Published in IET Wireless Sensor Systems
Received on 10th December 2012
Revised on 25th June 2013
Accepted on 1st July 2013
doi: 10.1049/iet-wss.2013.0028

ISSN 2043-6386

Efficient mechanism to exchange group membership identities among nodes in wireless sensor networks

Alekha Kumar Mishra, Ashok Kumar Turuk

Department of Computer Science and Engineering, National Institute of Technology Rourkela, Rourkela, India
E-mail: alekha@gmail.com

Abstract: It is necessary for nodes in wireless sensor networks to exchange their group membership identities in certain applications such as cluster formation and clone detection. The communication cost associated with exchanging group membership identities is higher especially in a dense network. Existing schemes make use of Bloom filter in exchanging membership information. Although the Bloom filter is an efficient mechanism for exchanging membership information, yet it suffers from higher probability of false positive, that is, a node may be detected to be a member of a group, when it is not. In this study, the authors propose two schemes called transpose bit-pair coding (TBC), and sub-mat coding (SMC) for exchanging group membership information. The proposed schemes do not generate false positive, and have lower communication and storage overhead. The authors have compared TBC and SMC with Bloom filter. Parameters considered for the comparison are: (i) communication overhead in terms of number of bits required to exchange the group membership information and (ii) probability of false positive. It is observed that the communication overhead in TBC and SMC is significantly lower in comparison with Bloom filter and the schemes have no cases of false positive.

1 Introduction

A wireless sensor network (WSN) [1, 2] consists of a large number of tiny sensing devices that are densely deployed around the target. WSN supports a wide range of applications [3–5]. Certain applications such as subset or cluster formation, and clone detection require the exchange of group membership information among the nodes for replica detection. Group membership information consist of the identities of all nodes in the group. Trivial method of exchanging the group membership information is to send the individual identity of all nodes in the group. However, this method leads to higher message overhead even for a group of smaller size. Another method to exchange group membership information is to use bit-stream. The number of bits in the bit-stream is equal to the size of the network. The ith bit in the bit-stream is set to one, if the node with identity, i, is a member of the group. In this scheme, the size of the bit-stream increases proportionately with the size of the network. For larger networks, the overhead associated with communication and storage is higher. Another mechanism proposed in the literature for exchanging the identity of members in a group is to use Bloom filter [6–8]. This is an efficient technique for exchanging group membership information. However, it suffers from higher cases of false positive; where a node is found to be a member of a group when it is not.

Exchange of group membership information among the nodes is a store and forward process. For exchanging group membership information the overhead associated with the schemes should not only be lower, but also do not give rise to any case of false positive. In this paper, we propose two schemes called: (i) transpose bit-pair coding (TBC), and (ii) sub-mat coding (SMC) to reduce the communication and storage overhead associated with exchanging group membership information among nodes. The proposed schemes not only reduce the communication and storage overhead, but also do not generate any false positive.

The rest of the paper is organised as follows: related works reported in the literature are presented in Section 2. The proposed schemes for exchanging group membership information are described in Section 3. Performance of the proposed schemes vis-a-vis a few existing ones is summarised in Section 4, and finally a few conclusions are made in Section 5.

2 Related work

One of the simplest method for exchanging group membership information is to exchange the identity of each member of the group. This form of exchange was adopted by Choi *et al.* [9], Meng *et al.* [10], and Naruephiphat and Charnsripinyo [11] in their proposed schemes. The number of identities exchanged per message depends on the size of the message. With the increase in the network size, the number of nodes per group also increases. As a result, the number of message exchange required to share the group membership information increases. Therefore this method of exchange is not suitable for a dense network. To reduce the communication overhead, a subset of group membership identities is exchanged instead of the identities of entire group membership in [10]. This scheme can reduce some

communication overhead, but cannot provide the complete group membership information of a node.

Znaidi *et al.* [12], Zhang *et al.* [13], and Deng and Xiong [14] have used Bloom filter [6–8] in their proposed schemes to share the group membership information. A Bloom filter consists of the following: (i) One b-bit array B to store and test the membership of a node, and (ii) k number of hash functions, $H_i()$, $0 \le i \le k$ to map the identity of a group member to certain number of bits in B.

To add a member ID_x to the Bloom filter, all $H_i(ID_x)$ positions are set to one in the bit-array B. The membership of a node ID_y is tested by checking for a value one in every $H_i(ID_y)$ position of B. Group membership sharing mechanisms that are based on Bloom filter have lower memory and communication cost. However, they suffer from higher probability of false positive, that is, they may wrongly infer a node to be a member of a group, when it is not a member of that group. Given k, the probability of false positive p is given by

$$p = \left(1 - \left(1 - \frac{1}{b}\right)^{k.n}\right)^{k} \simeq \left(1 - \mathrm{e}^{-(k.n)/b}\right)^{k} \tag{1}$$

where b is the length of the array B and n is the number of members in a group. For a given value of k and b, the probability of false positive p, increases with increase in the size n and decreases as b increases. The optimal number of hash functions is given by

$$k = \left(\frac{b}{n}\right) \ln 2 \tag{2}$$

For a given false positive probability p, the length of the Bloom filter, b, is proportional to the number of elements being filtered, n, that is

$$b = -\frac{n \ln p}{(\ln 2)^2} \tag{3}$$

For an optimal value of k, Bloom filter with one per cent of error requires 9.6 bits per element. Size of the array B determines the communication and storage cost associated with Bloom filter. Each additional 4.8 bits per element, decreases the error rate by 1/10. Further details on Bloom filter can be found in [6–8].

Another method for exchanging group membership information is the use of bit-stream [15, 16]. In this scheme, the number of bits in a bit-stream is equal to the size of the network. The ith bit in the bit-stream is set to one, if the node with identity, i, is a member of the group. This method works well in a network of fixed size. However, the size of the bit-stream increases proportionately with the size of the network. In a dense network with relatively smaller group size, the overhead associated with communication and storage is higher.

Data compression techniques for WSNs are proposed in [17–19]. These techniques operate on sensed data and make extensive use of RAM. Therefore they are not suitable for exchanging group membership in WSN.

The bit-stream based mechanisms do not generate false positive. However, they have higher storage and communication cost. Mechanisms based on Bloom filter have higher cases of false positive. The communication and storage cost associated with such mechanisms are lower. Therefore we need a scheme that does not generate false positive, and the associated communication and storage cost is comparable with that of Bloom filter. In TBC and SMC, we have tried to balance between the cases of false positive and associated cost.

