Connectivity Monitoring in Wireless Sensor Networks

Mingze Zhang, Mun Choon Chan and A.L. Ananda
School of Computing, National University of Singapore
{zhangmi3,chanmc,ananda}@comp.nus.edu.sg

Abstract

Connectivity monitoring is useful in practical deployment of wireless sensor network. In order to understand the behavior and performance bottleneck, knowledge of the network connectivity is crucial. In this paper, we propose a flexible and efficient connectivity monitoring algorithm (H^2CM) that has three components and operates in a divide and conquer manner. The components include hop vector distance based filtering, Bloom filters and signature hashing and are designed to work with different combinations of network and neighbor set sizes. In simulation, communication cost reduction of H^2CM compare to maximal compression of neighborhood information varies from 65% to 85% for large networks ($>$ 1000 nodes) and from 40% to 70% for a medium size network (a few hundred nodes). We have also implemented the algorithm in TinyOS and evaluated its performance on a testbed with 34 motes.

1 Introduction

In traditional network management, network topology is one of the key parameters that needs to be known in order to perform operations like performance management, fault detection and isolation, and capacity planning. In the Internet, despite the fact that the control and ownership are highly distributed, researchers have also attempted to gain understanding of the Internet topology. Examples include [9, 12, 19, 20].

In wireless sensor network, in addition to tackling traditional network issues such as fault monitoring/debugging and root-cause analysis [17], connectivity information also helps in ways that are unique to how sensor network operates. For example, it is observed in [7] that connectivity statistics can be used to compute mean topological density, study the impact of link asymmetry, evaluate geographical routing algorithms, and assess behaviors of algorithms that depend on spatial correlation.

However, obtaining connectivity information efficiently in wireless sensor network is generally a hard problem. First, connectivity is highly unpredictable due to low power transmission, limited energy resource, adhoc deployment and factors such as obstacles and movement in the environment. Second, as connectivity of wireless links can vary over time, nodes need to send information to the central controller periodically or on-demand, via multiple hops. The cost can be significant due to the limited energy and bandwidth resources available on the sensor nodes.

In this paper, we propose H^2CM, a flexible and efficient algorithm to obtain connectivity information of the nodes located in the area of interest (monitored nodes). H^2CM is based on a *divide-and-conquer* approach, in which several techniques are combined to deal with various network and neighbor set sizes. These techniques are (1) hop count filtering, (2) Bloom filter and (3) use of a single hash value as checksum. By varying the amount of information exchanged, H^2CM is able to provide different level of connectivity information accuracy.

H^2CM is flexible in that each node can be individually configured to provide the desired accuracy. As a result, nodes deem more important can be be configured to provide more accurate connectivity information. H^2CM is efficient in reducing communication cost, even when complete connectivity information of the nodes is required. Simulation results show that for a large network ($>$ 1000 nodes) with node densities varying from 5 to 30, over 99.99% of all links are discovered and the communication savings vary from 65% to 85% compare to maximal compression of neighborhood information. For a medium size network (a few hundred nodes), about 40% to 70% savings can be achieved. We have implemented H^2CM in a sensor testbed with 34 MICA2 nodes. The algorithm is implemented using less than 80 lines of TinyOS code and adds about 600 bytes of ROM image size (code size). Even with such a small network, the total communication cost is comparable to the cost of using maximal compression.

The rest of the paper is organized as follows. Related work is presented in Section 2 and the system model in Section 3. Existing techniques in reducing communication cost of connectivity monitoring are introduced in Section 4. We present the proposed algorithm in Section 5. Evaluation results are shown in Section 6 and conclusion in Section 7.

2 Related Work

In [11], the authors propose TopDisc algorithm for sensor networks. The algorithm only let those nodes in minimum dominating set (MDS) send their neighborhood information to the central controller, thereby reducing the communication overheads. In [10], the authors propose a multi-resolution topology retrieval protocol. The algorithm makes use of minimal virtual dominating set (MVDS) to define the distinguished nodes that will respond to the topology probes. By adjusting the virtual radius of MVDS, multi-resolution can be achieved. In the above mentioned topology discovery algorithms, the total communication cost are tradeoff for the accuracy of network topology information. In [8], the authors propose a mesh based topology retrieval algorithm with slow moving nodes in wireless ad hoc networks. However, the main focus here is reliable topology information collection, and the authors do not address the communication cost.

The topology discovery algorithms mentioned above try to select a small percentage of the nodes who will respond to the topology discovery probes, and each of these nodes may only send partial neighborhood information to the central controller. Therefore, the total communication cost can be tradeoff for the accuracy of network topology information. In [21], the problem of complete topology discovery is discussed, the work is based on the assumption that location information is available, the neighborhood pattern is also assumed to have strong correlation with the distance between a pair of the sensor nodes.

In this paper, our approach to cost reduction is through reducing the amount of data sent by each node where the accuracy can be pre-configured. A large savings can be achieved even when complete connectivity information is required. Note that the proposed algorithm allows the choice of monitoring any subset of nodes. Therefore, one can still choose to use MDS-based approach [11] together with our proposal.

Our proposed solution makes use of Bloom filter [3, 4] as a tool for neighborhood information retrieval with much less communication cost at each node with loss of only a very small amount of information.

Bloom filter is a simple and space-efficient probabilistic data structure that belongs to the class of approximate membership testers as given in [6]. It is used to represent a set with much less space requirement than directly representing the entire whole set. Membership testing over Bloom filters is simple and fast though a small probability of false positives may present. Recently Bloom filters have been widely applied in networking areas such as distributed caching [13, 18], p2p and overlay networks [5], measurement [2], and many others.

3 System Model

There are T nodes in the network and each node has a unique global ID. This ID can be its own MAC address or assigned by any ID assignment protocol that ensures the globally unique property [15]. We assume that through pre-planning or a one-time initialization process, the central controller is aware of the identities of the deployed nodes.

The size of node ID t (in number of bits) is at least $\lceil \log(T) \rceil$ bits. In this paper we use $\log$ to represent logarithm of base 2 unless otherwise mentioned. We let $X_i = \{x_1, \ldots, x_{m_i}\}$ be the set of neighbors of a node i and m_i be the size of X_i. When the context is clear, we use m to represent m_i and X to represent X_i.

