] is used in the application layer to evaluate and manage trust in an adaptive manner. More details can be found in [17].

2.2. Problem Statement

We consider a network of N nodes, where the maximum length (depth) of any forwarding path (tree) is L. Assume that the maximum number of bits that can be used to embed provenance information in a single packet is B. Based on this bit budget, there is an integer $m, 1 < m \leq L$ such that at most m consecutive node identities (that is, $m-1$ consecutive edges) can be embedded into a single packet. We must perform the following operations:

(1) **Provenance Embedding**: In a forwarding tree $G = (V, E)$ rooted at the base station, each node $n_i \in V$ makes an independent decision whether to embed its identity into the packet, starting a connected sub-graph, with probability p_i. We need to design a provenance embedding method to carry a partial path $P =< n_{i_1}, n_{i_2}, \cdots n_{i_m} >$ into a single packet where $n_{i_j} \in V, 1 \leq j \leq m$ and $(n_{i_k}, n_{i_{k+1}}) \in E, 1 \leq k \leq$

$m-1$. This problem is a simple extension of the edge sampling approach in IP traceback [13].

(2) **Provenance Construction**: At the base station, we must construct the entire provenance tree $G = (V, E)$ by exploiting partial path information collected from a number of received packets, with an upper bound on the number of packets required to construct the provenance.

(3) **Provenance Evolution**: After topological changes, e.g., due to failures or mobility, we must bound the time that it takes to reflect the changes in the constructed provenance.

(4) **Provenance Delivery**: We must specify interfaces or methods to deliver the constructed provenance to an existing provenance-based trust framework.

3. Related Work

Few provenance-based trust frameworks have been proposed to date [8, 9]. These frameworks do not consider energy-efficiency in WSNs.

The problem of provenance transmission is related to the IP traceback problem that determines the forwarding path of spoofed packets [18]. IP traceback methods include hop-by-hop tracing [19, 20], out-of-band ICMP traceback [21], and in-band probabilistic packet marking [13, 14]. Hop-by-hop tracing is not well-suited to WSNs due to its large storage requirement. Hot-spot based traceback methods designed for mobile ad-hoc networks [22, 23] store packet information at the nodes, and traceback is performed hop-by-hop to determine the hot-spot where the attacker is located. In our case, provenance information is continuously required at the base station to compute trust scores of descendant nodes. Hot-spot based methods would incur unnecessary delay in trust score calculation. Out-of-band ICMP traceback requires out-of-band communication and increased bandwidth which limit its usability in resource-constrained WSNs.

In this work, we adapt Probabilistic Packet Marking (PPM) since it does not require additional storage or out-of-band communication. PPM assumes static routes which may not hold in our case. Additionally, PPM requires a significant number of packets to construct the forwarding path. Network coding variants of PPM [24, 25] require fewer packets to construct the forwarding path. Network coding approaches, however, have a high computational complexity and increase the length of the packet, as marking coefficients are transmitted with the packet. Cheng et al. [26] determine the optimal marking probability for each node to reduce the number of packets required to construct the forwarding path.

Multi-hop Network Tomography (MNT) [27] is a recently proposed algorithm for reconstructing the packet path, the per-hop arrival order, and the per-hop arrival timing of individual packets. The algorithm is, however, particularly designed for networks where nodes have the dual functionality of both generating and forwarding packets. In contrast, our approach is more generic and does not make specific assumptions about the roles of nodes.

Sultana et al. [28] proposed a secure provenance scheme for wireless sensor networks that uses in-packet Bloom filters to encode provenance. However, even for a 14-hop path, their scheme requires 240 bits to embed provenance information. This limits the practical usability of this scheme in large scale multi-hop networks.

4. Probabilistic Provenance Embedding

In this section, we present three provenance embedding schemes as part of our Probabilistic Provenance Flow (PPF) approach. All three methods incorporate node identifiers into a packet probabilistically and only differ in how they encode these identifiers.

4.1. Juxtaposition of Ranks

In the *rank method*, instead of embedding the node ID directly into a packet, $rank(\text{ID})$ (defined in Definition 1 below) of the node is embedded, since every node ID is uniquely identifiable by its rank, and the rank would need fewer bits than the identity.

Definition 1. *Consider $U = \{ID_1, ID_2, \cdots ID_N\}$ as the set of N node IDs. There is a permutation of U, $\sigma(U) = \{ID_{a_1}, ID_{a_2}, \cdots ID_{a_N}\}$, such that, $ID_{a_j} < ID_{a_{j+1}}$, for $1 \leq j \leq N-1$. Rank of any node ID, $ID_i \in U$, denoted as $rank(ID_i)$, is the position of ID_i in $\sigma(U)$.*

Assume that the packet meta-data has space to hold identities of up to m nodes. We use a counter of $\log_2 m$ bits to track the number of already embedded ranks in the packet. Initially, the buffer and counter contain zeroes. Every node n_i decides to start a connected subgraph with probability p_i. Once it decides to do so, it overwrites the previous information by doing the following: it zeroes out the entire provenance field and then embeds its rank at the beginning of the buffer and sets the counter to one. If a node decides not to overwrite, it checks for empty buffer space using the counter

field. If there is space, it adds its rank into the first available slot in the buffer and increments the counter. Fig. 1 shows an example of this method where the buffer space can hold at most three node identities in a single packet.

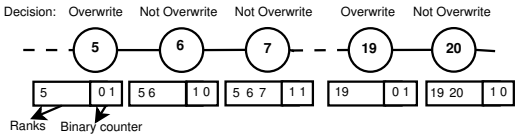


Figure 1: Provenance encoding using juxtaposition of ranks (numbers inscribed in the circles indicate *rank* of nodes).

4.2. Prime Multiplication

Our second encoding scheme, the *prime method*, is based on prime multiplication. In a reasonably large network, this method can embed more node IDs within same number of bits (on the average), compared to the rank method. To the best of our knowledge, this method has not been used in any prior work.

Table 1: Bit requirements for multiplication of m node IDs, picked randomly from the first f prime numbers.

f	$m = 3$		$m = 4$		$m = 6$	
	Avg	Max	Avg	Max	Avg	Max
500	30.80	36	40.76	47	59.82	69
1000	34.25	39	45.35	52	67.02	76
2000	37.82	42	49.84	56	73.64	83
5000	42.33	47	55.83	62	82.45	93

Definition 2. *Let P_n be the largest prime number that is less than or equal to the positive integer n. We define the set of usable IDs, $Q_{P,s}$ where P is a prime number and s is a positive integer:*

$$Q_{P,s} = \{n \in \mathbb{N} \mid 2 \leq n \leq P \ \ and \ \ 0 \leq n - P_n \leq s\}$$

Definition 3. *For any positive integer $n \in Q_{P,s}$, for some P and s, we define two functions:*

- $prime(n)$ = P_n *(the largest prime number that is less than or equal to n).*
- offset$(n) = n - P_n$.

The prime method is motivated by the idea of using prime numbers as node IDs and encoding a set of IDs through their multiplication which can be uniquely factorized. However, prime multiplication incurs computational and spatial overhead when the participating prime numbers become larger. As shown in Table 1, the average number of bits required to multiply m prime numbers increases with the increasing size of the domain of these numbers. This shows the infeasibility of using prime numbers directly as node IDs. Thus, we define $Q_{P,s}$ (Definition 2) to ensure that node IDs will not differ by more than s from their nearest prime numbers where s is referred as the ***spread factor***. Then, we encode a sequence of node IDs by multiplying their nearest prime numbers and summing up the corresponding offset values (Definition 3). Before describing the details of encoding process, we describe the assignment of node IDs using set $Q_{P,s}$.

4.2.1. Node ID Assignment

For a network of N nodes, we pick a set $Q_{P,s} = \{q_1, q_2, \cdots q_z\}$ with the smallest z such that $z \geq N$. An in-place algorithm is used to produce a random permutation of $Q_{P,s}$, $\sigma(Q_{P,s}) = \{q_{a_1}, q_{a_2}, \cdots q_{a_z}\}$ and members of $\sigma(Q_{P,s})$ are assigned to all N nodes sequentially. For example, in an 8-node network, we can pick IDs for the nodes from a random permutation of $Q_{11,7} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$.

For a given number of nodes (N), the bit requirements for prime multiplication increase the most when $s = 0$, which makes $Q_{P,s}$ nothing but a set of prime numbers that are less than or equal to P ($= N$th prime number). With an increasing value of s, the set $Q_{P,s}$ can contain numbers upper-bound by relatively smaller value of P ($\ll N$th prime number). By tuning s, the largest element (P) of $Q_{P,s}$ can be made close to the total number of nodes (N). This brings about an interesting trade-off: reduction of the number of bits required for prime multiplication versus increase in the number of bits required for summation of offsets. We will investigate this trade-off in Section 7.

4.2.2. Encoding Process

To store provenance information, we divide the provenance buffer into two parts: *product* and *offset*. Every node n_i has an ID, say ID_i, that is a member of $Q_{P,s}$ for some P and s. As with the rank method, once a node n_i decides to start a connected sub-graph, it clears the provenance buffer. It then inserts $prime(ID_i)$ into the product part and *offset*(ID_i) into the offset part (Definition 3). If a node n_j decides not to overwrite, it retrieves the values stored in the product and offset parts. It then multiplies the value of the product with $prime(ID_j)$, adds *offset*(ID_j) to the offset, and stores the newly calculated values into the respective parts. Fig. 2 shows an example with $m = 2$.

We no longer need a counter field to track the number of node identities encoded in the provenance buffer because there is always a unique prime factorization of

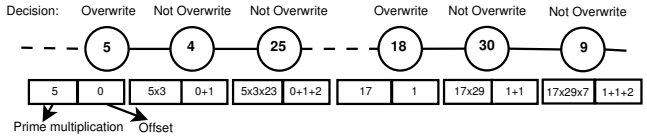


Figure 2: Provenance encoding using prime multiplication (numbers inscribed in the circles indicate ID of nodes).

the *product* part which gives the number of participating nodes.

4.3. Rabin Fingerprints

The prime method can typically accommodate more node identities (m) than the rank method, but prime multiplication results increase in size as N and m increase. In order to embed more node IDs into a single packet without requiring additional bits and excessive computational complexity, we investigate *Rabin fingerprints* [29].