3 Proposed mechanism

In this section, we propose two schemes: (i) TBC, and (ii) SMC for exchanging group members' identity among the nodes. In the proposed scheme, group membership information is encoded into a bit-stream, which is exchanged among the nodes. The encoded bit-stream is stored at a node and is decoded to obtain the group membership information. The encoding technique is lossless and does not generate false positive.

The format used in representing the group membership information is described in Section 3.1. TBC and SMC are described in Section 3.2 and 3.3, respectively.

3.1 Representation of group membership information

In this sub-section, we describe the representation of group membership information in the proposed schemes. Let N be the network size, n_G be the average size of a group, where $n_G < N$, and S_G be an arbitrary group. Members of the group, S_G, are represented in the form of a matrix $\boldsymbol{M}$, as shown below

$$\boldsymbol{M} = \begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & m_{0,\mathrm{Dim}-1} \\ m_{1,0} & m_{1,1} & \cdots & m_{1,\mathrm{Dim}-1} \\ \cdots & \cdots & \cdots & \cdots \\ m_{\mathrm{Dim}-1,0} & m_{\mathrm{Dim}-1,1} & \cdots & m_{\mathrm{Dim}-1,\mathrm{Dim}-1} \end{bmatrix} \tag{4}$$

where, the element $m_{i,j}$ stores 1-bit of information about the membership of a node whose identity is $(Dim^* i + j)$. Given the N, n_G and S_G, the matrix $\boldsymbol{M}$ is constructed as follows

$$m_{i,j} = \begin{cases} 1 & \text{iff node with an identity, } \left(Dim^* i + j\right) \in S_G \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

We have assumed the network size, $N \le \mathrm{Dim} \times \mathrm{Dim}$, and the identity of nodes are unique integral value in [0, N–1].

3.2 Transpose bit-pair coding

TBC takes the group membership matrix $\boldsymbol{M}$, as input and produces a bit-stream, C, where $|C| \ll N$. The bit-stream is generated considering each pair of elements $m_{i,j}$ and $m_{j,i}$ of $\boldsymbol{M}$. An element is considered only once in the generation of a bit-stream. Let c be the partial bit-stream generated from a pair of elements $m_{i,j}$ and $m_{j,i}$. Then, c is generated as shown below

$$c = \begin{cases} m_{i,j} & i = j \\ 0 & i \ne j,\ m_{i,j} = m_{j,i} = 0 \\ 10 & i \ne j,\ m_{i,j} = 1,\ m_{j,i} = 0 \\ 110 & i \ne j,\ m_{i,j} = 0,\ m_{j,i} = 1 \\ 111 & i \ne j,\ m_{i,j} = 1,\ m_{j,i} = 1 \end{cases} \tag{6}$$

The encoding process in TBC is given in Algorithm 1 (see Fig. 1), where only the upper triangular matrix of $\boldsymbol{M}$ is

doi: 10.1049/iet-wss.2013.0028

Algorithm 1:

Input: Matrix, $\boldsymbol{M}$
Output: Bit-stream, C

1 $k \longleftarrow 0$
2 **for** $i \leftarrow 0$ **to** $Dim - 1$ **do**
3 $\quad C(k) \longleftarrow m_{i,i}$
4 $\quad k \longleftarrow k + 1$
5 $\quad$ **for** $j \leftarrow i + 1$ **to** $Dim - 1$ **do**
6 $\quad\quad$ **if** $m_{i,j} = 0$ & $m_{j,i} = 0$ **then**
7 $\quad\quad\quad C(k) \longleftarrow 0$
8 $\quad\quad\quad k \longleftarrow k + 1$
9 $\quad\quad$ **else if** $m_{i,j} = 1$ & $m_{j,i} = 0$ **then**
10 $\quad\quad\quad C(k) \longleftarrow 1,\ C(k+1) \longleftarrow 0$
11 $\quad\quad\quad k \longleftarrow k + 2$
12 $\quad\quad$ **else if** $m_{i,j} = 0$ & $m_{j,i} = 1$ **then**
13 $\quad\quad\quad C(k) \longleftarrow 1,\ C(k+1) \longleftarrow 1,\ C(k+2) \longleftarrow 0$
14 $\quad\quad\quad k \longleftarrow k + 3$
15 $\quad\quad$ **else**
16 $\quad\quad\quad C(k) \longleftarrow 1,\ C(k+1) \longleftarrow 1,\ C(k+2) \longleftarrow 1$
17 $\quad\quad\quad k \longleftarrow k + 3$

Fig. 1 *Transpose bit-pair coding*

Algorithm 2:

Input: Bit-stream, C
Output: Matrix, $\boldsymbol{M}'$

1 $k \longleftarrow 0$
2 **for** $i \leftarrow 0$ **to** $Dim - 1$ **do**
3 $\quad m'_{i,i} \longleftarrow C(k)$
4 $\quad k \longleftarrow k + 1$
5 $\quad$ **for** $j \leftarrow i + 1$ **to** $Dim - 1$ **do**
6 $\quad\quad$ **if** $C(k) = 0$ **then**
7 $\quad\quad\quad m'_{i,j} \longleftarrow m'_{j,i} \longleftarrow 0$
8 $\quad\quad$ **else**
9 $\quad\quad\quad k \longleftarrow k + 1$
10 $\quad\quad\quad$ **if** $C(k) = 0$ **then**
11 $\quad\quad\quad\quad m'_{i,j} \longleftarrow 1,\ m'_{j,i} \longleftarrow 0$
12 $\quad\quad\quad$ **else**
13 $\quad\quad\quad\quad k \longleftarrow k + 1$
14 $\quad\quad\quad\quad$ **if** $C(k) = 0$ **then**
15 $\quad\quad\quad\quad\quad m'_{i,j} \longleftarrow 0,\ m'_{j,i} \longleftarrow 1$
16 $\quad\quad\quad\quad$ **else**
17 $\quad\quad\quad\quad\quad m'_{i,j} \longleftarrow 1,\ m'_{j,i} \longleftarrow 1$

Fig. 3 *Transpose bit-pair decoding*

traversed. The bit-stream C can also be generated by traversing only the lower triangular matrix of $\boldsymbol{M}$. In Step 3 of Algorithm 1 (see Fig. 1) each element of the main diagonal is read, and written into the bit-stream. From Step 6 to 17 a pair of elements at index (i, j) and (j, i) of $\boldsymbol{M}$ is read and the corresponding bit-stream is generated as per the transition diagram shown in Fig. 2*a*. The time complexity in generating the bit-stream is $O(\text{Dim}^2) = O(N)$ as $N \simeq \text{Dim}^2$.