Connectivity information is uni-directional (links can be asymmetric, which is common in wireless networks [1, 7]). Based on existing link management process using periodic beacons, each node determines the set of connected neighbor nodes with incoming links. The definition of a connected neighbor depends on the application domain. For example, a node A can be considered to be connected to node B if at least one of the last several beacon packets transmitted by A can be received by B.

The connectivity monitoring process is performed by the central controller and the individual nodes to be monitored. Monitoring can be performed for a single node, the whole network, or any subset of nodes. Each monitored node sends its own neighborhood information to the central controller via possibly multiple hops.

4 Related Encoding Techniques

Before presenting our approach in Section 5, we first present other possible techniques in encoding (and possibly reducing) the amount of data by each node and their corresponding limitations. These techniques include direct transmission, bitmap, and hashing.

Direct Transmission: In the most direct form, a node transmits its neighbor IDs directly to the central controller. Without considering the packet overheads, the size of data to be sent is mt bits, where m is number of neighbors of the node and t is the size in bit of the node's ID. When mt is small, e.g. in a sparsely deployed network, direct transmission may be the most appropriate mechanism.

Bitmap Representation: With bitmap, each node transmits a bit string of size T to the central controller. The central controller decides whether node k is a neighbor of node i by looking at the k^{th} bit in the bit string node i transmits. The size of data transmitted is at least T bits if the bitmap representation is compact.

For large network, ($T \gg m$) and $m < \frac{T}{t}$, direct transmission is more efficient than bitmap. For mid-size network with relatively high density, m may be larger than $\frac{T}{t}$ and bitmap is more efficient.

If we consider the case where the bitmap can be efficiently compressed, the data size can be much smaller, especially for large sensor network. However, in any case, the maximal compression is lower bounded by $\log \binom{T}{m} \geq m(\log(T) - \log(m))$ [16].

Since the physical address of a sensor node (e.g., MAC address) can be 16 bits or even 32 bits and more, the identities of the sensor nodes need to be mapped to position in the bitmap for efficient representation. Such a mapping needs to be performed in advance and nodes have to be informed if there are changes to the mapping. For sensor network where self organization is important, use of pre-configured and static information is a serious drawback.

Exact Membership Testing Using Hashing Hashing is a common solution to compress the data for membership information. The space required to hash the neighbor IDs such that they can be decoded *exactly* is also lower bounded by $\log \binom{T}{m}$ [6].

5 H^{2}CM: Efficient and Flexible Connectivity Monitoring

In this section, we describe H^{2}CM, a Hop vector distance and Hashing-based Connectivity Monitoring scheme.

The central controller is assumed to maintain three sets of nodes for each monitored node i. The three sets are: V_i, the set of confirmed neighbors of node i; U_i, the set of nodes whose relationship with node i cannot be determined; and W_i, the set of confirmed non-neighbors of node i. Let $v_i = |V_i|$, $u_i = |U_i|$ and $w_i = |W_i|$.

Note that W_i is introduced only for convenience of the description and does not have to be maintained in practice. Initially, V_i and W_i are empty. U_i contains all other nodes in the network except node i. At all times, the union of i, V_i, W_i and U_i are the set of all nodes in the network T.

Each sensor node i transmits its own connectivity information to central controller. Intuitively, H^{2}CM tries to reduce u_i and increase v_i at central controller using the combination of several techniques so that the most appropriate technique can be applied in different situations. The objective is to achieve the desired accuracy with the minimum communication cost, where the accuracy is defined by $\frac{v_i}{m_i}$. Recall that m_i is the number of connected neighbor nodes with incoming links. Hence, $\frac{v_i}{m_i} \leq 1$.

The first technique of the algorithm applies when the values of $\frac{u_i}{m_i}$, m_i and u_i are large. The technique utilizes hop count to compute hop vector distance between nodes to identify possible set of neighbors. The value of u_i and thus $\frac{u_i}{m_i}$ maintained at central controller can be effectively reduced. This is presented in Section 5.1.

The second technique involves the use of Bloom filters for approximate membership testing. This technique is most appropriate when $\frac{u_i}{m_i}$ is less than some bounded value (see Section 5.3). Our use of Bloom filter is unique in two ways. First, traditional Bloom filter can only remove non-members (move elements from U_i to W_i). In our approach, Bloom filter can also confirm nodes as members (move elements from U_i to V_i). Second, we use a combination of normal and counting Bloom filters depending on the values of m_i, u_i and v_i. The details are presented in 5.2. Analysis on the behaviors of Bloom filters is provided in Section 5.3.

In the third technique, if $m_i - v_i$ and u_i are small enough, a node can use hashing to generate signature of all m_i nodes to help the central controller identify the complete set of its neighbors. The discussion is presented in Section 5.4.

Data needed for the techniques applied can be combined into a single packet resulting in only one transmission from each monitored node i. In Section 5.5, we describe how all these techniques are put together.

5.1 Hop Vector Distance

In order to more "accurately" decide if two nodes can be neighbors, location information is the most natural neighborhood constraint. Only nodes that are within the maximum communication distance can be neighbors. However, localization itself is a challenging research issue and often incurs substantial overhead. In this work, we propose the use of hop vector distance computed from hop count based localization [14] to remove a large amount of non-neighbors for each node when T is large.

We assume at the end of the localization process, the central controller knows the locations of all nodes. While the hop count localization process and collecting of location information from all nodes incur substantial cost, this process will only need to be performed once. It can be reused for later cycle of connectivity collection or update as long as there is no substantial change of this initialized hop counts to the relative locations of the neighbors. For a large network, taking into account the gain in reducing the candidate set for all nodes and the cost amortized over many monitoring cycles, the benefit can easily outweigh the cost.

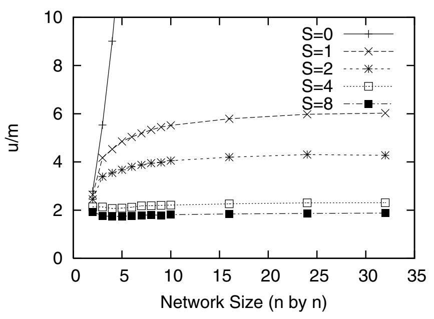
Assume that there are S anchors in the network and each node maintains its own hop count to the S anchors in a hop count vector $(h_{i1}, \ldots, h_{iS})$. The hop vector distance d between two nodes i and j is calculated using 3-norm distance between two hop count vectors, i.e., $d = \sqrt[3]{\left(\sum_{s=1}^{S} |h_{is} - h_{js}|^3\right)}$. A lower norm like 2-norm is not desirable because it is not able to differentiate the two cases where two nodes have absolute hop count difference vectors (2,0,0,0,0,...,0) and (1,1,1,1,0,...,0). 3-norm provides enough accuracy to differentiate most cases.