A *Rabin fingerprint* calculates a near-perfect and space-efficient unique representation of a sequence of bits. For a sequence of bits $n_1, n_2, \cdots n_m$, of length m, the Rabin fingerprint is given by the following expression, where α and M are constant integers:

$$RF(n_1, n_2, \cdots n_m) = (n_1\alpha^{m-1} + n_2\alpha^{m-2} + \cdots + n_m) \mod M$$

The fingerprint of the concatenation of two sequences X and Y can be computed as follows:

$$\begin{aligned} RF(X||Y) &= RF(RF(X)||Y) \\ &= RF(RF(X) \times \alpha^l) + RF(Y) \end{aligned}$$

where, $||$ represents concatenation and l is the length of Y.

4.3.1. Encoding Fingerprints

The partial path traversed by a packet can be considered as a bit sequence of IDs of the nodes on that partial path. We aim at transmitting the fingerprint of that bit sequence sequence instead of transmitting the actual sequence. Every node uses two constant integers α and M to compute its fingerprint. As the packet traverses the path, we could easily compute the contribution of every node to the fingerprint and add it to the contributions of its predecessor nodes on that path. For example, if the packet traverses the partial path $< n_1, n_2, \cdots n_m >$, node $n_i, 1 \leq i \leq m$ has a contribution of $ID_i\alpha^{b(m-i+1)}$ to the fingerprint associated with that path. Here, b is the number of bits required to represent a single ID. However, the incremental sum of these contributions requires a large and variable number of bits in the packet which is undesirable. Hence, we exploit the concatenation property of Rabin fingerprints, which allows any node n_k to compute the fingerprint of node IDs from ID_{n_1} to ID_{n_k} by concatenating its own ID (ID_{n_k}) to the fingerprint value of previous nodes ID_{n_1} to $ID_{n_{k-1}}$. The following equation makes this claim clear:

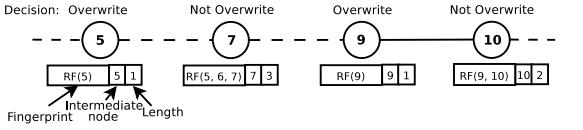
$$\begin{aligned} &RF(ID_{n_1}, ID_{n_2}, \cdots ID_{n_k}) \\ &\quad = RF(RF(ID_{n_1}, ID_{n_2}, \cdots ID_{n_{k-1}})||ID_{n_k}) \end{aligned} \tag{1}$$

Thus, every node on a path can update the fingerprint without requiring any extra bits as the fingerprint value is always less than the divisor M. Note that since we perform all arithmetic operations over $\mathbf{Z_2}$, the concatenation operation and fingerprint calculation (which require only shift and XOR operations) can be performed in linear time [30]. The time complexity to calculate fingerprints can be further improved by using a precomputed lookup table as discussed in Appendix A.

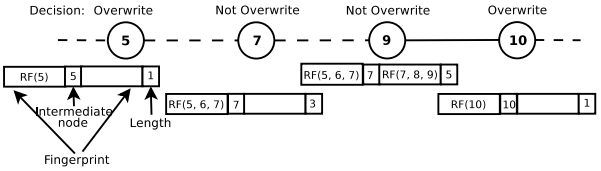
To store provenance information, we divide the provenance buffer into three fields: *fingerprint*, *intermediate node*, and *length*. As in other PPF methods, every node n_i decides to start a connected sub-graph with probability p_i. Once it decides to do so, it clears the buffer and inserts ID_i and 1 into the fingerprint and length fields, respectively. If a node n_j decides not to overwrite, it retrieves the values stored in the fingerprint and length fields. If the length is less than m, it updates the current fingerprint value (say X) by computing $RF(X||ID_j)$ and increments the current value of the length field by one. If the newly computed length is less than or equal to m, ID_j is stored in the *intermediate node* field that will aid in decoding the provenance as discussed later.

4.3.2. Partitioning Fingerprints

Since we need to exploit previous knowledge about node ordering to retrieve provenance information from the fingerprint (as we will discuss in the next section), transmitting large partial provenance information using fingerprints may not always be advantageous. In WSNs, nodes are vulnerable and error-prone and the routing path may change due to the lossy nature of the wireless medium. This may cause inconsistency between the current and the previously stored ordering among nodes, making the entire fingerprint-based provenance information useless. To mitigate this problem, we divide the fingerprint field into $r(r > 1)$ parts around the



(a) Embedding provenance with non-partitioned fingerprint



(b) Embedding provenance with $m = 5$ and $r = 2$

Figure 3: Provenance encoding using Rabin fingerprints (numbers inscribed in the circles indicate ID of nodes).

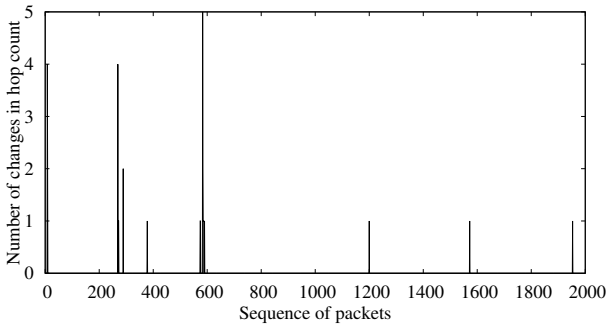


Figure 4: Snapshot of topological changes for 2000 packets in a 10 × 10 grid network.

$(r-1)$ *intermediate nodes* so that changes in ordering in one part do not affect other parts and changes in ordering can be reflected.

Each part contains fingerprints of at most $\lceil \frac{m+r-1}{r} \rceil$ node IDs where the length field indicates the total number of participating node IDs. For example, assume we wish to embed information about a partial path $< n_1, n_2, \cdots n_m >$ into a single packet with $m = 7$ and $r = 2$. As the packet traverses the network, node n_4 becomes the intermediate node and the first part of the fingerprint contains $RF(ID_1, ID_2, ID_3, ID_4)$ and the second part contains $RF(ID_4, ID_5, ID_6, ID_7)$.

Both parts of the fingerprint are calculated according to equation 1 and the length field indicates the combined length of the both parts. Note that, a non-partitioned fingerprint is a special case where the intermediate node corresponds to the lone fingerprint. Fig. 3(a) depicts an example of the non-partitioned case and Fig. 3(b) depicts the partitioning approach with $m = 5$ and $r = 2$.

4.4. Handling Link Changes

The decoding process for prime and fingerprint methods requires a priori knowledge of order of nodes (as discussed in Section 5). Since topological changes are prevalent in wireless sensor networks, we need to keep node order information up-to-date so that the prime and fingerprint encoding methods can correctly decode provenance. A straightforward solution is to invoke the rank approach every $t_{embedding}$ seconds. A small value of $t_{embedding}$ reduces the benefits of applying the bit-efficient prime and fingerprint methods, while longer $t_{embedding}$ increases decoding error and reduces the overall effectiveness of the trust framework.

To study link change, we conduct a simple simulation experiment. Fig. 4 shows a snapshot of link changes for 2000 packets transmitted from a source to the base station in a highly asymmetric 10 × 10 grid network. Here *link change* for a packet denotes the number of next hop changes on the way from the source to the base station with respect to the path followed by the preceding packet. We observe that the time interval between two successive topological changes varies, making it difficult to choose a fixed value for $t_{embedding}$. Thus, we design a reactive approach (shown in Fig. 5) to handle link changes by exploiting *next hop* information available from the network layer. The goal of our approach is to rapidly detect changes and to transmit provenance of nodes that are part of the changed links with order information among them.

Each node maintains a *parent list* containing its most recent *next hops* (towards the base station). When a forwarder node receives a packet, it updates this list if the current *next hop* is not present in the list. It also switches a single bit (which we call *path changed*) in the provenance field of the packet to 1 and embeds the rank of its own ID as provenance. Every node receiving the packet with the *path changed* bit set to 1 concatenates the rank of its ID with the existing provenance information in the packet. Note that once the *path changed* bit of a packet is set to 1, no forwarder node is allowed to override the provenance information which makes transmission of provenance of changed links the highest priority. If the current *next hop* is present in the parent list, the node can follow any of the probabilistic encoding schemes discussed earlier.

We recommend that the size of the *parent list* of a node be computed as a function of the *number of neighboring nodes*, the *rate of link changes*, and the *packet reception rate* (PRR) at the base station for that node. When PRR at the base station for a node is low, topological change information passed from that node may be lost. With a smaller parent list, the node will record

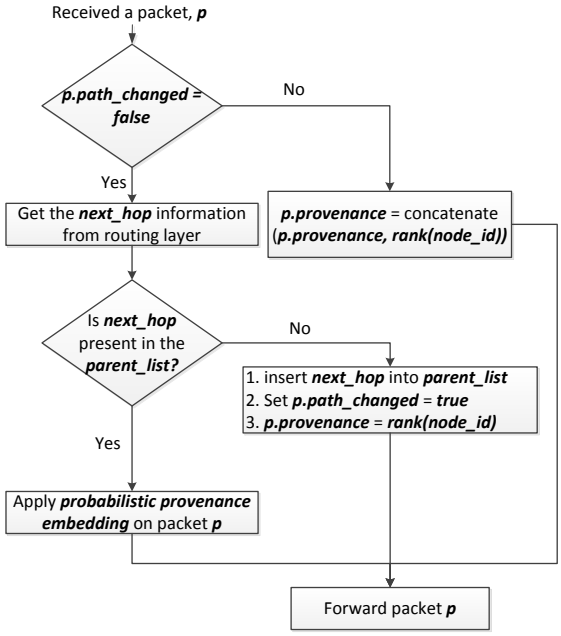


Figure 5: Handling topological changes in an intermediate node.

fewer next hop changes out of the lost ones. This increases the probability of retransmitting lost topological change information in a subsequent packet. If the rate of link change is low, a smaller parent list for a node suffices as well. When PRR at the base station for a node is good, a larger parent list allows a node to record more next hop changes and ensures faster transition to probabilistic encoding schemes. In our simulation experiments, we found that the performance of PPF is not highly sensitive to the size of the parent list, and chose g for a $g \times g$ grid network.