The decoding process in TBC is given in Algorithm 2 (see Fig. 3). Step 3 of the algorithm generates the element across the main diagonal. Step 5 to 17 generates a pair of elements $m_{i,j}$ and $m_{j,i}$ of matrix $\boldsymbol{M}$ as per the transition diagram shown in Fig. 2*a*. The time complexity of the decoding process in TBC is $O(N)$. We claim the following from the encoding and decoding process in TBC.

Claim 1: The encoding and decoding process in TBC is unique.

Proof: The encoding and decoding process in TBC is said to be unique, if the following two conditions hold true:

(i) The mapping f: $\boldsymbol{M} \rightarrow C$ is one-to-one, and
(ii) The decoding of bit-stream C is unambiguous

The mapping function f: $\boldsymbol{M} \rightarrow C$ is uniquely defined by the (6). The state transition diagram shown in Fig. 2*a* reads a pair of element and uniquely generates the corresponding bit-stream. Therefore the mapping function f is one-to-one.

The decoding process follows the prefix codes, that is, no code is a prefix of any other code. Transition diagram in Fig. 2*b* shows the decoding process. From the transition

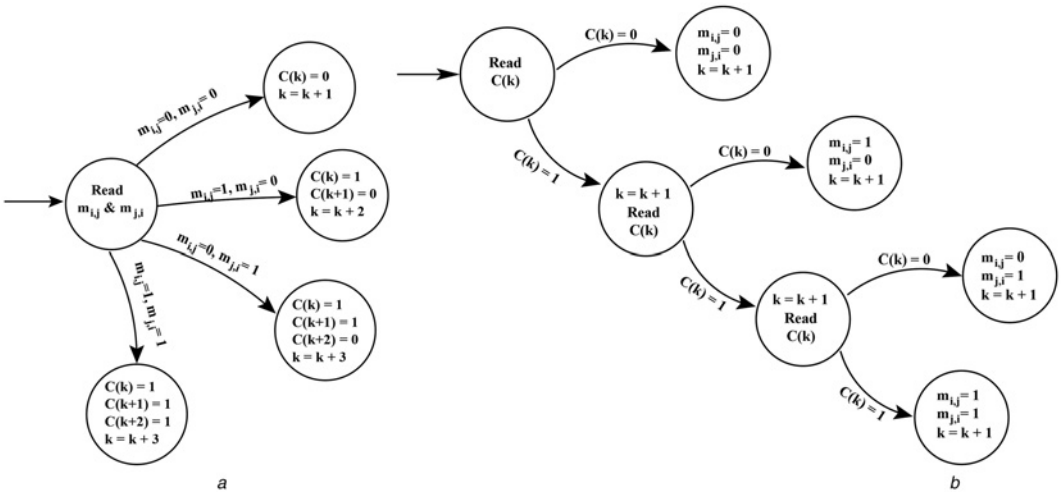

Fig. 2 *Transition diagrams showing*
a Encoding in TBC
b Decoding in TBC

Table 1 Encoding of matrix, ***M***, in TBC

Indices $(i, j)(j, i)$	Bits $m_{i,j}m_{j,i}$	Code c	Code $(i, j)(j, i)$	Bits $m_{i,j}m_{j,i}$	Code c	Indices $(i, j)(j, i)$	Bits $m_{i,j}m_{j,i}$	Code c
(0, 0)(0, 0)	00	0	(0, 1)(1, 0)	00	0	(0, 2)(2, 0)	00	0
(0, 3)(3, 0)	00	0	(0, 4)(4, 0)	00	0	(0, 5)(5, 0)	00	0
(0, 6)(6, 0)	10	10	(0, 7)(7, 0)	00	0	(0, 8)(8, 0)	00	0
(0, 9)(9, 0)	00	0	(1, 1)(1, 1)	11	1	(1, 2)(2, 1)	00	0
(1, 3)(3, 1)	00	0	(1, 4)(4, 1)	00	0	(1, 5)(5, 1)	00	0
(1, 6)(6, 1)	00	0	(1, 7)(7, 1)	00	0	(1, 8)(8, 1)	00	0
(1, 9)(9, 1)	11	111	(2, 2)(2, 2)	00	0	(2, 3)(3, 2)	00	0
(2, 4)(4, 2)	00	0	(2, 5)(5, 2)	10	10	(2, 6)(6, 2)	00	0
(2, 7)(7, 2)	00	0	(2, 8)(8, 2)	10	10	(2, 9)(9, 2)	00	0
(3, 3)(3, 3)	00	0	(3, 4)(4, 3)	00	0	(3, 5)(5, 3)	10	10
(3, 6)(6, 3)	00	0	(3, 7)(7, 3)	00	0	(3, 8)(8, 3)	10	10
(3, 9)(9, 3)	00	0	(4, 4)(4, 4)	00	0	(4, 5)(5, 4)	00	0
(4, 6)(6, 4)	11	111	(4, 7)(7, 4)	00	0	(4, 8)(8, 4)	00	0
(4, 9)(9, 4)	00	0	(5, 5)(5, 5)	00	0	(5, 6)(6, 5)	00	0
(5, 7)(7, 5)	00	0	(5, 8)(8, 5)	00	0	(5, 9)(9, 5)	10	10
(6, 6)(6, 6)	00	0	(6, 7)(7, 6)	00	0	(6, 8)(8, 6)	00	0
(6, 9)(9, 6)	00	0	(7, 7)(7, 7)	00	0	(7, 8)(8, 7)	01	110
(7, 9)(9, 7)	00	0	(8, 8)(8, 8)	00	0	(8, 9)(9, 8)	00	0
(9, 9)(9, 9)	00	0						

diagram it is observed that the decoding process is unambiguous. □

We illustrate below the encoding and decoding process in TBC with a suitable example. Let us consider a network with the following parameters: $N = 100$, $n_G = 12$, and $S_G = \{6, 11, 19, 25, 28, 35, 38, 46, 59, 64, 87, 91\}$. The matrix $\boldsymbol{M}$ representing S_G is given below

$$\boldsymbol{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Table 1 shows the encoding process of matrix $\boldsymbol{M}$, to generate the bit-stream C for the given S_G. The bit-stream generated by TBC for S_G in the above example is {0000001000010000000 1110001000100001000100001110000000100000011000000}. The generated bit-stream gives the membership information of a group, which is shared among the nodes.