Each node sends to the central controller the ID of the neighbor node with the largest hop vector distance. With this information, the central controller can move nodes from U_i to W_i (reduce u_i). All nodes with larger hop vector distance cannot be a neighbor of node i. The utility of the hop vector distance based technique is in terms of the size of potential neighbors in U_i relative to the actual neighbor size m_i after this phase.

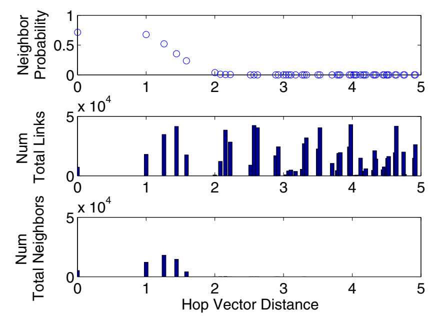
We evaluate the utility of the hop vector distance using simulation. The network area is varied from 2×2 to 32×32 and maximum transmission range is normalized to 1. The number of anchors S is set from 1 to 8 and the node density λ simulated is from 5 to 30. In order to generate graphs with different characteristics, we also define two parameters *link connectivity (LC)* and *link asymmetricity (LA)*. Link connectivity is defined as the ratio of node pairs that are able to communicate (at least in one direction) to nodes that are within maximum communication range. Link asymmetricity is defined as the percentage of asymmetric links over the total number of uni-directional and bi-directional links.

The result of one specific case when $\lambda = 10$, $LC = 0.8$ and $LA = 0.2$ is shown in Figure 1. The trends are similar for other cases. Figure 1(a) shows the ratio of average u and m versus the network size as well as the number of anchors S. Without the hop vector distance based filtering technique (number of anchors $S = 0$), the value of $\frac{\overline{u}}{\overline{m}}$ increases fast with the network size, where $\overline{u}$ and $\overline{m}$ are expected values of u and m. It can be observed that, with 4 anchors, the ratio of average u and m is less than 2.3 even for very large network. A larger number of anchors only performs marginally better.

Figure 1(b) shows the distribution of neighbors versus the hop vector distance d when $S = 4$. The top graph shows the probability that a node with a specific hop vector distance away from node i is actually a neighbor of i. The middle graph shows hop vector distance for all node pairs (2560x2559) with respect to the hop vector distance d. The distribution of actual number of neighbors of a node with respect to d is shown in the bottom figure. We can see that when the hop vector distance of



(a) Hop vector distance utility evaluation



(b) Neighbor probability versus hop vector distance

Figure 1. Effects of hop vector distance based technique.

two nodes is greater than 2, the probability that they are neighbors drops to almost 0. It in turn shows the effectiveness how hop vector distance can be used to reduce the value of u_i.

From the results, we can see that, filtering based on hop vector distance is very useful for reducing the size of U_i, especially in large sensor networks.

5.2 Bloom Filter

5.2.1 Bloom Filter Preliminaries

The standard form of Bloom filter represents a set $X = \{x_1, \ldots, x_m\}$ using a bit array of length b bits. k independent hash functions $f_1, \ldots, f_k$ are needed. Each of these function hashes input values uniformly into output values in the range $[1, b]$.

To construct the Bloom filter, the bit $f_i(x)$ of the bit array is set to 1 for each $i \in [1, k]$ and for each element $x \in X$. To check whether an element y is in X, we simply check whether the bit positions $f_i(y)$ for all $i \in [1, k]$ are 1. Bloom filter guarantees that there is no false negative, but there can be false positive.

There is another formulation of Bloom filter which has a slightly different form. The bit array of size b is divided into k disjoint bit arrays of size $\frac{b}{k}$ each. Each of the hash functions h_1 to h_k has an output range of $[1, \frac{b}{k}]$. To construct the Bloom filter, set the bit position $h_i(x)$

of bit array i to be 1 for each $x \in X$. The membership testing is similar to the standard form. The false positive probability can be approximated as $((1-(1-\frac{k}{b})^m)^k \approx (1-e^{-km/b})^k$, which is asymptotically close to the false positive rate of standard Bloom filter [4]. Given fixed values of m and k, the most efficient bit array size $\frac{b}{k}$ is $m\log(e)$.

A more general form of Bloom filter is to expand each bit in the bit array into a c bit counter. This is also known as the counting Bloom filter [13]. Whenever an element in x is hashed into an entry, we increase the counter associated with that entry by 1 if there is no overflow (greater than 2^c-1). Hence, a counting Bloom filter provides an exact count of the number of items that match that entry if there is no overflow. Note that when $c=1$, it becomes a normal Bloom filter.

As we will be using counting Bloom filter with different values of c, we choose to use the second form of Bloom filter in our design because it allows for easier combination of results from different bit arrays using different c values.

5.2.2 Bloom Filter Based Approach

To apply Bloom filter in the context of connectivity monitoring, we assume each node i sends the central controller k_i rounds of counting Bloom filters with number of bits per entry from c_1 to c_{k_i} for round 1 to round k_i respectively. For each round of Bloom filter, the bit positions are set according to the hash values of all the elements in neighbor set X_i using the corresponding hash function.

Upon receiving the Bloom filters from a node i, the central controller is then able to remove some nodes from U_i to W_i (reduce u_i), as well as from U_i to V_i (reduce u_i and increase v_i) according to the properties described below.

Nonmember Removal Property: The first property of a Bloom filter is as same as the traditional usage of Bloom filters explained in Section 5.2.1. We call it *Nonmember Removal* property. Upon receiving the bit array from a node, the central controller tests each element in U_i and moves those that are not neighbors from U_i to W_i. After enough rounds of nonmember removal from U_i to W_i, the set U_i will be V_i and the central controller can confirm the membership of set V_i (e.g., by checking the length of U_i is equal to the total number of neighbors of a node).

Membership Confirmation Property: The second property of Bloom filter applies when $X_i \subseteq (U_i \cup V_i)$, which means the initial guess of a node's neighborhood information at the central controller (U_i) by hop vector distance based scheme contains exactly all the neighbors of that node. We then have the following theorem.