Although our proposed topological change handling scheme requires an extra bit in the provenance field, it offers several advantages:

- Topological changes are rapidly propagated and reflected in the provenance graph constructed at the base station.
- Rank-based decoding can be used to decode the provenance information of a packet with the *path changed* bit set to 1.
- Nodes can automatically utilize an efficient encoding scheme (e.g., prime or fingerprint) as soon as no further topological changes are observed.

5. Decoding and Constructing Provenance

The provenance buffer of a packet is examined at the base station to retrieve the embedded partial provenance (or path) information. With the rank embedding approach, we can easily extract the embedded identities from the provenance buffer, since we have the length field, and the rank of each node ID uses a fixed number of bits. However, with both the prime and fingerprint embedding methods, we assume that information about ordering among nodes is known from a previously constructed provenance graph, $G_{pre} = (V_{pre}, E_{pre})$. Here, V_{pre} is the set of node IDs and E_{pre} is the set of edges among these nodes indicating provenance flow.

5.1. Decoding Process for Prime Method

We apply a standard prime factorization algorithm over the product part of the provenance buffer to retrieve the set of nearest prime numbers (say, $X = \{X_1, X_2, \cdots X_m\}$). Then, we use Depth-First-Search (DFS) *with* backtracking over G_{pre} to find all the possible paths consisting of m node IDs whose nearest prime numbers form any permutation of X. We need to modify DFS to compare the nearest prime number to a particular node ID with the members of set X while visiting that node, and track the offset of the ID when a match is found. With this modification, DFS with backtracking outputs all possible sets of node IDs whose nearest prime numbers form a permutation of X. Since every ID does not differ from its nearest prime number by more than s, we can find at most s^m such sets. For each such set, we sum up the offset values and calculate the difference from the offset value retrieved from the received packet. If the difference is zero, we record the matched set as the retrieved provenance.

5.2. Decoding Process for Fingerprint Method

Upon reception of a packet, we retrieve the r fingerprint(s), associated intermediate node ID(s), and the length field indicating the number of participating node IDs. For every intermediate node, $ID_i, 1 \leq i \leq r-1$, we perform a DFS with backtracking over $G_{pre} = (V_{pre}, E_{pre})$ with the following modifications:

- Set ID_i is the root for DFS with backtracking.
- Search all the nodes within $\lceil \frac{m+r-1}{r} \rceil$ hops away from the node ID_i and compute the Rabin fingerprint using a pre-computed lookup table as discussed in Appendix A.

- After computing every fingerprint of length $\lceil \frac{m+r-1}{r} \rceil$, we compare them to the retrieved fingerprints RF_i and RF_{i+1}. If a match is found, we record the matched path as provenance of the received packet.

Note that, for every intermediate node, we are searching a smaller portion of the graph G_{pre} using DFS with backtracking.

False Positive Rate for Fingerprints

Assume that we have at most $x = \lceil \frac{m+r-1}{r} \rceil$ node IDs per partition, where m is the total number of node IDs embedded per packet. Since decoding each pair of partitions is independent of others, it suffices to analyze the false positive probability of fingerprinting a path of x node IDs originating from a particular intermediate node ID. Assume that there are n such paths in the provenance graph. Then, the false positive probability is $\leq \frac{n^2 x.b}{2^k}$, where k is the number of bits used for fingerprinting and b is the length of the bit representation of one node ID.

If the maximum fan-out of the network is f, then n is upper-bound by f^{x-1} which gives,

$$\begin{aligned} \textit{False positive probability} &\leq \frac{f^{2(x-1)} x.b}{2^k} \\ &= \frac{b(m+r-1) f^{\frac{2(m-1)}{r}}}{r.2^k} \end{aligned} \tag{2}$$

5.3. Construction of Provenance

Depending on the network characteristics and bit budget, decision makers may choose one of the three provenance encoding schemes as the default one. When a packet is received at the base station, the *path changed* bit of the packet is checked to determine the encoding method that was used to embed provenance. If the *path changed* bit is set to 1, provenance (i.e. partial forwarding path) of the received packet is extracted using the decoding process of rank approach. Otherwise, the decoding process for the default scheme is used to extract embedded provenance.

Fig. 6 shows a block diagram of provenance construction using PPF. The process of provenance construction maintains two data structures: (i) $G = (V, E)$ representing the current provenance graph and (ii) $G_{pre}(V_{pre}, E_{pre})$ encompassing previously constructed provenance. Once we have decoded partial path information from the received packet, provenance construction is straightforward. After collecting sufficient packets with embedded provenance (i.e., when we have at least one ID from each node), we combine the partial paths to produce the complete provenance graph, $G = (V, E)$. Here, V is the set of nodes and for some $v_i \in V, v_j \in V$, $(v_i, v_j) \in E$ iff (ID_i, ID_j) belongs to some partial provenance encoded in a received packet. Before the next round of provenance construction, we set $V_{pre} = V_{pre} \cup V$ and $E_{pre} = E_{pre} \cup E$ to update $G_{pre}(V_{pre}, E_{pre})$ which is used by the decoding processes of prime and fingerprint methods. Note that the initial G is constructed from the partial provenance embedded in the packets carrying changed path information (*path changed* bit 1) only.

With the prime and fingerprint approaches, decoding errors can occur when a packet carrying changed path information is lost in the network and the base station is unaware of the changes. Subsequently sent packets containing encoded information of the changed path will cause decoding errors at the base station. Further processing can be employed to recover from these errors, such as checking other combinations of partial paths by checking edges between nodes on the recorded path and the nodes that are 1 or 2-hop away from them. These extensions will be the subject of our future work.

5.4. Complexity Analysis

Decoding using the rank method is straightforward and takes only $O(m)$ time, where m is the maximum number of nodes embedded per packet.

In case of the prime method, we use the General Number Field Sieve (GNFS) algorithm for prime factorization. The asymptotic running time for this algorithm for a b-bit number is $O(\exp((\frac{64b}{9})^{\frac{1}{3}} (\log b)^{\frac{2}{3}}))$.

In an N-node network, node IDs require at most $\lceil \log_2 N \rceil$ bits with the appropriate choice of s and m. Thus, multiplication of m node IDs requires at most $m\lceil \log_2 N \rceil$ bits which makes the complexity of prime factorization $O(\exp((\frac{64m\lceil \log_2 N \rceil}{9})^{\frac{1}{3}} (\log(m\lceil \log_2 N \rceil))^{\frac{2}{3}}))$.

We also need to perform DFS with backtracking. This entails s^m comparisons to find a path of m nodes (Section 5.1). We know that s can be approximated as $\frac{N}{\pi(N)} \sim \ln N$ (as discussed in the next section). Thus, the time required to decode provenance from a single packet becomes

$$O(\exp((\frac{64m\lceil \log_2 N \rceil}{9})^{\frac{1}{3}} (\log(m\lceil \log_2 N \rceil))^{\frac{2}{3}}) + \exp(N) + (\ln N)^m)$$

which is exponential in terms of N.

In case of the fingerprint method, we need to search the provenance graph and update the fingerprint using

concatenation while visiting a node on that graph. This is performed using DFS with backtracking which takes exponential time in terms of N in the worst case. However, nodes beyond $\lceil \frac{m+r-1}{r} \rceil$-hops away from the root can be pruned in this case. Thus the actual number of nodes visited by the algorithm is much lower in practice.

Regarding the construction of entire provenance in an N-node network, we can represent the provenance graph using an adjacency matrix. The total number of edges of the graph can be at most $O(N^2)$. Since edge insertion requires constant time, the worst case complexity for constructing the entire provenance is $O(N^2)$. We need $O(N^2)$ space to hold the provenance graph apart from the space required by the trust framework. Alternatively, we can use an adjacency list, which offers a different tradeoff.

6. Trust Framework and PPF

In order to assess the trustworthiness of a data item, traditional trust frameworks associate a trust score with each data item. Although the actual meaning of the trust score varies from application to application, this score can be used for comparison or ranking [9]. The data item having the highest trust score value can be labeled as the most trustworthy item with respect to other data items. Here we borrow the definition of trust of a data item given in [8]: "The trust of a data item i, denoted as $T(i)$, is the probability of i being true, as perceived by the receiver."

Provenance-based trust frameworks compute trust scores over a collection of data items based on their values and provenance information. These data items pertain to the same physical event and are received at the base station within a specified time window. The trust score of each data item, $T(i)$ is adjusted based on the *value similarity* and *provenance dissimilarity* of the data items. If similar data values have different provenance, this may increase the trustworthiness of data items. If the two data items have different provenance, they can be considered supportive to each other. In contrast, if they share similar provenance, this is not a clear indication of trustworthiness. Thus both *value similarity* and *provenance dissimilarity* of a data item contribute to its trust score.

PPF is agnostic of the way *provenance dissimilarity* is calculated under different trust frameworks. In order for us to explain the interface between PPF and trust frameworks, we consider the following definition of *provenance dissimilarity score* (ρ_d), a variant of the *path difference factor* proposed in [8]:

$$\rho_d = \frac{\sum_{t \in C, t \neq d} \rho(d,t)}{|C| - 1}$$

where C denotes a collection of data items and $\rho(d,t)$ indicates provenance difference between two data items d and t.

$$\rho(d,t) = \frac{max\{|G_d|, |G_t|\} - |I\{G_d, G_t\}|}{max\{|G_d|, |G_t|\}}$$

Here, G_d and G_t indicate provenance corresponding to the data items d and t, and $I\{G_d, G_t\}$ indicates set of nodes that are common to G_d and G_t.

6.1. Integration with Trust Framework

For a given collection of data items, trust frameworks require the complete provenance of each data item to calculate provenance dissimilarity score. Unfortunately, PPF only typically provides partial provenance of each data item. The partial provenance of a data item provided by PPF is either a part of the previously constructed complete forwarding path or a part of a new forwarding path that will be followed by the data items of the same source in near future. Thus after decoding partial provenance $\{ID_{i_1}, ID_{i_2}, \cdots ID_{i_m}\}$ of a data item (originated from source i), PPF returns its complete provenance to the trust framework as follows:

- Find paths from source i to ID_{i_1} and from ID_{i_m} to the base station on the current provenance graph G. If no such paths are found on G, continue searching over the graph G_{pre}.
- Concatenate the above paths with the decoded partial paths to form complete provenance for the received data item.