The decoding algorithm generates a matrix $\boldsymbol{M}'$, from which the membership information of the group is obtained. The elements of $\boldsymbol{M}'$ for the bit-stream, C, considered in the above example is shown in Table 2. From $\boldsymbol{M}'$, we obtain the membership information of the group as {6, 11, 19, 25, 28, 35, 38, 46, 59, 64, 87, 91}, which is same as the one considered in the example.

3.3 Sub-mat coding

In SMC scheme, the membership matrix $\boldsymbol{M}$, is logically divided into 2×2 sub-matrices as shown below. Each sub-matrix of $\boldsymbol{M}$ is then used to generate a portion of the bit-stream C (see (7)).

where

$$\boldsymbol{M}_{i,j} = \begin{bmatrix} m_{2^*i,2^*j} & m_{2^*i,2^*j+1} \\ m_{2^*i+1,2^*j} & m_{2^*i+1,2^*j+1} \end{bmatrix} \tag{8}$$

Let c be the partial bit-stream generated from a sub-matrix

$$\boldsymbol{M} = \begin{bmatrix} \begin{bmatrix} m_{0,0} & m_{0,1} \\ m_{1,0} & m_{1,1} \end{bmatrix} & \cdots & \begin{bmatrix} m_{0,\mathrm{Dim}-2} & m_{0,\mathrm{Dim}-1} \\ m_{1,\mathrm{Dim}-2} & m_{1,\mathrm{Dim}-1} \end{bmatrix} \\ \cdots \quad \begin{bmatrix} m_{\mathrm{Dim}-2,0} & m_{\mathrm{Dim}-2,1} \\ m_{\mathrm{Dim}-1,0} & m_{\mathrm{Dim}-1,1} \end{bmatrix} & \cdots & \cdots \quad \begin{bmatrix} m_{\mathrm{Dim}-2,\mathrm{Dim}-2} & m_{\mathrm{Dim}-2,\mathrm{Dim}-1} \\ m_{\mathrm{Dim}-1,\mathrm{Dim}-2} & m_{\mathrm{Dim}-1,\mathrm{Dim}-1} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{M}_{0,0} & \boldsymbol{M}_{0,2} & \cdots & \boldsymbol{M}_{0,\frac{\mathrm{Dim}}{2}-1} \\ \boldsymbol{M}_{2,0} & \boldsymbol{M}_{2,2} & \cdots & \boldsymbol{M}_{2,\frac{\mathrm{Dim}}{2}-1} \\ \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{M}_{\mathrm{Dim}-2,0} & \boldsymbol{M}_{\mathrm{Dim}-2,2} & \cdots & \boldsymbol{M}_{\frac{\mathrm{Dim}}{2}-1,\frac{\mathrm{Dim}}{2}-1} \end{bmatrix} \tag{7}$$

doi: 10.1049/iet-wss.2013.0028

Table 2 Decoding of bit-stream, C, in TBC

Index (i, j)	Input bit	$m'_{i,j}$ & $m'_{j,i}$	Index (i, j)	Input bit	$m'_{i,j}$ & $m'_{j,i}$
0, 0	0	$m'_{0,0} = 0$	0, 1	0	$m'_{0,1} = m'_{1,0} = 0$
0, 2	0	$m'_{0,2} = m'_{2,0} = 0$	0, 3	0	$m'_{0,3} = m'_{3,0} = 0$
0, 4	0	$m'_{0,4} = m'_{4,0} = 0$	0, 5	0	$m'_{0,5} = m'_{5,0} = 0$
0, 6	1	read next bit	0, 6	0	$m'_{0,6} = 1$, $m'_{6,0} = 0$
0, 7	0	$m'_{0,7} = m'_{7,0} = 0$	0, 8	0	$m'_{0,8} = m'_{8,0} = 0$
0, 9	0	$m'_{0,9} = m'_{9,0} = 0$	1, 1	1	$m'_{1,1} = 1$
1, 2	0	$m'_{1,2} = m'_{2,1} = 0$	1, 3	0	$m'_{1,3} = m'_{3,1} = 0$
1, 4	0	$m'_{1,4} = m'_{4,1} = 0$	1, 5	0	$m'_{1,5} = m'_{5,1} = 0$
1, 6	0	$m'_{1,6} = m'_{6,1} = 0$	1, 7	0	$m'_{1,7} = m'_{7,1} = 0$
1, 8	0	$m'_{1,8} = m'_{8,1} = 0$	1, 9	1	read next bit
1, 9	1	read next bit	1, 9	1	$m'_{1,9} = 1$, $m'_{9,1} = 1$
2, 2	0	$m'_{2,2} = 0$	2, 3	0	$m'_{2,3} = m'_{3,2} = 0$
2, 4	0	$m'_{2,4} = m'_{4,2} = 0$	2, 5	1	read next bit
2, 5	0	$m'_{2,5} = 1$, $m'_{5,2} = 0$	2, 6	0	$m'_{2,6} = m'_{6,2} = 0$
2, 7	0	$m'_{2,7} = m'_{7,2} = 0$	2, 8	1	read next bit
2, 8	0	$m'_{2,8} = 1$, $m'_{8,2} = 0$	2, 9	0	$m'_{2,9} = m'_{9,2} = 0$
3, 3	0	$m'_{3,3} = 0$	3, 4	0	$m'_{3,4} = m'_{4,3} = 0$
3, 5	1	read next bit	3, 5	0	$m'_{3,5} = 1$, $m'_{5,3} = 0$
3, 6	0	$m'_{3,6} = 1$, $m'_{6,3} = 0$	3, 7	0	$m'_{3,7} = 1$, $m'_{7,3} = 0$
3, 8	1	read next bit	3, 8	0	$m'_{3,8} = 1$, $m'_{8,3} = 0$
3, 9	0	$m'_{3,9} = 1$, $m'_{9,3} = 0$	4, 4	0	$m'_{4,4} = 0$
4, 5	0	$m'_{4,5} = m'_{5,4} = 0$	4, 6	1	read next bit
4, 6	1	read next bit	4, 6	1	$m'_{4,6} = m'_{6,4} = 1$
4, 7	0	$m'_{4,7} = m'_{7,4} = 0$	4, 5	0	$m'_{4,8} = m'_{8,4} = 0$
4, 5	0	$m'_{4,9} = m'_{9,4} = 0$	5, 5	0	$m'_{5,5} = 0$
5, 6	0	$m'_{5,6} = m'_{6,5} = 0$	5, 7	0	$m'_{5,7} = m'_{7,5} = 0$
5, 8	0	$m'_{5,8} = m'_{8,5} = 0$	5, 9	1	read next bit
5, 9	0	$m'_{5,9} = 1$, $m'_{9,5} = 0$	6, 6	0	$m'_{6,6} = 0$
6, 7	0	$m'_{6,7} = m'_{7,6} = 0$	6, 8	0	$m'_{6,8} = m'_{8,6} = 0$
6, 9	0	$m'_{6,9} = m'_{9,6} = 0$	7, 7	0	$m'_{7,7} = 0$
7, 8	1	read next bit	7, 8	1	read next bit
7, 8	0	$m'_{7,8} = 0$, $m'_{8,7} = 1$	7, 9	0	$m'_{7,9} = m'_{9,7} = 0$
8, 8	0	$m'_{8,8} = 0$	8, 9	0	$m'_{8,9} = m'_{9,8} = 0$
9, 9	0	$m'_{9,9} = 0$			