Theorem 1 *Hash each element in X_i into a counting Bloom filter with number of bits per entry c ($c \geq 1$). Assuming the value of the counting Bloom filter at entry j is $s(j)$, then if there are only $s(j)$ elements in $U_i \cup V_i$ that hash into entry j, then the $s(j)$ elements in $U_i \cup V_i$ that hash into entry j must all be in X_i.*

Proof: This can be proved by contradiction. Consider one element is in U_i that hashes into entry j but is not in X_i. Since $X_i \subseteq (U_i \cup V_i)$, the number of elements in X_i that hashes into entry j cannot exceed $s(j)-1$, which is a contradiction because there are at least $s(j)$ elements in X hashed into entry j. ■

Upon receiving Bloom filter data from a node, the central controller can confirm that some elements in U_i are in X_i. We call this *Membership Confirmation* property. This is an interesting property because unlike the traditional usage of Bloom filters, which probabilistically tests whether an element is in the set, now we are able to confirm some portion of the elements are in the set. These elements are moved from U_i to V_i. Note that this property is not utilized in traditional Bloom filter applications because X_i is generally not a subset of $U_i \cup V_i$ whereas in our case X_i is always a subset of $U_i \cup V_i$.

Counting Removal Property: This property only applies for a counting Bloom filter with bits per entry c greater than 1. One property of the counting Bloom filter is that it supports deletion of an element when there is no overflow. Based on this property, the central controller can remove some elements of confirmed set V_i from the counting Bloom filter if overflow does not occur. We call this property as *Counting Removal* property. Give a Bloom filter bit array, this property should be applied if possible before the previous two properties to be applied.

An example showing how these properties can be applied is shown in Figure 2. In the example, initially $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$, V and W are empty. Two rounds of Bloom filters are applied with first round a normal Bloom filter with $c_1 = 1$ and second round a counting Bloom filter with $c_2 = 2$. After first round, x_6, x_7 and x_8 are moved into W because they all hash into a bit in the bit array with value of 0. x_3 and x_9 are moved to V according to membership confirmation property because they are the only nodes that hash into the bit array with bit value of 1. In round 2, firstly counting removal property is applied (x_3 and x_9), then according to nonmember removal property, x_2 is moved into W and according to membership confirmation property, x_1, x_4 and x_5 are moved into V.

Integrating these three properties together, the central controller is able to remove some of the non-neighbors from U_i to W_i, and it is also able to move the confirmed neighbors from U_i to V_i. Note that for nonmember re-

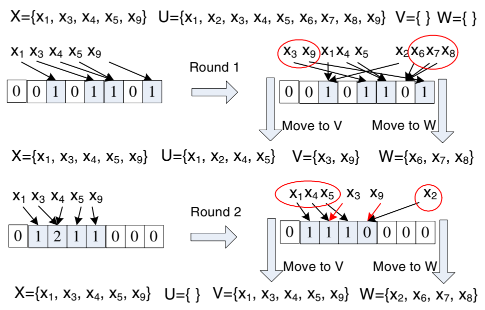

X={x1, x3, x4, x5, x9} U={x1, x2, x3, x4, x5, x6, x7, x8, x9} V={ } W={ }
x1 x3 x4 x5 x9
0 0 1 0 1 1 0 1
Round 1
x3 x9 x1 x4 x5 x2 x6 x7 x8
0 0 1 0 1 1 0 1
Move to V
Move to W
X={x1, x3, x4, x5, x9} U={x1, x2, x4, x5} V={x3, x9} W={x6, x7, x8}
x1 x3 x4 x5 x9
0 1 2 1 1 0 0 0
Round 2
x1 x4 x5 x3 x9 x2
0 1 1 1 0 0 0 0
Move to V
Move to W
X={x1, x3, x4, x5, x9} U={ } V={x1, x3, x4, x5, x9} W={x2, x6, x7, x8}

Figure 2. Example of Bloom filter.

moval property, a normal Bloom filter is more efficient than a counting Bloom filter, whereas for membership confirmation property and counting removal property, a counting Bloom filter is better than a normal one because there is less probability of overflow. The challenge now is when to use a normal Bloom filter ($c = 1$) and when to use a counting Bloom filter ($c > 1$) and of what size. We answer this question in the next section.

5.3 Bloom Filter Theoretical Analysis

In the following analysis, we assume the set of neighbor IDs of a node to be X, the confirmed neighbor set of this node at central controller to be V, and the non-confirmed neighbor set of this node at central controller to be U. Let $m = |X|$, $v = |V|$ and $u = |U|$. Also let $Y = V \cup U$ and $n = |Y| = v + u$. Note that $V \subseteq X$ and $X \subseteq Y$. Let $Z = V + U - X$ and $z = |Z| = v + u - m$. Z is therefore the set of nonmembers (but central controller is still not sure) in U.

For a Bloom filter sequence of $C = [c_1, \ldots, c_k]$, the number of total bits required is $b \sum_{j=1}^{k} c_j = m \log(e) \sum_{j=1}^{k} c_j$. This value increases linearly with k. To reduce the total number of bits required, k has to be bounded to a small value.

In this section, we investigate the effectiveness of how different sequences of Bloom filters C help in identifying neighbors and non-neighbors. We will first analyze the behavior of normal Bloom filters, followed by counting Bloom filters. The behavior of several rounds of mixed normal and counting Bloom filters will be studied at the end.

5.3.1 Normal Bloom Filters

Hashing m elements into a bit array of size $b = m\log(e)$, the probability that ith bit is set j times $P(m, b, s(i) = j)$ (or in short $P(m, b, j)$, or simply $P(j)$) is given by binomial distribution,

$$P(j) = \binom{m}{j} \left(\frac{1}{b}\right)^j \left(1 - \frac{1}{b}\right)^{m-j} = \frac{\binom{m}{j}}{(b-1)^j} P(0) \tag{1}$$

After a node hashes all its neighbor set X in a normal Bloom filter and sends the data to the central controller, the central controller will hash all nodes in U and V into the same size of bit array. The expected number of nonmember nodes can be removed from U is given by $P(0)z$. The number of nonmember nodes that still remain in U is then $(1 - P(0))z$.