Provenance constructed in this manner provide near perfect accuracy in trust score calculation as we will observe in our simulation results (Section 6.1).

6.2. Trustworthiness vs. Transmission Overhead

Since sensor nodes are deployed near the phenomena to be sensed [31], data items with similar values are likely to share forwarding paths (or provenance). However, in an ideal (i.e., attack free) environment, trust scores evolve faster with data items having dissimilar provenance than with the items having shared provenance [9]. Dissimilar provenance of a collection of data items may help reduce the severity of an attack because

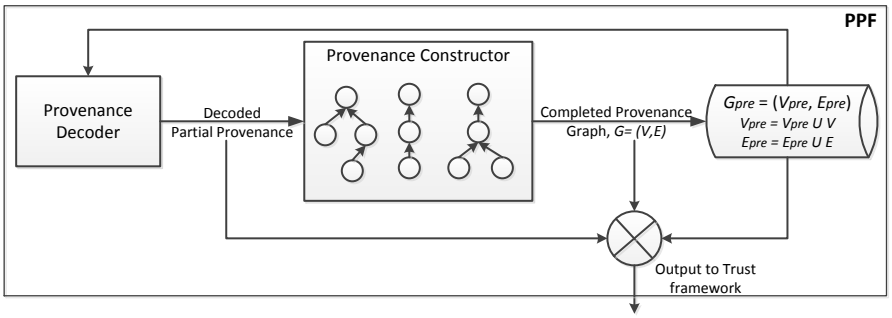


Figure 6: Construction of provenance and integration with trust framework

a single attacker cannot be present on multiple forwarding paths at the same time. For example, if we consider an adversary model where multiple attackers chain along a path and forward packets selectively [32], most of the data items sharing that forwarding path will be affected. In a threat model where multiple attackers collude to inject false data [33], similar provenance of data items helps locate the attack region faster. Thus, it is desirable to control the forwarding path of data items so that required similarity or dissimilarity among provenance can be ensured.

Forwarding paths are defined and maintained by the routing protocols on the sensor nodes. These paths or routes are usually built as part of a minimum cost tree rooted at the base station. In routing terminology, the cost for a node is the cost for its next hop plus the cost of its link to the next hop. Thus the cost of a route is the sum of the costs of its links towards the base station. Assuming expected transmissions (ETX) [34] as a cost metric, a forwarder node will route the data item through the next hop that requires the least estimated number of transmissions (to reach the base station) with respect to other neighbors. In this case, choosing a next hop other than the default one would result in a different forwarding path with a higher ETX value. This reveals an interesting trade-off between trustworthiness (or provenance dissimilarity) and transmission overhead: making provenance more dissimilar increases transmission overhead.

We propose a solution (termed as *controlled routing*) integrated with the default TinyOS routing protocol (CTP) to provide decision makers with a tunable parameter to control the extent of provenance dissimilarity and transmission overhead. Before forwarding a packet, the *next hop* function of the CTP routing engine is used to determine the best next hop (i.e., neighbor with the smallest ETX) out of the routing table of the forwarder. Our solution overrides this function with a parameterized *next hop* function which takes an argument called *ETX threshold (th_{etx})* and performs following steps:

- Before forwarding a packet, we examine the routing table of the forwarder node to determine a list of eligible next hops. We discard neighboring nodes that create self-loops and have estimated ETX beyond the smallest ETX by more than th_{etx}.
- We hash on the ID of the originator node of the packet to generate a value i between 1 and the number of eligible next hops. The i-th next hop in the eligible next hop list is returned as the output of the function.

Choosing next hop based on the hash value of the originator node ID keeps the forwarding path of the data items (generated from that node) consistent and makes the solution more tailored to the PPF encoding schemes. Further, by increasing the value of th_{etx}, we can increase the degree of dissimilarity among provenance of a collection of data items at the cost of higher transmission overhead, and vice versa.

7. Spread Factor

The prime method requires two parameters s and P that define the set of node IDs, $Q_{P,s}$.

7.1. Approximating the Spread Factor

For a given number of nodes, N, we want to determine the spread factor, s that minimizes the highest value of $Q_{P,s}$. This value of s depends on the prime gap.

Table 2: Prime gaps below the number n.

n	Prime Gap			
	Observed mean (μ)	Observed stdev	gap_{avg}	$\frac{\mu}{gap_{avg}}$
500	5.29	3.11	5.12	1.03
1000	5.96	3.55	5.82	1.02
2000	6.61	4.50	6.52	1.01
5000	7.48	5.30	7.44	1.01

Definition 4. *A prime gap is the difference between two successive prime numbers, p_k and $p_{(k+1)}$, where p_k is the kth prime number. Thus, a prime gap of length n is a run of $n-1$ consecutive composite numbers between two successive primes.*

We use the **Prime Number Theorem** to approximate the average length of prime gaps. The theorem gives an asymptotic form for the prime counting function $\pi(n)$, which counts the number of primes less than some integer n. According to this theorem (proved independently by Hadamard (1896) and de la Valle Poussin (1896)),

$$\begin{aligned}\pi(n) &\sim \sum_{k=0}^{\infty} \frac{k!n}{(\ln n)^{k+1}} \\ &\sim \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2n}{(\ln n)^3} + \cdots \end{aligned} \quad (3)$$

It has been shown that summation of the first three terms in equation 3 is a better estimate for $\pi(n)$ (Derbyshire 2004, pp. 116-117). Now, we can approximate the average length of prime gaps below n as

$$gap_{avg}(N) \approx \frac{n}{\pi(n)} \sim \frac{1}{\frac{1}{\ln n} + \frac{1}{(\ln n)^2} + \frac{2}{(\ln n)^3}}.$$

Table 2 shows the theoretical mean along with empirical mean and standard deviation of prime gaps for different values of n. The last column of this table gives the ratio between empirical and theoretical mean which justifies the approximation above. Assume that P_n denotes the nth prime number. By choosing a spread factor, s, that approximates to $gap_{avg}(N)$ for some N, we can obtain a set of numbers upper-bound by some prime number $P_{\pi(N)+1} \geq N$. We denote this set as $Q_{P_{\pi(N)+1}, gap_{avg}(N)}$. Due to the high variation in prime gaps with respect to $gap_{avg}(N)$ the cardinality of this set becomes less than N. Assume that using the same spread factor ($s = gap_{avg}(N)$), we find a set of numbers, $Q_{P_{\pi(N')},s}$ such that $|Q_{P_{\pi(N')},s}|$ is the smallest number greater than or equal to N. Similarly, by choosing some values larger than $gap_{avg}(N)$ for spread factor s, we can have a set of numbers $Q_{P_{\pi(N'')},s}$, where $|Q_{P_{\pi(N'')},s}|$ is the smallest number greater than or equal to N. Clearly, $P_{\pi(N')} > P_{\pi(N'')}$, which makes the latter set more favorable in terms of bit requirements (as observed in Table 1). We use the following optimistic choice:

$$s \geq gap_{avg}(N) \approx \frac{N}{\pi(N)} \approx \frac{1}{\frac{1}{\ln N} + \frac{1}{(\ln N)^2} + \frac{2}{(\ln N)^3}}. \quad (4)$$

7.2. Choice of Spread Factor

For a network of size N, we first calculate $gap_{avg}(N)$ and by setting $s = \lfloor gap_{avg}(N) \rfloor$, we pick a set $Q_{P_{\pi(N')},s}$ such that $|Q_{P_{\pi(N')},s}|$ is the smallest number greater than or equal to N. Considering the requirements for prime multiplication and summation of offset, we estimate the worst case bit requirements per packet as:

$$B^P_{worst}(s,m) = \log_2(\prod_{i=0}^{m-1}(P_{\pi(N')} - i)) + \log_2(m \times s),$$

where m is the number of node IDs embedded per packet.

Then we increase s by one and determine the corresponding set of node IDs. After calculating the worst case bit requirements for the new set, we compare the newly calculated value with the current one. If the newly calculated set outperforms the current one in terms of worst case bit requirements, we set the new set to be the current one. We continue until the newly calculated set requires more bits than the current one. Table 3 shows the comparison among the required number of bits for different values of s with varying numbers of nodes and number of per-packet node IDs. For a particular number of nodes and per-packet node IDs, the last column gives the best choice for s (s^*).

8. Bit Budget

A fixed budget of bits (B_{budget}) is available for embedding provenance of at most m nodes within the meta-data of a packet. We give the value of m for our three encoding methods in this section.

8.1. Bit Usage for Rank Method

In an N-node network, bit requirements for the rank method are $B^R(m) = m \times \log_2 N + \log_2 m$, where, the first term on the right hand side indicates the required number of bits to embed m *ranks*, and the second term accounts for the counter that tracks the number of embedded ranks. Thus, we choose the largest m such that $B^R(m) \leq B_{budget}$.

Table 3: Worst case bit requirements for varying number of nodes, per-packet node IDs, and choice of s.

m	N	gap_{avg}	$s = \lfloor gap_{avg} \rfloor$		$s = \lfloor gap_{avg} \rfloor + 1$		$s = \lfloor gap_{avg} \rfloor + 2$		$s = \lfloor gap_{avg} \rfloor + 3$		$s = \lfloor gap_{avg} \rfloor + 4$		s^*
			P_{max}	Bits	P_{max}	Bits	P_{max}	Bits	P_{max}	Bits	P_{max}	Bits	
3	500	5	613	32	587	33	563	33	547	33	521	33	5
3	1000	5	1307	36	1229	36	1163	36	1109	36	1063	36	5
3	2000	6	2683	40	2477	39	2377	39	2269	39	2213	39	10
3	5000	7	6779	44	6367	43	6079	43	5857	43	5669	44	10
4	500	5	613	43	587	42	563	42	547	43	521	43	7
4	1000	5	1307	47	1229	47	1163	46	1109	47	1063	47	7
4	2000	6	2683	51	2477	51	2377	51	2269	51	2213	51	7
4	5000	7	6779	56	6367	57	6079	57	5857	57	5669	56	7
5	500	5	613	52	587	51	563	52	547	52	521	52	6
5	1000	5	1307	57	1229	57	1163	57	1109	57	1063	57	6
5	2000	6	2683	62	2477	63	2377	63	2269	62	2213	62	6
5	5000	7	6779	70	6367	70	6079	69	5857	69	5669	69	11

Table 4: Average case bit requirements for varying numbers of nodes and per-packet node IDs with s^*.