$M_{i,j}$. Then, c is generated as shown below (see (9))

where $\|$ is used as concatenation operator.

Algorithm 3 (see Fig. 4) shows the encoding process in SMC. In Step 2–13 of the algorithm each element of the sub-matrix $\boldsymbol{M}_{i,\,j}$ is read and its corresponding bit-stream is generated using (9). Fig. 5*a* shows the generation of bit-stream from each sub-matrix $\boldsymbol{M}_{i,j}$. The time complexity of this algorithm is $O(\text{Dim}^2) = O(N)$.

Decoding process in SMC is shown in Algorithm 4 (see Fig 6). Step 2–13 of the algorithm construct each sub-matrix. Transition diagram in Fig. 5*b* shows the construction of sub-matrices. The time complexity of the decoding process is $O(N)$. We claim the following from the encoding and decoding process in SMC.

Claim 2: The process of encoding and decoding in SMC is unique.

Proof: The encoding and decoding process in SMC is said to be unique, if the following two conditions hold true:

(i) The mapping f: $\boldsymbol{M} \rightarrow C$ is one-to-one and
(ii) The decoding of bit-stream C is unambiguous

Algorithm 3:

Input: Matrix, $\boldsymbol{M}$

Output: Bit-stream, C

1 $k \longleftarrow 0$
2 **for** $i \leftarrow 0$ **to** $Dim - 1$ *with increment 2* **do**
3 **for** $j \leftarrow 0$ **to** $Dim - 1$ *with increment 2* **do**
4 **if** $m_{i,j} = m_{i,j+1} = m_{i+1,j} = m_{i+1,j+1} = 0$ **then**
5 $C(k) \longleftarrow 0$
6 $k \longleftarrow k + 1$
7 **else**
8 $C(k) \longleftarrow 1$
9 $C(k+1) \longleftarrow m_{i,j}$
10 $C(k+2) \longleftarrow m_{i,j+1}$
11 $C(k+3) \longleftarrow m_{i+1,j}$
12 $C(k+4) \longleftarrow m_{i+1,j+1}$
13 $k \longleftarrow k + 5$

Fig. 4 *Sub-MAT coding*

$$c = \begin{cases} 0 & \text{if } \boldsymbol{M}_{i,j} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ 1 \| m_{2^*i,2^*j} \| m_{2^*i,2^*j+1} \| m_{2^*i+1,2^*j} \| m_{2^*i+1,2^*j+1} & \text{otherwise} \end{cases} \tag{9}$$

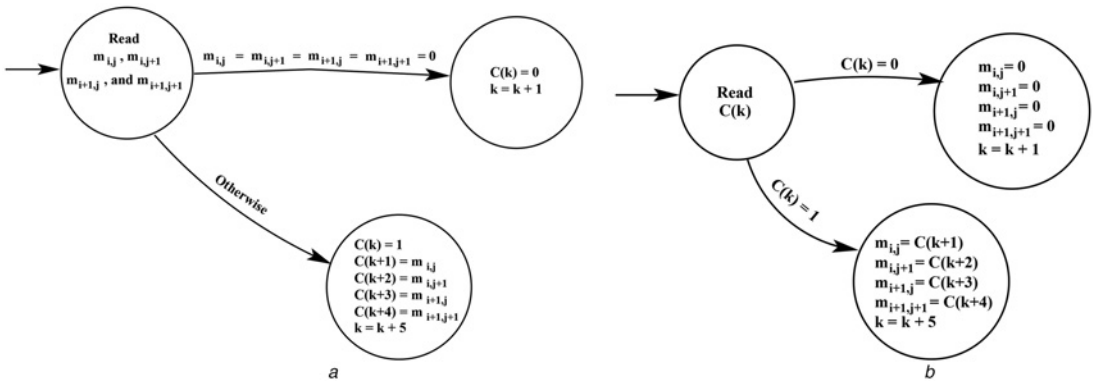

Fig. 5 *Transition diagrams showing*

a Encoding in SMC
b Decoding in SMC

Algorithm 4:

Input: Bit-stream, C

Output: Matrix, $\boldsymbol{M}'$

```
k ⟵ 0
for i ← 0 to Dim with increment 2 do
    for j ← 0 to Dim with increment 2 do
        if C(k) = 0 then
            m'_{i,j} ← m'_{i,j+1} ← m'_{i+1,j} ← m'_{i+1,j+1} ← 0
            k ⟵ k + 1
        else
            m'_{i,j} ⟵ C(k + 1)
            m'_{i,j+1} ⟵ C(k + 2)
            m'_{i+1,j} ⟵ C(k + 3)
            m'_{i,j} ⟵ C(k + 4)
            k ⟵ k + 5
```

Fig. 6 *Sub-MAT decoding*

In SMC, the mapping function f: $\boldsymbol{M} \rightarrow C$ generates a bit-stream corresponding to the sub-matrix $\boldsymbol{M}_{i,j}$ as given in (9). Fig. 5*a* shows that the generation of bit-stream is one-to-one. The decoding of the bit-stream is unambiguous as shown in Fig. 5*b*. □