Without considering those nodes already in V (without considering counting removal property), the number of nodes that can be confirmed by the central controller to be neighbors is, the number of bits in the bit string that only one node in X hashes into ($P(m, b, 1)b$), times the probability that for any bit, none of the node in Z (nonmember nodes) is hashed into. This is given by

$$P(1)bP(z, b, 0) \tag{2}$$

where $P(z, b, 0)$ is the probability that a bit remains 0 by hashing z nodes into bit array size of b. The percentage of nodes that will be confirmed by the central controller is then $p = \frac{P(1)bP(z,b,0)}{m} = P(1)P(z, b, 0)(\log e)$.

Similarly, in consecutive k rounds of normal Bloom filters, the average nonmember nodes in U at round j is $z_j = (1 - P(0))^{(j-1)}z$, and the average percentage of nodes that can be identified by the jth round p_j is given by (without considering the nodes that have been confirmed by previous rounds),

$$p_j = P(1)P(z(1 - P(0))^{(j-1)}, b, 0)(\log e) \tag{3}$$

5.3.2 Counting Bloom Filters

In general, a counting Bloom filter is not space-efficient compare to a normal Bloom filter for the purpose of nonmember removal. However, it can tolerate overflows so that the membership confirmation and counting removal property can be applied more efficiently. We will only consider counting Bloom filter of $c = 2$ in this section for the following two reasons. First, we want to use Bloom filter to provide better performance in reducing communication cost than direct transmission of compressed data. A Bloom filter of $c > 2$ is too costly. Second, in our application, when $c > 2$, the gain of tolerance on overflow is small compare to $c = 2$.

Without considering the set of confirmed neighbors, V, Equation 2 can be generalized to,

$$(P(1) + 2P(2) + 3P(3))bP(z, b, 0) \tag{4}$$

Note that for normal Bloom filters in sequence of two, the total number of nodes identified is (excluding

V),

$$(p_1 + p_2 - p_1 p_2)m \tag{5}$$

where, p_1,p_2 are given in Equation 3.

Comparing Equation 4 and 5, we can show that if $n > \frac{2\log 0.7661}{\log(1-1/(m\log e))} + m$, two consecutive normal Bloom filter will be better than a counting Bloom filter of $c = 2$. We omit the proof due to space limit.

However, what has been analyzed on counting Bloom filter is purely based on the nonmember removal and membership confirmation properties. Counting Bloom filter has one more advantage: if the central controller has already identified some portion of elements to be in X, these elements can be deleted from the counting Bloom filters if there is no overflow.

Recall that $v = |V|$ is the number of nodes that have already been identified by the central controller. By deleting them from the counting Bloom filter, there will be more "0" entries available which can be useful in nonmember removal property. Since $c = 2$, it is possible to delete those that have already been identified only when number of elements hashed into the bit is 1 or 2. Deleting the element(s) will give more 0 entries. The increase of percentage of 0 entries is given by $P(1)\frac{v}{m} + 2P(2)\frac{v^2}{m^2}$.

When v is close to m, the increase in number of 0 entries is large. Removing more nonmembers will help confirmation and removal in the next round of Bloom filters, and will help to increase the chance of successfully using signature hash for identification as will be explained in Section 5.4.

The discussion leads to the following heuristic. If the number of confirmed neighbors (v) is small comparing to the actual number of neighbors (m), two normal Bloom filters ($c = 1$) tends to perform better a counting Bloom filter with $c = 2$) of the same total size, and vice versa.

5.3.3 Bloom Filters of k Rounds

In this section, we will try to generalize the previously mentioned heuristic to multiple rounds (≥ 2) of normal and counting Bloom filter. The goal is to use just enough Bloom filter data so that the number of unconfirmed nodes $m - v$ is smaller than some pre-defined value or small enough to utilize a simple signature hash value for complete identification (Section 5.4).

For each round i, if it is normal Bloom filter, we have

$$\begin{aligned} p_i &= P(1)P(z,b,0)\log e \\ z' &= (1 - P(0))z \\ p' &= p + p_i - p_i p \end{aligned} \tag{6}$$

If it is a counting Bloom filter of $c = 2$, we have

$$\begin{aligned} p_i &= (P(1) + 2P(2) + 3P(3))P(z,b,0)\log e \\ z' &= (1 - P(0) - P(1)p - 2P(2)p^2)z \\ p' &= p + p_i - p_i p \end{aligned} \tag{7}$$

where z' and p' are the updated value of z and p for the next round respectively.

The final total percentage of nodes can be identified in average is then,

$$\sum_{i=1\to k} p_i - \sum_{i=1\to k, j=1\to k, i\neq j} p_i p_j + ... \tag{8}$$

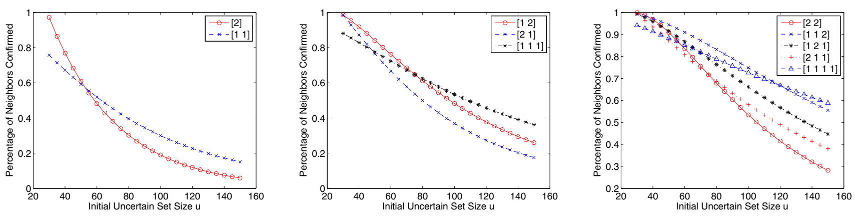
As an illustration, we plot the percentage of neighbors confirmed by the central controller for $\overline{m} = 30$ in Figure 3 for different values of initial uncertain set size $\overline{u}$. Values of different $\sum_{j=1}^{k} c_j$ are plotted, where c_j is the value of c for the Bloom filter size at round j. We use $[c_1, \ldots, c_k]$ to represent Bloom filters of k rounds. In the plots, all combinations of using $c = 1$ or 2 to sum to the values 2, 3 and 4 are compared. The results for other values of m are similar to Figure 3 except that the crossover points occur at different values.

From the figure, it can be observed that the Bloom filter sequence that starts with $c_1 = 1$ always outperforms those that starts with $c_1 = 2$. This coincides with the theorem shown in Section 5.3.2. From Section 5.1, we also see that with using hop vector distance as a filter, $\frac{\overline{u}}{\overline{m}}$ is between 2 to 2.5 when $S = 4$ (i.e., initial value of $\overline{u}$ input to Bloom filters is between 60 to 75). This corresponds to 40% to 60% of neighbors to be confirmed in the best case in Figure 3(a), 65% to 85% in Figure 3(b), and 85% to 95% in Figure 3(c). $\sum c_j = 4$ with sequence [1 1 2] or [1 1 1 1] are good choices under this situation.