N	$m = 3$		$m = 4$		$m = 5$	
	s^*	avg	s^*	avg	s^*	avg
500	5	27.82	7	36.16	6	44.35
1000	5	31.25	7	40.53	6	49.9
2000	10	34.5	7	45.14	6	55.81
5000	10	38.99	7	51.11	11	62.4

8.2. Bit Usage for Prime Method

In an N-node network, we can pessimistically pick a value of m such that $B^P_{worst}(m, s^*) \leq B_{budget}$. This does not guarantee the best usage of available bits since prime multiplication of m node IDs does not always need a fixed number of bits (as in the case of rank approach) and $B^P_{worst}(m, s^*)$ can hold more than m node IDs in many cases. Hence, we consider average case bit requirements before choosing an m for a particular bit budget. The average bit requirements per packet are

$$B^P_{avg}(s, m) = \log_2(m \times s) + \frac{m}{\pi(N')}\Big(\sum_{i=2}^{\log_2 \pi(N')-1} (i * [\pi(2^i) - \pi(2^{i-1})]) + \log_2 \pi(N') * [\pi(N') - \pi(2^{\log_2 \pi(N')-1})]\Big).$$

Table 4 shows the average number of bits calculated for different numbers of nodes and per-packet node IDs with their corresponding s^*. We choose an m^* such that $B^P_{avg}(s^*, m^* - 1) \leq B_{budget} \leq B^P_{avg}(s^*, m^*)$. For example, in a 1000-node network with 40 bits available for provenance embedding, we choose m^* to be 4 since $B^P_{avg}(5, 3) \leq 40 \leq B^P_{avg}(7, 4)$ (Table 4). This choice provides the opportunity to embed more node IDs per packet on the average.

Table 5: Choosing (r, m) for different bit budgets in a 5000-node network with $b = 13$ and $\epsilon = 5$.

r	$B_{budget} = 32$		$B_{budget} = 64$		$B_{budget} = 128$	
	m	x	m	x	m	x
1	2	2	9	9	25	25
2	n.a.	n.a.	7	4	21	11
3	n.a.	n.a.	n.a.	n.a.	16	6
4	n.a.	n.a.	n.a.	n.a.	9	3

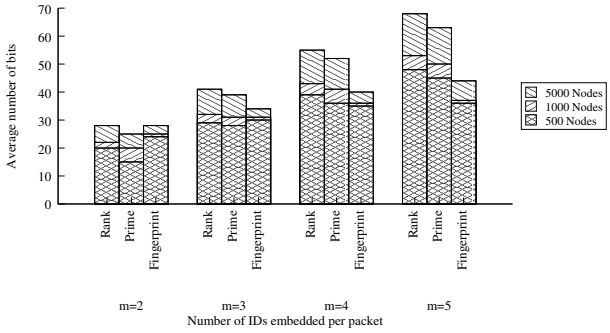


Figure 7: Average bit requirements for per packet provenance in networks of different sizes. (Stacked data along a single column is given in relative values that need to be added to get the absolute values of each textured bar.)

8.3. Bit Usage for Fingerprint Method

In an N-node network, the bit requirements for embedding m IDs per packet can be expressed as

$$B^F(r, m) = \begin{cases} b + \log_2 m + k, & r = 1, \\ (r-1)b + \log_2 m + rk, & r > 1, \end{cases} \quad (5)$$

where r is the number of partitions in the provenance buffer, k is the number of bits required for each fingerprint and b is the number of bits required to represent one node ID. We need to choose an m and r such that $B^F(r, m) \leq B_{budget}$. Here, $r = 1$ denotes the non-partitioned case, where the entire provenance buffer can be regarded as a single partition.

Table 6: TOSSIM channel and radio parameters for different topologies.

		Topology	
		Random	Grid
Channel Parameters	PATH_LOSS_EXPONENT	4.7	4.7
	SHADOWING_STANDARD_DEVIATION	3.2	3.2
	D0	1.0	1.0
	PL_D0	55.4	55.4
Radio Parameters	NOISE_FLOOR	-106.0	-105.0
	S11	0.9	3.7
	S12	-0.7	-3.3
	S21	-0.7	-3.3
	S22	1.2	6.0
	WHITE_GAUSSIAN_NOISE	4	4

Assuming a false positive probability of $2^{-\epsilon}, \epsilon \geq 0$, from equation 2,

$$k = 2(x-1)\log_2 f + \log_2 b + \log_2 x + \epsilon \quad (6)$$

where $m = (rx - r + 1)$.

First, we consider the case when $r = 1$, which leads to $x = m$. Then, combining (5) and (6) we have,

$$2(m-1)\log_2 f + 2\log_2 m \leq B_{budget} - b - \log_2 b - \epsilon. \quad (7)$$

Similarly, considering $r > 1$, we have

$$r(2(x-1)\log_2 f + \log_2 x + b + \log_2 b + \epsilon) + \log_2(rx - r + 1) \leq B_{budget} + b. \quad (8)$$

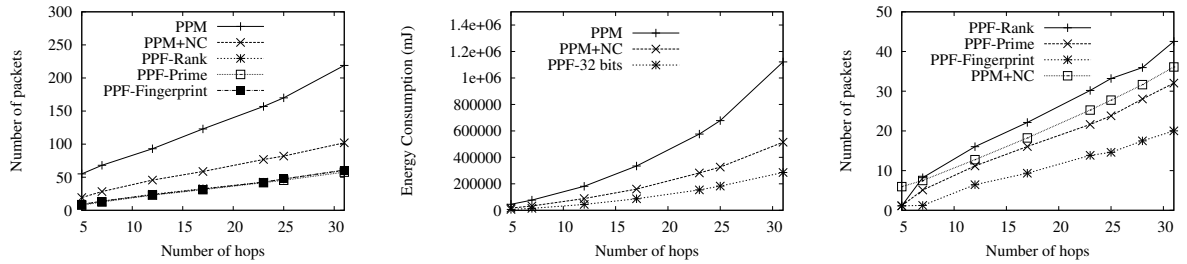
We determine the maximum value of m for different values of $r \geq 1$ using the above two equations. Decision makers are left to choose the appropriate pair of (r, m) based on the rate of link failure or changes, and the average fan-out of the network. In Table 5, we consider a 5000-node network with average fan-out $f = 4$, $b = 13$ bits, and $\epsilon = 5$ to show the possible choices for (r, m) where x indicates maximum number of node IDs contained in each partition. Note that fanout should be restricted so that excessive energy consumption at the junction node does not partition the network [35]. We also compare the theoretically calculated average bit requirements of the fingerprint method with the two other encoding schemes in Fig. 7. Clearly, the fingerprint method requires fewer bits than the other methods, as the value of N and m increase.

9. Simulations

We conduct simulations using TOSSIM [36] for networks with hop counts ranging from 5 to 31, and number of nodes ranging from 5 to 500. For energy analysis, we use POWERTOSSIMZ [37] which uses the *micaz* energy model. We do not consider energy consumption related to CPU computations since TOSSIM cannot capture CPU usage [37]. However, all nodes other than the base station only perform encoding operations which have low computational complexity and are unlikely to draw significant CPU power. A base station with no resource constraints can perform the decoding operations, e.g., prime factorization, for a moderate number of nodes in reasonable time. We compare the performance of the encoding schemes of **PPF** with the following two variants of probabilistic packet marking as they are the closest to our approach (though they were designed for wired IP networks):

- **PPM** [13, 14]: The most basic probabilistic packet marking scheme which embeds one node ID per packet and uses a *distance field* to track the position of the embedded node ID on the forwarding path from the source to the base station.
- **PPM with Network Coding** [24, 25]: Incorporates network coding with packet marking to embed a linear combination of a probabilistically chosen set of connected node IDs per packet. This scheme also uses a *distance field* to track the position of linearly combined node IDs on the forwarding path from the source to the base station.

Since **PPM** and **PPM+NC** are designed for IP traceback, we need to adapt them for wireless sensor networks. PPM will use 16 bits for embedding a single node ID (per TinyOS) and 8 bits for the *distance* field. PPM+NC computes a linear combination of node IDs over $F_{2^{16}}$ (since node IDs are 16 bits). The coefficients used to compute linear combinations are chosen over F_{2^2}. PPM+NC uses the rest of the bit budget to embed the coefficients and a counter that tracks number of participating nodes in the linear combination. We note that, our implementation of PPM+NC is a slightly improved version of the original one. In the original version of PPM+NC, a packet received at the base station



(a) Provenance construction with 32-bit budget (b) Aggregate energy consumption with 32-bit budget (c) Provenance construction with 64-bit budget

Figure 8: Comparison among different encoding schemes of PPF and PPM variants in a random topology.

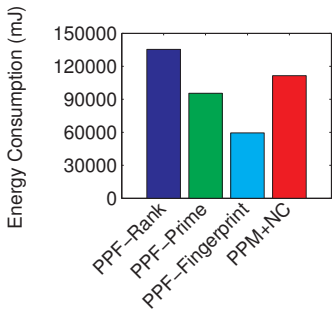


Figure 9: Energy consumption for 25-hop network with 64-bit budget.

does not contain any linear combination of node IDs if no intermediate nodes mark the packet. We modified this so that a packet received at the base station contains a linear combination of the first few node IDs even if no intermediate nodes mark the packet.

9.1. Performance Metrics

The following performance metrics are considered to evaluate our proposed schemes:

- *Number of transmitted packets:* The total number of packets transmitted by all nodes on a path from a particular source to the base station before complete provenance is constructed for that path.
- *Aggregate energy consumption:* The total energy consumed (in *mJ*) by all nodes that participated in encoding provenance of a path from a particular source to the base station.
- *Decoding error:* The percentage of node IDs that cannot be decoded due to link changes or false positive rates (if any).