We explain below the encoding and decoding process in SMC using the example in Section 3.2. Each sub-matrix of $\boldsymbol{M}$ is given below

$$\boldsymbol{M} = \begin{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

The code generated from each of sub-matrix $\boldsymbol{M}_{i,j}$ of $\boldsymbol{M}$ represented in the form of a matrix, $\boldsymbol{M}^c$, is shown below

Table 3 Decoding of bit-stream, C in SMC

Index (i, j)	Input Bit	Action	Matrix
0, 0	1	read next 4 bits	$m'_{0,0} = 0$, $m'_{0,1} = 0$, $m'_{1,0} = 0$, $m'_{1,1} = 1$
0, 2	0	–	$m'_{0,2} = m'_{0,3} = m'_{1,2} = m'_{1,3} = 0$
0, 4	0	–	$m'_{0,4} = m'_{0,5} = m'_{1,4} = m'_{1,5} = 0$
0, 6	1	read next 4 bits	$m'_{0,6} = 1$, $m'_{0,7} = 0$, $m'_{1,6} = 0$, $m'_{1,7} = 0$
0, 8	1	read next 4 bits	$m'_{0,8} = 0$, $m'_{0,9} = 0$, $m'_{1,8} = 0$, $m'_{1,9} = 1$
2, 0	0	–	$m'_{2,0} = m'_{2,1} = m'_{3,0} = m'_{3,1} = 0$
2, 2	0	–	$m'_{2,2} = m'_{2,3} = m'_{3,2} = m'_{3,3} = 0$
2, 4	1	read next 4 bits	$m'_{2,4} = 0$, $m'_{2,5} = 1$, $m'_{3,4} = 0$, $m'_{3,5} = 1$
2, 6	0	–	$m'_{2,6} = m'_{2,7} = m'_{3,6} = m'_{3,7} = 0$
2, 8	1	read next 4 bits	$m'_{2,8} = 1$, $m'_{2,9} = 0$, $m'_{3,8} = 1$, $m'_{3,9} = 0$
4, 0	0	–	$m'_{4,0} = m'_{4,1} = m'_{5,0} = m'_{5,1} = 0$
4, 2	0	–	$m'_{4,2} = m'_{4,3} = m'_{5,2} = m'_{5,3} = 0$
4, 4	0	–	$m'_{4,4} = m'_{4,5} = m'_{5,4} = m'_{5,5} = 0$
4, 6	1	read next 4 bits	$m'_{4,6} = 1$, $m'_{4,7} = 0$, $m'_{5,6} = 1$, $m'_{5,7} = 0$
4, 8	1	read next 4 bits	$m'_{4,8} = 0$, $m'_{4,9} = 0$, $m'_{5,8} = 1$, $m'_{5,9} = 1$
6, 0	0	–	$m'_{6,0} = m'_{6,1} = m'_{7,0} = m'_{7,1} = 0$
6, 2	0	–	$m'_{6,2} = m'_{6,3} = m'_{7,2} = m'_{7,3} = 0$
6, 4	1	read next 4 bits	$m'_{6,4} = 1$, $m'_{6,5} = 0$, $m'_{7,4} = 1$, $m'_{7,5} = 0$
6, 6	0	–	$m'_{6,6} = m'_{6,7} = m'_{7,6} = m'_{7,7} = 0$
6, 8	0	–	$m'_{6,8} = m'_{6,9} = m'_{7,8} = m'_{7,9} = 0$
8, 0	1	read next 4 bits	$m'_{8,0} = 0$, $m'_{8,1} = 0$, $m'_{9,0} = 1$, $m'_{9,1} = 1$
8, 2	0	–	$m'_{8,2} = m'_{8,3} = m'_{9,2} = m'_{9,3} = 0$
8, 4	0	–	$m'_{8,4} = m'_{8,5} = m'_{9,4} = m'_{9,5} = 0$
8, 6	1	read next 4 bits	$m'_{8,6} = 0$, $m'_{8,7} = 1$, $m'_{9,6} = 0$, $m'_{9,7} = 0$
8, 8	0	–	$m'_{8,8} = m'_{8,9} = m'_{9,8} = m'_{9,9} = 0$

doi: 10.1049/iet-wss.2013.0028

Table 4 Simulation parameters

Parameter	Value
network size (N)	1024–10 000
group size	2.5–15 (in % of N)
number of groups	10–40
maximum number of neighbours	150
matrix dimension (Dim)	32–100

$$\boldsymbol{M}^c = \begin{bmatrix} 10001 & 0 & 0 & 11000 & 10001 \\ 0 & 0 & 10101 & 0 & 11010 \\ 0 & 0 & 0 & 11000 & 10001 \\ 0 & 0 & 11000 & 0 & 0 \\ 10001 & 0 & 0 & 10100 & 0 \end{bmatrix}$$

The matrix $\boldsymbol{M}^c$, is read row-wise to generate the bit-stream C. The bit-stream C generated by SMC for the group S_G considered in the example is {1000100110001000100101010110100001100010001001100000100010010100 0}.

Decoding of the bit-stream C in SMC is shown in Table 3.

4 Simulation and results

In this section, we analysed the performance of TBC and SMC through simulation. Parameters considered for simulation are shown in Table 4. We have compared TBC and SMC with the following schemes: (i) that exchange the identity of each node; we call this as Scheme$_1$ and (ii) that uses bit-stream of size N; we call this as Scheme$_2$ and (iii) that uses Bloom filter. We have assumed that nodes are identified by a unique integer and is represented by 16 bits. Table 5 shows the space required in terms of number of bits by each scheme to store the group membership information. Size of the group is considered to be 150. Scheme$_1$ and that uses Bloom filter depends only on the group size. Therefore the number of bits required in these two schemes is constant for network of different size. Scheme$_2$ solely depends on the network size. The number of bits required in Scheme$_2$ increases with the network size. Both TBC and SMC depend on the group size and network size. Therefore the number of bits required increases marginally with network size.

Next, we varied the size of the group from 2.5% to 15% of N. Tables 6–8 show the number of bits required when the network size, N is 1024, 4900, and 10 000, respectively. It is observed from the Tables 6–8 that SMC requires a lesser number of bits in comparison with other schemes for a given group size. This is because, in SMC and TBC the bit-stream is generated from a sparse matrix representing the group membership information. More the sparseness in the matrix, lesser the number of bits generated.