Given the neighbor set size m_i (known at node i), as well as initial uncertain set size u_i, node i is able to estimate the best Bloom filter sequence it requires for the central controller to confirm at least the pre-defined percentage of neighbors given by Equation 8. Considering the limited computational resources, these equations may seem complex for implementation on the sensor nodes. Fortunately, in our applications, the ratio $\frac{\overline{u}}{\overline{m}}$ is almost always below 2.5 after the first technique is applied. Thus, the values of $\sum_{j=1}^{k} c_j$ tends to be small to give a good performance.

5.4 Signature Hashing

Because Bloom filter is a probabilistic structure, as more neighbor nodes are recognized, less new members can be confirmed at the next round. To completely recognize all the neighbors, a large amount of rounds may be required.



(a) $\sum_j c_j = 2$ (b) $\sum_j c_j = 3$ (c) $\sum_j c_j = 4$

Figure 3. Comparison of consecutive Bloom filters ($\overline{m} = 30$).

However, if most of the neighbors have been recognized by the central controller, a node can just simply choose to hash all its neighbor IDs into a signature value (say 32 bits), and append this signature value to the bit arrays generated by the Bloom filter to the central controller. After applying Bloom filter on the bit arrays, the central controller may perform search using the hash value to obtain the complete list of neighbors if needed.

The cost of searching is upper bound by $\binom{u}{m-v}$. If the central controller sets the search threshold per node to 10^6, then any value of $u \leq 22$ can be searched, independent of the value of $m - v$ (since $\binom{22}{11} < 10^6$). Similarly, if the search threshold is 10^4, then $u \leq 12$ is always fine.

In cases where the central controller finds it too expensive to perform the search, it can ask for more rounds of Bloom filters data from those nodes, which would be very few in number. Finally, note that since hash values can collide, there is a very small probability that some neighbors are falsely identified from nodes in U. However, the collision probability is very small when u is small and hash value is sufficiently large.

It is also worth noting that, with signature hashing, the complete neighborhood information of each node can be obtained with very high probability. However, its communication cost can be even smaller than the scheme without signature hashing where only a percentage of neighbors could be obtained, because for the former case, less rounds of Bloom filters can be required. This is verified by comparing the simulation results in Section 6.1 and 6.2.

5.5 Flow of H^{2}CM

There are two parts to the execution, one on the central controller and the other on the nodes monitored. We only show the pseudo code on the monitored nodes in Algorithm 1. In addition, for ease of explanation, we only show the algorithm for complete connectivity (with signature hashing). If only a pre-defined percentage of neighbors is required and additional computation cost is also allowed at central controller, the node has to estimate and compare the communication costs of the schemes with and without signature hashing, and chooses the one with lower cost. This is not included in Algorithm 1.

Each monitored node first estimates the required Bloom filter sequence such that the communication cost can be minimized while the searching threshold can be satisfied (line 1-15). Note that this estimation is based on the fact that signature hashing will be applied. If the signature hashing will not be applied, the estimation shall be based on the percentage of neighbors confirmed as analyzed in Section 5.3. This is not shown in the pseudo code.

Note that the value of u (line 18) is closely related to the network parameters such as *link connectivity* and *link asymmetricity*, and it can be obtained in several ways. A node can either estimate locally based on the value of m or obtain from central controller's broadcast message. The simulation in Section 5.1 shows that with hop count vector based localization and with relatively low link connectivity and high link asymmetricity, $2m$ to $2.5m$ is the good approximation on upper bound of u (i.e., in line 18, $\alpha = 2$ to 2.5). We will also apply this settings in our evaluation of large scale sensor networks.

After computing the Bloom filter bit sequence, a check (line 21)[1] is performed to see if it is better to simply send the node IDs directly instead. Otherwise, the Bloom filter sequence and signature of the neighbor IDs are sent to the central controller (line 24-29).

5.6 Extensions

We present two extensions to the algorithm. First, since the connectivity changes over time, the algorithm may need to be applied periodically. However, if the connectivity of the whole network does not change too much, collecting the connectivity information from scratch periodically may not be a good idea. The common way to perform incremental update is by differential method, i.e., a node will only report to the central controller about what has changed.

[1]Note that in the evaluation section, to compare the performance of H^{2}CM with other techniques like maximal compression of bitmap, this check is not performed.

Algorithm 1 H^{2}CM at Each Monitored Node

```
Require: Neighbor table $X$, hop count location of neighbors, neighbor ratio $\alpha$
    and searching threshold $thresh$.
function BFS($u, m, b, thresh$)
    $nBins \leftarrow 1, search \leftarrow \infty$
    while $search > thresh$ do
        for $c \in \{c | \sum c[i] = nBins, c[i] = 1$ or $2\}$ do
            Calculate $z$ and $v$ using Equation 8
            $u \leftarrow m - v + z$
            if $\binom{u}{m-v} < search$ then
                $search \leftarrow \binom{u}{m-v}, bfs \leftarrow c$
            end if
        end for
        $nBins \leftarrow nBins + 1$
    end while
    return $bfs$
end function

$j \leftarrow \arg\max_{i \in X} d_i$
$m \leftarrow |X|, u \leftarrow \lceil \alpha m \rceil, b \leftarrow \lceil \log(e)m \rceil, s \leftarrow m \log(|T|)$
$c \leftarrow$ BFS($m, u, b, thresh$)
if $\log(|T|) + b(\sum c[i]) + sizeof(sig) \geq s$ then
    Send IDs directly
else
    Allocate space of $b(\sum c[i])$ bits
    for $i \in [1, len(c)]$ do
        Do Bloom filtering of each neighbor IDs
    end for
    $sig \leftarrow$ signatures of neighbor IDs
    Send ID[$j$], Bloom filter and $sig$
end if
```

The proposed algorithm can be easily extended to this differential update process. For the set of neighbors that has been removed, the original neighbor set at the central controller becomes the initial uncertain set U. Each node knows the exact value of $|U|$ and number of neighbors that have been removed, so it may estimate the Bloom filter sequence required with good accuracy. For the set of new neighbors, the initial uncertain set U is the one constrained by the hop vector distance minus the original neighbor set. Same algorithm as introduced in previous sections can be applied.