9.2. Simulation Setup

All experiments are performed using a transmission rate of 250 kbps, the default transmission rate of the *micaz* mote, where every data-generating sensor sends data towards the base station every 2 s. The probability for embedding a node ID is $p = \frac{1}{25}$. Before starting data transmission, the following initialization steps are performed:

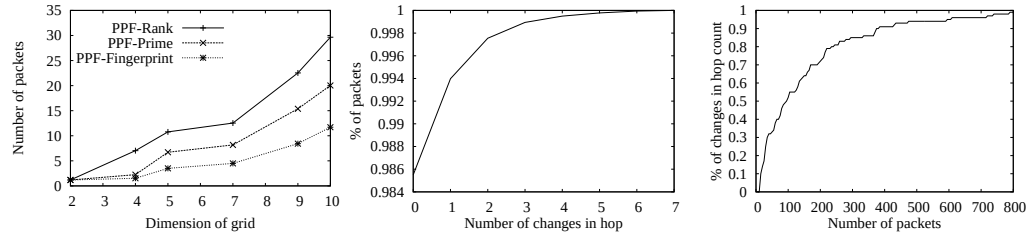
- Every node is assigned a node ID from the appropriate set $Q_{P,s*}$ (discussed in section 7).
- Every node computes and stores the lookup table necessary to compute Rabin fingerprints.
- The sender node sends 500 dummy packets towards the base station to ensure convergence of the routing protocol CTP. These packets are discarded at the base station and thus not used in the provenance construction process.

All results are *averaged over 1000 runs*, and we find the standard deviation to be extremely small. Unless otherwise stated, we use the above default values in our simulations.

9.3. Random Topology

We start with a randomly deployed 500-node network where the position of the source node is varied to simulate paths of different hops (from 5 to 31). The channel and radio parameters for this topology are listed in Table 6. The nodes have an average noise floor of -106 dBm, a standard deviation of 4.0 dB for the white Gaussian noise, and a low level of asymmetry. The network has very low rate of link changes. We place the same constraint on usable bits (32 bits) for provenance embedding per packet on all schemes.

Fig. 8(a) shows the number of packets required to construct provenance for increasing numbers of hops from a single source to the base station. The results



(a) Provenance construction with 64-bit budget

(b) CDF of changes in hop count for a 10 × 10 grid network.

(c) CDF of number of packets between two group of changes in hop count for a 10 × 10 grid network.

Figure 10: Convergence of provenance construction and distribution of topological changes in highly asymmetric grid networks.

reveal that all three schemes of PPF have identical performance in this case, since they can embed only 3 node IDs (on average) per packet using 32 bits. However, they require at least 33% fewer packets than both PPM variants. The original PPM scheme requires a large number of packets since it embeds a single node ID per packet. In contrast, PPM+NC uses 16 bits (out of 32) for the linear combination, 8 bits for the distance field, 6 bits for storing coefficients, and the remaining 2 bits to count the coefficients used. Thus it embeds a linear combination of 3 node IDs in a packet. However, in order to construct a forwarding path of length d hops, PPM+NC converges upon reception of d unique linear combinations of node IDs, whereas PPF requires d different node IDs.

Fig. 8(b) compares the aggregate energy consumption for the two PPM variants and PPF-Prime (the PPF method that requires the lowest number of packets for a 32-bit budget). PPF with a 32-bit budget consumes at least 30% less energy than the PPM variants.

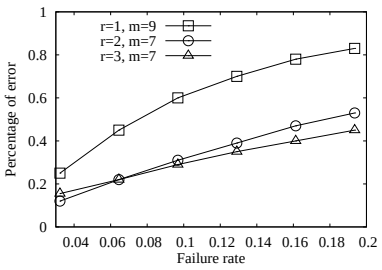
We perform the same experiment in a 5000-node network with a 64-bit budget. We skip PPM since it embeds only one node ID per packet and requires the same number of packets. Since TOSSIM does not scale to 5000 nodes, we randomly assign node IDs from a set of 5000 numbers and take the average over experimental results of several TOSSIM runs. Fig. 8(c) shows that the prime method requires fewer packets than the rank method, while the fingerprint method outperforms both in this case. The reason is that with a 64-bit budget, the fingerprint method ($r = 2$) embeds 7 node IDs per packet with a low false positive rate (< 0.001), whereas the prime and rank methods embed 5 and 4 node IDs on average, respectively. Though PPM+NC can embed a linear combination of up to 17 node IDs (16 bits for linear combination field, 8 bits for distance field, 34 bits for the coefficients and 5 bits for the counter field), it still requires more packets than the efficient PPF schemes. Specifically, PPF-Fingerprint requires at least 45% fewer packets than PPM+NC.

Fig. 9 compares the aggregate energy consumption for PPM+NC and PPF methods with a 64-bit budget in a 25-hop network. We find that PPM+NC consumes less energy with respect to PPF-Rank and the percentage of energy gain is only 17 in this case. The best PPF scheme (PPF-Fingerprint in this case) reduces energy consumption by 46% with respect to PPM+NC. Further, PPF with a 64-bit budget reduces energy consumption by more than 60%, compared to its 32-bit counterpart.

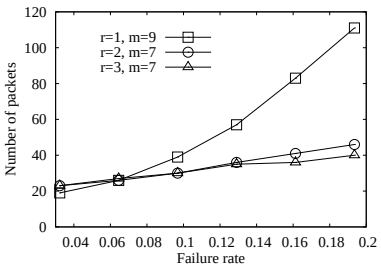
9.4. Grid Network Topology

We use highly asymmetric grid topologies to demonstrate the effect of topological changes on our proposed schemes. The radio and channel parameters used for these topologies are shown in Table 6. The grid dimensions are varied from 2×2 to 10×10, while node density is kept constant with nodes spaced 2.0 meters apart. The source node is positioned at coordinate (0,0) and the base station is located at the upper rightmost corner of the grid. The average hop count is observed to range from 2 to 19 for grid dimensions of 2 to 10. The bit budget considered here is 64 bits.

We combine PPF with the *topological changes handling* scheme discussed in Section 4.4, and compare the number of packets required by PPF-Rank, PPF-Fingerprint, and PPF-Prime. We do not consider PPM variants in this simulation since they do not handle topological changes. Fig 10(a) depicts the simulation results. PPF-Fingerprint outperforms other methods as the grid dimensions (and hence the hop count) increase. It is interesting to note that PPF methods combined with topological change handling perform similar to the basic PPF methods (without the topological change handling) in a network with negligible link changes.



(a) Decoding error



(b) Number of packets required

Figure 11: Provenance construction with varying rate of link changes.

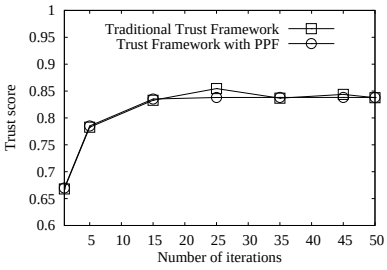
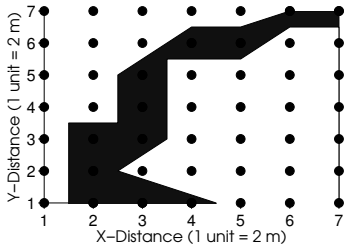
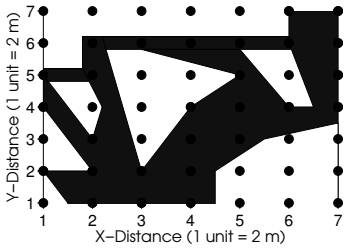


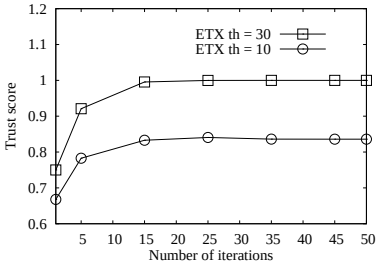
Figure 12: Trust scores of data items



(a) Nodes in provenance (th_{etx} = 10)



(b) Nodes in provenance (th_{etx} = 30)



(c) Evolution of trust scores

Figure 13: Nodes in provenance and effect on trust score evolution

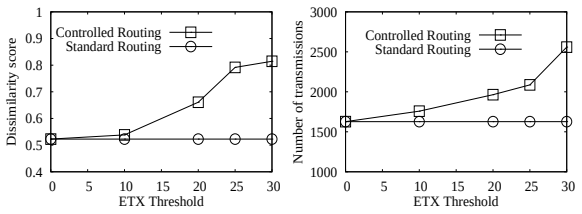
To understand the simulation results, we consider a 10 $\times$ 10 grid network. The source node transmits 20,000 packets towards the base station. Fig. 10(b) shows the distribution of the number of changes in hop count for successive packets received at the base station (after discarding out-of-order packets). About 98% of the packets that experienced topological changes have exactly a 1-hop difference with respect to the path followed by the preceding packet. Fig. 10(c) shows the distribution of the number of packets received at the base station between two consecutive groups of topological changes. The number of packets between two topological changes significantly varies. In the 80% case, at least 50 packets are being received at the base station between two changes. Thus, we find that once there is a topological change, it affects a small number of hops on the path from the source to the base station. After that, the path remains stable for large number of packets. This is why PPF with topological change handling incurs negligible overhead without affecting convergence rate.

9.5. Varying Link Change Rates

We have observed that incorporating the topological change handling scheme into PPF helps track changing paths. In such scenarios, the fingerprint method provides additional resistance to decoding error over other encoding methods, since it transmits a large provenance subgraph with smaller partitions as discussed in Section 4.3.2.

To observe the effect of partitioning fingerprints, we study the decoding error of the fingerprint method with respect to the rate of link changes. Here, *decoding error* denotes the percentage of node IDs that cannot be decoded due to link changes or false positive rates, where *rate of link changes* indicates the average number of link changes per unit time. We artificially introduce link failures and associated path changes that are randomly distributed over a time window of 200 s along a 30-hop path.

Fig. 11(a) shows that as link changes increase, the fingerprint method with two partitions ($r = 2$) has a decoding error lower than the non-partitioned case because of its low sensitivity to topological changes. However, the fingerprint method with $r = 3$ suffers from a false positive rate of about 0.16 with a 64-bit budget in this particular experiment, and performs worse than the $r = 2$ case in the presence of low rate of link changes (when decoding error due to link changes is small). With a high rate of link changes, the case of $r = 3$ shows a small improvement over the $r = 2$ case,



(a) Provenance dissimilarity score (b) Transmission overhead

Figure 14: Trade-off between provenance dissimilarity and transmission overhead

but a relatively higher false positive rate in a dense network will nullifies that improvement (as indicated in Table 5). Fig. 11(b) shows the effect of the decoding error in constructing provenance. The fingerprint method with $r = 2$ converges with a fewer number of packets even in the presence of a high rate of link changes. The false positive rate is negligible.