Then, we compared with a metric called space saving percentage. We define the space saving percentage of a scheme with respect to a base scheme as follows (see (10))

The plot for space saving percentage against group size of TBC and SMC with respect to Bloom filter is shown in Figs. 7*a* and *b*, respectively. It is observed from the figures that the space saving percentage is higher in TBC and SMC for larger group size. This is because, the number of bits required to represent group membership information in Bloom filter is directly proportional to the group size.

We have also compared the proposed schemes with Bloom filter for false positive cases. The plot for the false detection probability against group size is given in Fig. 8*a*. The value of b is taken to be 750 bits and the number of hash functions, k is set to 5, 10 and 15. It is observed from the figure that the false detection probability increases with the increase in number of hash functions for a fixed value of b. Fig. 8*b* shows the plot for false detection probability against group size, where the value of b is varied to 250, 500 and 750 bits. The value of k is taken to be 5. It is observed from the figure that the false detection probability decreases with the increase in the size of b.

Fig. 9 shows the comparison of SMC with Bloom filter for false positive cases. False detection probability of Bloom

Table 5 Number of bits required to store group membership information in Scheme$_1$, Scheme$_2$, Bloom filter, TBC and SMC

N	Scheme$_1$	Scheme$_2$	TBC	SMC	Bloom filter with 1% error	Bloom filter with 0.1% error
1024	2400	1024	778	737	1440	2160
2500	2400	2500	1550	1175	1440	2160
3600	2400	3600	2124	1426	1440	2160
4900	2400	4900	2793	1813	1440	2160
6400	2400	6400	3520	2176	1440	2160
8100	2400	8100	4374	2592	1440	2160
10 000	2400	10000	5400	3100	1440	2160

Table 6 Number of bits required to store group membership information in Scheme$_1$, Scheme$_2$, Bloom filter, TBC, and SMC for N = 1024

Group size (% of N)	Scheme$_1$	Scheme$_2$	TBC	SMC	Bloom filter with 1% error	Bloom filter with 0.1% error
2.5	410	1024	573	359	246	369
5	819	1024	615	461	492	738
7.5	1229	1024	646	533	738	1106
10	1639	1024	686	604	983	1475
15	2458	1024	758	737	1475	2212

$$\frac{\text{Number of bits required in the base scheme} - \text{Number of bits required by the new scheme}}{\text{Number of bits required in base scheme}} * 100 \quad (10)$$

Table 7 Number of bits required to store group membership information in $Scheme_1$, $Scheme_2$, Bloom filter, TBC, and SMC for N = 4900

Group size (% of N)	$Scheme_1$	$Scheme_2$	TBC	SMC	Bloom filter with 1% error	Bloom filter with 0.1% error
2.5	1968	4900	2646	1715	1176	1764
5	3920	4900	2842	2156	2352	3528
7.5	5888	4900	3038	2500	3528	5292
10	7840	4900	3185	2891	4704	7056
15	11760	4900	3577	3528	7056	10584

Table 8 Number of bits required to store group membership information in $Scheme_1$, $Scheme_2$, Bloom filter, TBC, and SMC for N = 10 000

Group size (% of N)	$Scheme_1$	$Scheme_2$	TBC	SMC	Bloom filter with 1% error	Bloom filter with 0.1% error
2.5	4000	10000	5400	3400	2400	3600
5	8000	10000	5800	4400	4800	7200
7.5	12000	10000	6100	5200	7200	10800
10	16000	10000	6500	5900	9600	14400
15	24000	10000	7200	7300	14400	21600

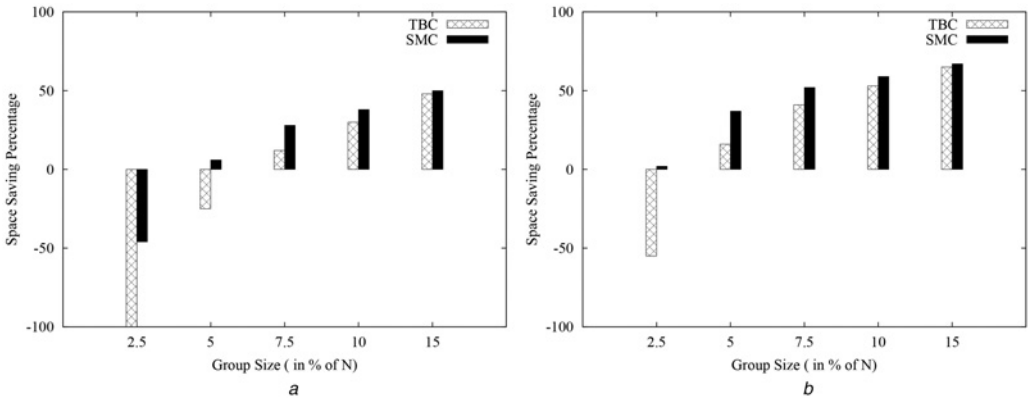

Fig. 7 *Space saving percentage against network size in TBC and SMC*

a Comparison with Bloom filter (1% error)
b Comparison with Bloom filter (0.1% error)

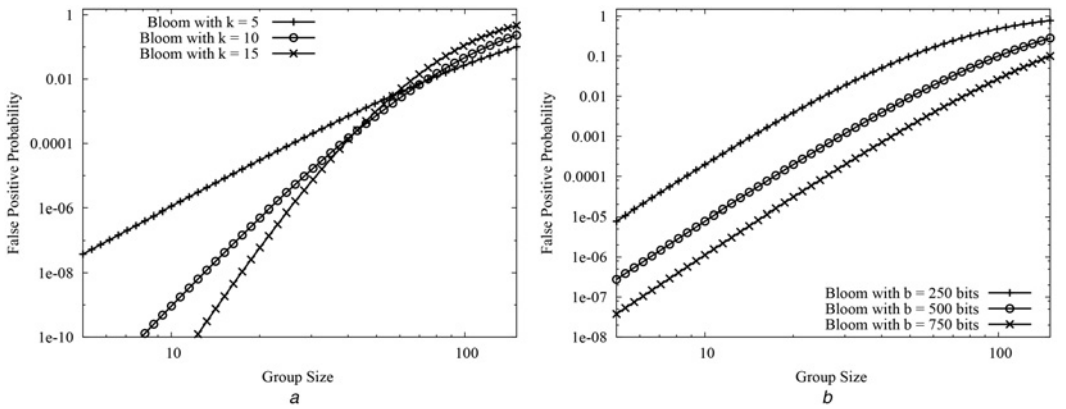

Fig. 8 *False positive probability of bloom filter against group size*

a For different value of k
b For different value of b

filter is plotted varying the value of b to 737, 1175 and 1813 for network size of 1024, 2500 and 4900, respectively. The value of k is considered to be 5. The false detection probability of Bloom filter increases with the increase in group size for a given network size, whereas, SMC do not give rise to false positive cases.