Second, the algorithm presented is for a single connectivity threshold. It is possible to extend the approach to monitor discrete link quality values with a small number of discrete levels. For example, to retrieve the link quality information of a node, we can first apply the connectivity monitoring algorithm introduced starting from the lowest link quality. Once the neighborhood information for the lowest link quality is known, we proceed with the next higher link quality by setting the initial uncertain set to be the set of confirmed neighbors in the previous round. The algorithm proceeds till the highest link quality.

6 Evaluation

In this section, we show our evaluation results using both simulation and testbed experiments. In the simulations, we assume that the packets can be delivered without any loss. In fact, as long as the hop vectors

λ	[ ]		[1]		[1 1]		[1 1 1]		[1 1 2]	
-	u	m-v	u	m-v	u	m-v	u	m-v	u	m-v
5	19	10	10	6.6	5.3	3.3	2.3	1.4	1.0	0.6
10	46	21	27	15	15	9.8	8.1	5.3	3.7	2.0
20	105	42	64	34	38	23	21	13.9	10	5.9
30	166	63	103	52	57	36	31	22	17	9.2

Table 1. Average values of u_i and $(m_i - v_i)$ after applying Bloom filter.

of sensor nodes are known to the central controller, any subsequent packet losses only affect the information accuracy of the node that initiates the packet, and they do not affact the overall correctness and efficiency of the algorithm. Also note that since energy consumptions of sensor nodes are dominated by wireless communication costs, in the simulations, we only consider the communication costs.

6.1 Large Network without Hashing

In the first set of experiments, we evaluate the performance of connectivity monitoring in a large network using hop vector distance filtering and Bloom filter. Signature hashing is not performed.

In order to compare the simulation results with the analysis in Section 5.3, and to illustrate the performance of different sequences of Bloom filters, we choose to use fixed sequence of Bloom filter. Therefore, the set of c_j used is the same for all nodes (instead of depending on m as proposed in the algorithm).

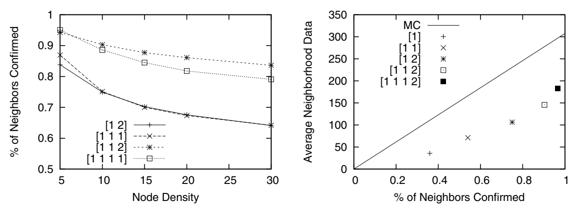
We simulate a large network of size 32×32 with node density varying from 5 to 30 (uniform distribution) per unit square. The maximum wireless communication range is normalized to 1. Each node has a unique ID of size 16 bits, which can support a network of size 2^{16}.

While a wide range of link connectivity and asymmetricity have been evaluated, we will only show the result for the setting of link connectivity and link asymmetry equal to 0.8 and 0.2 respectively. The communication cost is set to $\sum_j c_j$ = 0, 1, 2, 3 or 4.

Table 1 shows the average values of u and $m - v$ for different node density after utilizing the bit patterns generated by different sequence of c_js. The integers within the square braces denote the values of c_j used. Note that [] means no Bloom filter data is utilized (only the hop vector distance based scheme is performed).

It can be observed that for low node density, such as 5, even without any Bloom filter, it is still possible to search based on signature hash value if computational threshold is set to 10^6. Even for high node density, a Bloom filter sequence of [1 1 2] can still allow the central controller to confirm about 90% of the neighbors (without applying signature hashing).

Figure 4(a) shows the results when different combinations of c_j are used with different node densities. The



(a) $LC = 0.8, LA = 0.2$ (b) $\lambda = 10$, $LC = 0.8$, $LA = 0.2$

Figure 4. With hop vector & Bloom filter.

result shows that sequence [1 1 2] performs better than that of [1 1 1 1]. This coincides with the results in Figure 3. [1 1 1 1] is better than [1 1 2] only when initial uncertain set size u is much larger than number of neighbors m.

Figure 4(b) shows the average neighborhood information sent at each node versus the percentage of neighbors confirmed for different Bloom filter sequences. The line (MC) in the plot shows the neighborhood data required to let central controller confirm the same percentage of neighbors using maximal compression. The result shows that by using Bloom filters, the cost is strictly less than (about 50% to 60% of) the cost of maximal compression.

6.2 Performance in Large Network

In this simulation, we evaluate the performance of H^{2}CM in large sensor network. All three techniques are used and the length of the signature hash used is 32 bits. The network setting is same as previous section.

First, as an illustration of the utility of the signature hash, the cumulative distribution function for number of searches required after the Bloom filter sequence [1 1 2] is applied is shown in Figure 5(a). It can be seen that a large portion of nodes (80%) require little or no additional computational (10^4 or less) even for high node density of $\lambda = 30$. If one allows a search cost limit of 10^6, then for node density of 10 ($> 10,000$ nodes), close to 100% of all neighbors can be found in all our simulations. With node density of 30, less than 5% of nodes will require larger Bloom filter sequence ($\sum_j c_j > 4$).

The communication costs of different connectivity monitoring approaches are shown in Figure 5(b). They are maximal compression (MC), fixed c_j sequence of [1 1 2] for all nodes (BF [1 1 2]) and two cases where each node chooses its Bloom filter sequence such that the number of searches required at central controller is smaller than 10^6 and 10^4 respectively. These are labeled as VarBF(10^6) and VarBF(10^4). In the algorithm, we assume that $\frac{u_i}{m_i}$ is 2.5 after hop vector distance filtering.

For fixed Bloom filter sequence, the saving is about half the cost of maximal compression. VarBF(10^6) achieves the most savings. At low node density, the saving is up to 85%. Even at high node density, the reduction is 65%. The improvement of VarBF(10^4) over static sequence is small, indicating that significant search cost may be needed before the Bloom filter sequence can be shortened.

VarBF(10^6) is able to obtain all neighbor information in most cases except for the highest node density. When node density is 30, we observe that for less than 3% of the nodes, the computational cost needed at the central controller exceeds 10^6. Among these nodes with unconfirmed neighbors, the average number of unconfirmed links is 17 out of an average of 66 neighbors.

6.3 Performance in Mid-Size Network

In this section, we study the performance of our algorithm in a medium size sensor network using simulation. The network is a 4×4 square and node density varies from 5 to 30. The average total number of nodes in the network is from 80 to 500. As sensor testbeds with hundreds of nodes have been built (e.g. the Kansei sensor testbed), networks of such sizes are of practical interest.