9.6. Trust Framework and Provenance Similarity

We integrate PPF with a provenance-based trust framework [9] to iteratively compute trust scores. To evaluate the performance of the trust framework integrated with PPF, we use a 7 $\times$ 7 grid with the same channel and radio parameters as the grid topology (see Table 6). Five nodes (2, 3, 4, 5, and 6) generate data items every 1 s and the rest of the nodes act as forwarders. Each of the data generator nodes send 2000 data items. The top right most node (49) in the grid works as the base station which computes trust score over each collection of data items that have same sequence number but are generated from different originators. Fig. 12 shows that the trust score calculated using PPF evolves correctly as soon as the entire provenance is constructed at the base station. PPF accuracy in trust score calculation is similar to the traditional approach that includes every node ID on the forwarding path in the provenance.

We use the same 7 $\times$ 7 grid network with varying ETX threshold (th_{etx}) to investigate the effectiveness of our *controlled routing* scheme proposed in Section 6.2. We keep the data values reported from the originator nodes fixed over different packets to solely focus on the role of provenance similarity.

Fig. 13(a) and Fig. 13(b) show the provenance of the data items generated from node 2, 3, and 4 for $th_{etx} = 10$ and $th_{etx} = 30$ respectively. It is seen that data items have shared provenance with lower th_{etx} and different provenance with higher th_{etx}. Fig. 13(c) shows the resulting effect of provenance dissimilarity on the trustworthiness of data items: higher dissimilarity in provenance increase the trustworthiness of data items. Finally, Fig. 14 shows the trade-off between provenance dissimilarity and transmission overhead. As the ETX threshold, th_{etx}, increases, dissimilarity among provenance, ρ_d, increases at the expense of increased transmission overhead. This allows decision makers to select an appropriate th_{etx} value based on the security requirements and network conditions of the system.

10. Testbed Experiments

We ported the implementation of PPF to the TelosB platform. Our motes have an 8 MHz TI MSP430 microcontroller, 2.4 GHz radio, 10 kB RAM, and 1 MB flash for data logging. We also ported the implementation of PPM and PPM with Network Coding to this platform. We consider the same performance metrics as in 9.1 to compare with PPF.

10.1. Experimental Setup

We placed battery-powered TelosB motes in an indoor environment. When necessary, we controlled the transmission power of the motes to ensure multihop communication in the network. We assign node IDs from the set $Q_{P,s*}$. All nodes are started and stopped at the same time. The source node sends out packets every 500 ms. The probability of embedding a node ID is set to $\frac{1}{25}$. Before actual data transmission starts, every node computes and stores the lookup table necessary to compute Rabin fingerprints and the source node sends 500 dummy packets to ensure convergence of the routing protocol CTP. All results are *averaged over 500 runs*.

10.2. Multihop Linear Topology

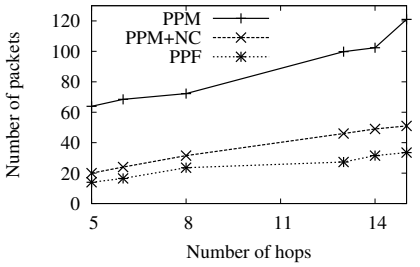
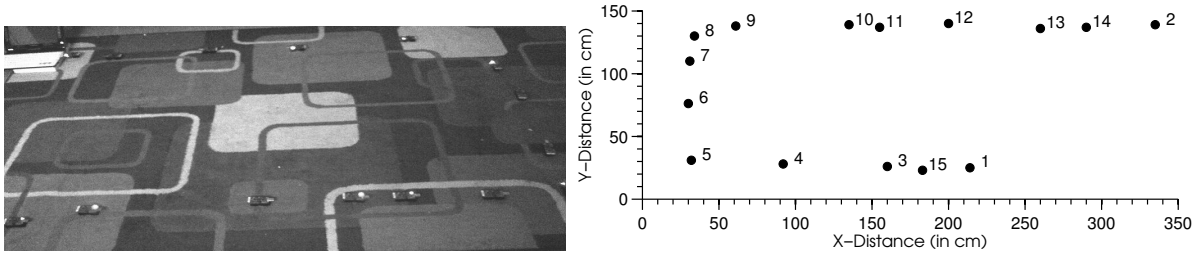


Figure 16: Provenance construction in a linear testbed

We construct a 3.5 m $\times$ 1.5 m topology consisting of 15 TelosB sensors deployed linearly in an apartment room. We used the lowest transmission power level to



(a) Partial view of testbed

(b) Coordinates of nodes

Figure 15: Placement of nodes in an apartment room in a multihop linear topology.

ensure multihop communication. Fig. 15 shows the coordinates of the nodes in the testbed, where nodes are labeled 1 to 15. We used node 1 as base station and connected it to a laptop through a USB cable. A Java application running on the laptop received the packets and performed the real-time decoding process. We conducted different sets of experiments by varying hop counts from 5 to 15. The bit budget considered here is 32 bits.

We compare PPF-prime (the best scheme for the 32-bit budget as seen from our simulation results) with PPM and PPM with network coding. Fig. 16 plots the number of packets required to construct provenance. As the hop count increases, PPF outperforms both PPM variants and reduces the number of required packets by more than 33%. This result validates the gain that we achieved in TOSSIM simulations.

10.3. Multihop Random Topology

We construct a 9.5 $m \times 1.6\ m$ topology consisting of 20 TelosB sensors deployed randomly in a Purdue University classroom. We used transmission power level 1 to create relatively weak wireless links. Fig. 17 shows the coordinates of the nodes in the testbed, where nodes are labeled 1 to 20. Node 2 is selected as the source and node 1 (connected to a laptop) is chosen as the base station. The hop count from the source to the base station was observed to be between 9 and 11. Since PPF schemes with a 32-bit budget exhibit almost identical performance for such a small hop count, we consider a 64-bit budget for this experiment.

We compare PPF-Rank, PPF-Prime and PPF-Fingerprint in the presence of topological changes. Fig. 18 depicts the average number of packets required to construct provenance for all three schemes. It is observed that PPF-Prime and PPF-Fingerprint perform better than PPF-Rank. Their performance is similar to basic PPF schemes in a network of 9-11 hops with negligible rate of link changes.

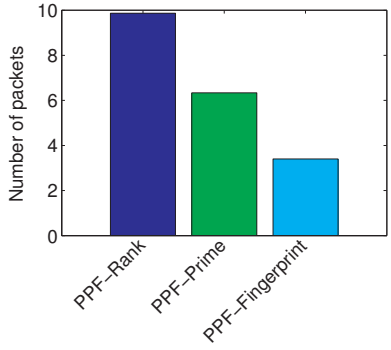
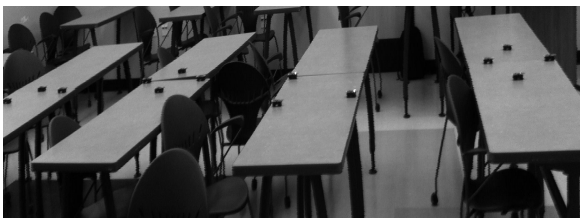


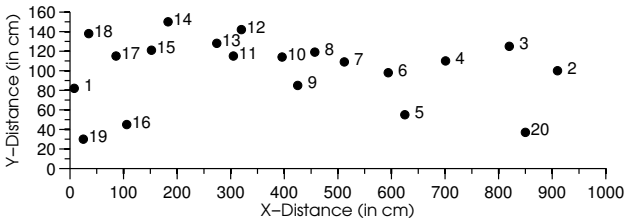
Figure 18: Number of packets required to construct provenance in the random topology testbed experiment.

11. Conclusions

We have presented an energy-efficient provenance transmission and construction approach for large-scale multi-hop wireless sensor networks, based on the idea of probabilistic incorporation of node identities. We adapt the probabilistic packet marking (PPM) approach for IP traceback, and propose three provenance encoding methods with a space constraint on the size of provenance data in each packet. We analyze the suitability of the methods based on the network size and bit budget via mathematical approximations and numerical methods. In contrast to PPM, our proposed approach requires fewer packets to construct network-wide provenance, and significantly reduces the aggregate energy consumption of the network, as demonstrated via both simulations and testbed experiments. We also incorporate a simple but robust scheme into PPF to handle topological changes. PPM variants do not consider such



(a) Partial view of testbed



(b) Coordinates of nodes

Figure 17: Placement of nodes in a Purdue University classroom in a multihop random topology.

changes. We demonstrate the effectiveness of PPF in highly dynamic and asymmetric networks using simulation and testbed experiments. PPF integration with a provenance-based trust framework reveals no degradation in accuracy of trust scores. We also explore the trade-off between trustworthiness or provenance dissimilarity of data items and transmission overhead. In this regard, we propose a solution to provide decision makers with a tunable parameter to control the extent of provenance dissimilarity and transmission overhead. In our future work, we will study how well a complete trust framework can detect and react to different attacks and failure scenarios.

References

[1] D. Butler, 2020 computing: Everything, everywhere, Nature 440 (2006) 402–405.

[2] M. Zuniga, B. Krishnamachari, Integrating future large-scale wireless sensor networks with the Internet, Tech. rep., USC Computer Science, cS 03-792 (2003).

[3] K. Fehrenbacher, A global sensor network launches to fight climate change, http://www.reuters.com/article/idUS359029730820110112 (January 2011).

[4] Shell to use CeNSE for clearer picture of oil and gas reservoirs, http://www.hpl.hp.com/news/2009/oct-dec/cense.html (2009).

[5] A. Doboli, et al., Cities of the future: Employing wireless sensor networks for efficient decision making in complex environments, Tech. rep., SUNYSB, cEAS Technical Report Nr 831 (April 2008).

[6] Sensor Andrew at Pennsylvania Smart Infrastructure Incubator, http://www.ices.cmu.edu/psii/sensor-andrew.html.