doi: 10.1049/iet-wss.2013.0028

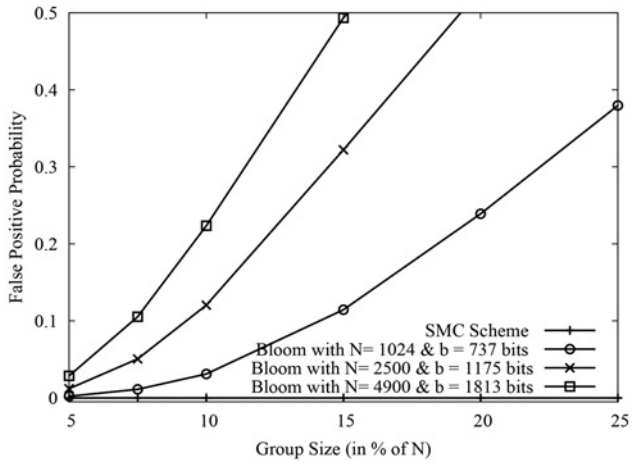

Fig. 9 *False positive probability against group size in bloom filter and SMC*

5 Conclusion

In this paper, we discussed the mechanisms for exchanging group membership information among the nodes in WSN. The strengths and weaknesses in existing mechanisms are identified. Mechanism for exchanging group membership information should have lower communication and storage overhead. The aforementioned properties are satisfied by the mechanisms that are based on Bloom filter. However, Bloom filter suffers from false positive. In this paper, we proposed two mechanisms called TBC and SMC. Both TBC and SMC generates a bit-stream from the membership matrix. The encoding and decoding process in each scheme is described. We have also shown that the encoding and decoding process in TBC and SMC is unique. TBC and SMC are compared with two trivial schemes and one that uses Bloom filter. The parameters considered for comparison are number of bits required to store group membership information, space saving percentage, and false positive probability. It is found that TBC and SMC do not generate false positive and the space saving percentage is significantly higher compared with Bloom filter. The encoding process in TBC and SMC can further be improved to achieve higher space saving without generating false positive.

6 References

1 Buratti, C., Conti, A., Dardari, D., Verdone, R.: 'An overview on wireless sensor networks technology and evolution', *Sensors*, 2009, **9**, (9), pp. 6869–6896

2 Zheng, J., Jamalipour, A.: 'Wireless sensor networks a networking perspective' (John Wiley and Sons, 2009)

3 Sohraby, K., Minoli, D., Znati, T.: 'Wireless sensor networks technology, protocols, and applications' (John Wiley and Sons, 2007)

4 Zhang, Y., Zhang, S.F., Ji, Y., Wu, G.X.: 'Wireless sensor network-enabled intravenous infusion monitoring', *IET Wirel. Sensor Syst.*, 2011, **1**, (4), pp. 241–247

5 Yongtai, H., Lihui, L., Yanqiu, L.: 'Design of solar photovoltaic micro-power supply for application of wireless sensor nodes in complex illumination environments', *IET Wirel. Sensor Syst.*, 2012, **2**, (1), pp. 16–21

6 Broder, A., Mitzenmacher, M.: 'Network applications of bloom filters: a survey', *Internet Math.*, 2002, **1**, (4), pp. 636–646

7 Mitzenmacher, M.: 'Compressed bloom Filters', *IEEE Trans. Netw. (TON)*, 2002, **10**, (5), pp. 604–612

8 Jimeno, M., Christensen, K.J., Roginsky, A.: 'Two-tier bloom filter to achieve faster membership testing', *Electron. Lett.*, 2008, **44**, (7), pp. 503–504

9 Choi, H., Zhu, S., Porta, T.F.L.: 'SET: detecting node clones in sensor networks'. IEEE Proc. Third Int. Conf. Security and Privacy in Communications Networks and the Workshops, SecureComm 2007, 2007, pp. 341–350

10 Meng, X., Lin, K., Li, K.: 'A note-based randomized and distributed protocol for detecting node replication attacks in wireless sensor networks'. Proc. Algorithms and Architectures for Parallel Processing. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 559–570

11 Naruephiphat, W., Ji, Y., Charnsripinyo, C.: 'An area-based approach for node replica detection in wireless sensor networks'. IEEE Proc. 11th Int. Conf. Trust, Security and Privacy in Computing and Communications, 2012, pp. 745–750

12 Znaidi, W., Minier, M., Ubeda, S.: 'Hierarchical node replication attacks detection in wireless sensors networks'. Proc. IEEE 20th Int. Symp. Personal, Indoor and Mobile Radio Communications, 2009, pp. 82–86

13 Zhang, M., Khanapure, V., Chen, S., Xiao, X.: 'Memory efficient protocols for detecting node replication attacks in wireless sensor networks'. Proc. 17th IEEE Int. Conf. Network Protocols, ICNP'09. 2009, pp. 284–293

14 Deng, X.M., Xiong, Y.: 'A new protocol for the detection of node replication attacks in mobile wireless sensor networks', *J. Comput. Sci. Technol.*, 2011, **26**, (4), pp. 732–743

15 Nyhoff, L.: 'Implementing Sets with Bitsets(Chapter 9)'. In: Marcia Hortan ADTs, Data Structures and Problem solving with C++ (Prentice Hall, 2004, 2nd edn.)

16 Schütz, D.: 'Bitstream X-Coder'. Proc. 13th European Conf. Pattern Languages of Programs (EuroPLoP 2008), 2008, pp. 01–08

17 Srisooksai, T., Keamarungsi, K., Lamsrichan, P., Araki, K.: 'Practical data compression in wireless sensor networks: a survey'. *J. Netw. Comput. Appl.*, 2012, **35**, (1), pp. 37–59

18 Kolo, J.G., Shanmugam, S.A., Lim, D.W.G., Ang, L.M., Seng, K.P.: 'An adaptive lossless data compression scheme for wireless sensor networks', *J. Sensors*, 2012, **2012**, pp. 01–20

19 Erratt, N., Liang, Y.: 'Compressed data-stream protocol: an energy-efficient compressed data-stream protocol for wireless sensor networks', *IET Commun.*, 2011, **5**, (18), pp. 2673–2683