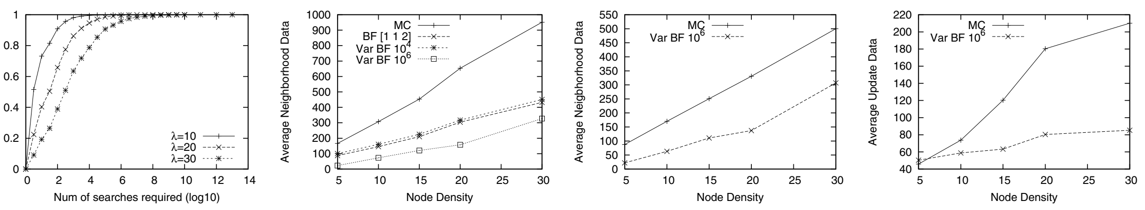
Figure 5(c) shows the result of average data required at each node for MC and VarBF(10^6). Each data point is an average of 100 runs. 4 beacons are used and signature hash is 32 bits. The link connectivity used is 0.8 and asymmetricity is 0.2. The result shows that communication cost can be reduced by 40% to 70%. The number of unconfirmed links is very small. However, we observe that, among all the simulation instances, a very small number of nodes wrongly identify their set of neighbors due to collision of signature (6 out of 160,000 cases).

6.4 Connectivity Update

In this section, we evaluate the performance of H^{2}CM for differential update where 10% of the existing links are removed and same number of new links among random chosen neighbor pairs are added. Simulation result is shown in Figure 5(d). With VarBF(10^6), the communication cost varies from about 35% to 45% of the cost using MC at high density. At low node density, the cost of VarBF(10^6) can be higher as only few neighbors have changed. Nevertheless, the total cost is small as well.

6.5 Testbed Evaluation

In this section, we present evaluation on a 34 node testbed made up of a combination of Mica2 and Mica2Dot nodes installed in a typical indoor office environment. We show that H^{2}CM can be efficiently implemented in TinyOS and run in actual deployment using



(a) CDF of number searches (b) Data sent per node (large) (c) Data sent per node (mid) (d) Update per node (large)

Figure 5. Performance of H^{2}CM in large and midsize networks.

real sensor motes. In our implementation, we use only 80 lines of NesC code and 600 bytes of extra image size (code size).

In the evaluation, 33 nodes sent connectivity information to a single Mica2 mote (central controller) via the collection tree. Link layer packet retransmissions are also enabled to cope with possible packet losses. Since the number of nodes in the network is small, we do not consider hop count information and only apply Bloom filter and signature based hashing. The total data size required per node for H^{2}CM is 40 bits (21 bit hash), which is the same as using bitmap. Note that sending neighbor IDs directly requires much larger data size comparing to H^{2}CM and bitmap.

We run the experiment over 12 hours and obtained over 4000 snapshots of the overall connectivity. Due to the small signature size, the collision probability of the signature hash is about 0.4%. When we increase the data size to 48 bits and set the signature hash to 29 bits, we do not find any collisions during the experiments.

7 Conclusion

In this paper, we presented an algorithm that can efficiently monitor connectivity of wireless sensor networks for various sizes. Given estimates of the network size and node density, H^{2}CM selects one or more techniques to obtain connectivity. Simulation results show that H^{2}CM works best for large network (> 1000 nodes) achieving savings of up to 85% compare to maximal compression of neighborhood information, even to achieve the complete connectivity information. We also have demonstrated that the algorithm is practical and can be easily implemented on TinyOS with little overhead.

Acknowledgement

This work is supported by the National University of Singapore under research grant R-252-000-265-112.

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *Proceedings of ACM SIGCOMM'04*, 2004.

[2] A.Kumar, J. Xu, L. Li, and J.Wang. Space-code Bloom filter for efficient traffic flow measurement. In *Proceedings of ACM SIGCOMM Conference on Internet Measurement'03*, 2003.

[3] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[4] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Maathematics*, 2004.

[5] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery over adaptive overlay networks. *ACM SIGCOMM Computer Communication Review*, pages 47–60, 2002.

[6] L. Carter, R. Floyd, J. Gill, G. Markowsky, and M. Wegman. Exact and approximate membership testers. In *Proceedings of STOC'78*, 1978.

[7] A. Cerpa, N. Busek, and D. Estrin. SCALE: A tool for simple connectivity assessment in lossy environments. Technical Report 21, CENS UCLA, 2003.

[8] R. Chandra, C. Fetzer, and K. Högstedt. A mesh-based robust topology discovery algorithm for hybrid wireless networks. Technical report.

[9] B. Cheswick. Internet mapping project. http://www.cheswick.com/ches/map/index.html.

[10] B. Deb, S. Bhatnagar, and B. Nath. STREAM: Sensor topology retrieval at multiple resolutions. *Journal of Telecommunications, Special Issue on Wireless Sensor Networks*.

[11] B. Deb, S. Bhatnagar, and B. Nath. A topology discovery algorithm for sensor networks with applications to network management. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, 2002.

[12] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *Proceedings of ACM SIGMETRICS'05*, 2005.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, pages 281–293, 2000.

[14] D. Niculescu and B. Nath. Ad-hoc positioning system. In *Proceedings of GLOBECOM'01*, 2001.

[15] E. Ould-Ahmed-Vall, D. M. Blough, B. S. Heck, and G. F. Riley. Distributed unique global ID assignment for sensor networks. In *Proceedings of IEEE MASS'05*.

[16] A. Pagh, R. Pagh, and S. S. Rao. An optimal Bloom filter replacement. In *Proceedings of ACM-SIAM SODA'05*, 2005.

[17] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of ACM Sensys'05*, 2005.

[18] A. rousskov and D. Wessels. Cache digests. *Computer Networks and ISDN Systems*, pages 2155–2168, 1998.

[19] N. Semret. African Internet topology and traffic report. http://comet.columbia.edu/ nemo/netmap.

[20] B. Zhang, R. Liu, D. Massey, and L. Zhang. Collecting the Internet AS-level topology. *ACM SIGCOMM Computer Communication Review (CCR), special issue on Internet Vital Statistics*, 2005.

[21] M. Zhang, M. C. Chan, and A. Ananda. Location-aided topology discovery for wireless sensor networks. In *Proceedings of IEEE International Conference on Communications (ICC)'08*, 2008.