[7] J. Ledlie, C. Ng, D. A. Holland, K. kumar Muniswamy-reddy, U. Braun, M. Seltzer, Provenance-aware sensor data storage, in: Proc. of Workshop on Networking Meets Databases, 2005.

[8] X. Wang, K. Govindan, P. Mohapatra, Provenance based information trustworthiness evaluation in multi-hop networks, in: Proc. of IEEE GLOBECOM, 2010.

[9] H.-S. Lim, Y.-S. Moon, E. Bertino, Provenance-based trustworthiness assessment in sensor networks, in: Proc. of International Workshop on Data Management for Sensor Networks, 2010.

[10] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, M. Turon, Health monitoring of civil infrastructures using wireless sensor networks, in: Proc. of IPSN, 2007, pp. 254–263.

[11] A. Arora et al., ExScal: Elements of an extreme scale wireless sensor network, in: Proc. of IEEE International Conference on Real-Time Computing Systems and Applications, 2005.

[12] K. Iwanicki, M. van Steen, On hierarchical routing in wireless sensor networks, in: Proc. of IPSN, 2009.

[13] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for IP traceback, in: Proc. of ACM SIGCOMM, 2000.

[14] D. X. Song, A. Perrig, Advanced and authenticated marking schemes for IP traceback, in: Proc. of IEEE INFOCOM, 2001.

[15] V. Thing, H. Lee, IP traceback for wireless ad-hoc networks, in: Proc. of IEEE Vehicular Technology Conference, 2004.

[16] S. M. I. Alam, S. Fahmy, An energy-efficient approach for provenance transmission in wireless sensor networks, in: Proc. of IEEE SECON, 2012.

[17] S. M. I. Alam, S. Fahmy, Energy-efficient provenance transmission in large-scale wireless sensor networks, in: Proc. of IEEE International Workshop on D-SPAN, 2011.

[18] A. Belenky, N. Ansari, On IP traceback, Communications Magazine, IEEE 41 (7) (2003) 142 – 153.

[19] H. Burch, Tracing anonymous packets to their approximate source, in: Proc. of USENIX conference on system administration, 2000.

[20] A. C. Snoeren, Hash-based IP traceback, in: Proc. of ACM SIGCOMM, 2001.

[21] ICMP traceback messages, http://tools.ietf.org/html/draft-ietf-itrace-04.

[22] Y. an Huang, W. Lee, Hotspot-based traceback for mobile ad hoc networks, in: Proc. of ACM Workshop on Wireless Security, 2005.

[23] H. Hsu, S. Zhu, A. Hurson, A hotspot-based protocol for attack traceback in mobile ad hoc networks, in: Proc. of ACM Symposium on Information, Computer and Communication Security, 2010.

[24] P. Sattari, M. Gjoka, A. Markopoulou, A network coding approach to IP traceback, in: Proc. of IEEE International Symposium on Network Coding, 2010.

[25] D. Dean, M. Franklin, A. Stubblefield, An algebraic approach to IP traceback, ACM Trans. Inf. Syst. Secur. 5 (2002) 119–137.

[26] B.-C. Cheng, H. Chen, Y.-J. Li, R.-Y. Tseng, A packet marking with fair probability distribution function for minimizing the convergence time in wireless sensor networks, Computer Communication 31 (2008) 4352–4359.

[27] M. Keller, J. Beutel, L. Thiele, How was your journey? uncovering routing dynamics in deployed sensor networks with multihop network tomography, in: Proc. of ACM Sensys, 2012.

[28] S. Sultana, G. Ghinita, E. Bertino, M. Shehab, A lightweight se-

cure provenance scheme for wireless sensor networks, in: Proc. of IEEE Parallel and Distributed Systems (ICPADS), 2012.
[29] M. O. Rabin, Fingerprinting by random polynomials, Tech. Rep. TR-CSE-03-01, Center for Research in Computing Technology, Harvard University (1981).
[30] A. Z. Broder, Some applications of rabin's fingerprinting method, Sequences II: Methods in Communications, Security, and Computer Science 5 (4) (1993) 143–152.
[31] D. Ganesan, R. Govindan, S. Shenker, D. Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, SIGMOBILE Mob. Comput. Commun. Rev. 5 (4) (2001) 11–25.
[32] B. Yu, B. Xiao, Detecting selective forwarding attacks in wireless sensor networks, in: Proc. of Parallel and Distributed Processing Symposium, 2006.
[33] S. Zhu, S. Setia, S. Jajodia, P. Ning, An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks, in: Proc. of IEEE Symposium on Security and Privacy, 2004.
[34] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, P. Levis, Collection tree protocol, in: Proc. of ACM Sensys, 2009.
[35] H. S. Kim, T. F. Abdelzaher, W. H. Kwon, Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks, in: Proc. of ACM Sensys, 2003, pp. 193–204.
[36] P. Levis, N. Lee, M. Welsh, D. Culler, TOSSIM: Accurate and scalable simulation of entire TinyOS applications, in: Proc. of ACM Sensys, 2003.
[37] E. Perla, A. Catháin, S. Carbajo, M. Huggard, C. McGoldrick, PowerTOSSIM z: realistic energy modelling for wireless sensor network environments, in: Proc. of ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks, 2008.

A. Implementation of Fingerprint Calculation

Assume that M denotes the irreducible polynomial of degree z required to calculate the fingerprint. We show how the fingerprint of the string $[b_1 \cdots b_t]$ can be calculated by extending it one bit at a time. For an arbitrary length l, we can write,

$$\begin{aligned} RF([b_1 \cdots b_l]) &= (b_1\alpha^{l-1} + b_2\alpha^{l-2} + \cdots b_l) \ \ mod \ \ M \\ &= r_1\alpha^{z-1} + r_2\alpha^{z-2} + \cdots r_z \end{aligned}$$

Now if we extend $[b_1 \cdots b_l]$ by one bit b_{l+1}, we have,

$$\begin{aligned} &RF([b_1 \cdots b_l]||b_{l+1}) = RF([b_1 \cdots b_{l+1}]) \\ &= (RF(b_1 \cdots b_l)\alpha + b_{l+1}) \ \ mod \ \ M \\ &= (RF(b_1 \cdots b_l)\alpha + b_{l+1}) \ \ mod \ \ M \\ &= ((r_1\alpha^{z-1} + \cdots r_z)\alpha + b_{l+1}) \ \ mod \ \ M \\ &= r_2\alpha^{z-1} + \cdots r_z\alpha + b_{l+1} + (r_1\alpha^z) \ \ mod \ \ M \end{aligned}$$

Observe that $\alpha^k \ \ mod \ \ M = \alpha^k - M = M - \alpha^k$. So $\alpha^k \ \ mod \ \ M$ is equivalent to M with the leading coefficient removed. Computing the fingerprint of A extended by b_{l+1} consists of one shift left operation with b_{l+1} as the input bit and r_1 as the output bit, and then, conditioned upon $r_1 = 1$, a bit-wise XOR operation, the second operand being M with the leading coefficient removed. In this way, we can extend the bit string up to length t to calculate the desired fingerprint. Since each bit extension requires only a constant number of shift and XOR operations, the time complexity of calculating the fingerprint is linear in number of bits of the input string.

We further show that time complexity of fingerprint calculation can be improved by extending a byte at a time and using a lookup table. If we extend $[b_1 \cdots b_l]$ by one byte $[b_{l+1}, b_{l+2}, \cdots b_{l+7}, b_{l+8}]$, we have

$$\begin{aligned} &RF([b_1 \cdots b_l]||[b_{l+1}, b_{l+2}, \cdots b_{l+7}, b_{l+8}]) \\ &= RF([b_1 \cdots b_{l+8}]) \\ &= (RF(b_1 \cdots b_l)\alpha^8 + b_{l+1}\alpha^7 + \cdots + b_{l+8}) \ \ mod \ \ M \\ &= ((r_1\alpha^{z-1} + r_2\alpha^{z-2} + \cdots r_z)\alpha^8 + \\ &\ b_{l+1}\alpha^7 + \cdots b_{l+7}\alpha + b_{l+8}) \ \ mod \ \ M \\ &= (r_9\alpha^{z-1} + r_{10}\alpha^{z-2} + \cdots r_z\alpha^8 + b_{l+1}\alpha^7 + \cdots b_{l+8}) \\ &+ (r_1\alpha^{z+7} + \cdots r_7\alpha^{z+1} + r_8\alpha^z) \ \ mod \ \ M \end{aligned}$$

Observe that the first part of the above equation can be determined by shifting the previous fingerprint, $RF([b_1 \cdots b_l])$ to left by 8 bits with the leading 8 bits removed and then XORing the output with the input byte: $([r_1 \cdots r_z] << 8) \oplus ([r_1 \cdots r_8] << z) \oplus ([b_{l+1} \cdots b_{l+8}])$. The second part is determined by a modulo operation where the dividend is the leading 8 bits of the fingerprint, $RF([b_1 \cdots b_l])$ shifted to left by z bits and the divisor is M: $([r_1 \cdots r_8] << z) \ \ mod \ \ M$. By combining the two parts, fingerprint can be determined as,

$$\begin{aligned} &RF([b_1 \cdots b_{l+8}]) \\ &= ([r_1 \cdots r_z] << 8) \oplus ([r_1 \cdots r_8] << z) \oplus \\ &([b_{l+1} \cdots b_{l+8}]) \oplus (([r_1 \cdots r_8] << z) \ \ mod \ \ M) \\ &= \underbrace{([r_1 \cdots r_z] << 8) \oplus ([b_{l+1} \cdots b_{l+8}])}_{\text{part 1}} \oplus \\ &\underbrace{(([r_1 \cdots r_8] << z) \oplus (([r_1 \cdots r_8] << z) \ \ mod \ \ M))}_{\text{part 2}}. \end{aligned}$$

Since M and z are known beforehand, part 2 of the above equation can be looked up from a precomputed table (of size $<$ 4 KB) which contains the values of the expression $(([r_1 \cdots r_8] << z) \oplus (([r_1 \cdots r_8] << z) \ \ mod \ \ M))$ for all possible values of the bits $[r_1 \cdots r_8]$. Thus the fingerprint of a bit string extended by one byte can be calculated using only 1 shift and 2 XOR operations.