Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 269781, 14 pages
http://dx.doi.org/10.1155/2015/269781

Research Article

A Novel OBDD-Based Reliability Evaluation Algorithm for Wireless Sensor Networks on the Multicast Model

Zongshuai Yan,[1,2] Chenhua Nie,[1] Rongsheng Dong,[1] Xi Gao,[1] and Jianming Liu[1]

[1]*Guangxi Key Laboratory of Trusted Software, School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China*
[2]*Zhongxing Telecommunication Equipment Corporation (ZTE Corporation), Shenzhen 518000, China*

Correspondence should be addressed to Chenhua Nie; nch1987smile@163.com

Received 16 July 2014; Revised 4 November 2014; Accepted 5 November 2014

Academic Editor: Xue Jun Li

The two-terminal reliability calculation for wireless sensor networks (WSNs) is a #P-hard problem. The reliability calculation of WSNs on the multicast model provides an even worse combinatorial explosion of node states with respect to the calculation of WSNs on the unicast model; many real WSNs require the multicast model to deliver information. This research first provides a formal definition for the WSN on the multicast model. Next, a symbolic OBDD_Multicast algorithm is proposed to evaluate the reliability of WSNs on the multicast model. Furthermore, our research on OBDD_Multicast construction avoids the problem of invalid expansion, which reduces the number of subnetworks by identifying the redundant paths of two adjacent nodes and *s*-*t* unconnected paths. Experiments show that the OBDD_Multicast both reduces the complexity of the WSN reliability analysis and has a lower running time than Xing's OBDD- (ordered binary decision diagram-) based algorithm.

1. Introduction

Wireless sensor networks (WSNs) have important applications in many prominent areas, such as military, medical industry, and environmental health monitoring [1]. All of these applications require a high level of reliability to operate safely and effectively [2]. If the WSN is unreliable, the tasks may not be completed wasting sensor resources. In order to minimize the cost while maximizing the reliability, first evaluating the reliability is an indispensable step before the successful deployment of WSNs [3].

WSN communication can be divided into two categories: infrastructure communication and application communication [4]. Infrastructure communication relates to transmitting the control and configuration messages from the sink to the sensors, while application communication relates to the transmission of the sensed data from the sensors to the sink. The infrastructure consists of sensors and their current deployment status; thus, infrastructure communication is needed to keep the network functional, ensuring robust operation even in hostile environments, while also optimizing the overall performance. Park and Sivakumar classify data delivery models into three models: sink to a single sensor node (unicast), sink to a group of sensors (multicast), and sink to all sensors (broadcast) [4]. In this paper, in accordance with the reliability problem experienced by WSNs on the multicast model proposed by Shrestha et al. [5], we perform in-depth research on the algorithm used to evaluate the reliability of WSNs on the multicast model, namely, the k-terminal reliability of WSNs.

Computing the two-terminal reliability of WSNs is #P-hard [6]. Calculating the k-terminal reliability for WSNs provides an even worse combinatorial explosion of node states with respect to the calculation of WSNs on the unicast model.

In effectively evaluating the reliability of large-scale WSNs, the process is more difficult using traditional methods, such as factoring, inclusion-exclusion algorithm, and the sum of disjoint products (SDP). Recently, an implicitly symbolic representation and manipulation technique, called a symbolic graph algorithm or symbolic algorithm [7, 8], has emerged in order to combat or ease combinatorial state explosion. Ever since Akers proposed the binary decision diagram

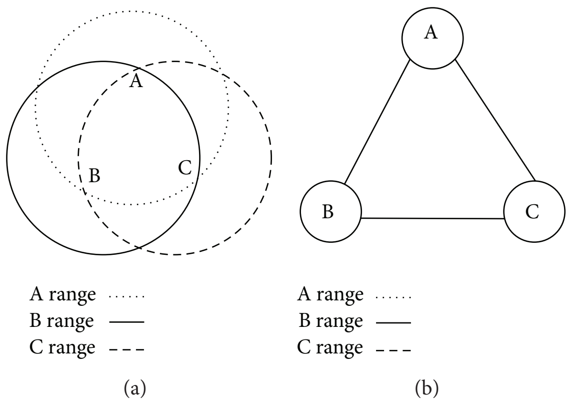


Figure 1: (a) A wireless network; (b) the corresponding graph model.

(BDD) [9] and Bryant popularized the use of BDD by introducing decision variable ordering and simplified rules [10, 11], which make the BDD a canonical form (OBDD) for a Boolean function. Typically, symbolic algorithms based on OBDD are efficient methods in computing the network reliability. Yeh et al. [12] proposed an efficient approach to compute the reliability of an undirected k-terminal network based on 2-terminal reliability functions. The approach constructs the 2-terminal reliability functions of the $(k-1)$ terminal-pairs based on the edge expansion diagram using OBDD and then constructs the k-terminal reliability function by combing these $(k-1)$ 2-terminal reliability functions with OBDD's "AND" operation. Ghasemzadeh [13] presented a method to analyze the k-terminal reliability based on factoring using the "If-Then-Else(ITE)" operation of OBDD. Hardy et al. [14] studied k-terminal network reliability using network decomposition based on edge deletion/contraction with OBDD and recognizes isomorphic subgraphs represented by boundary set in order to reduce redundant computations. However, Yeh, Ghasemzadeh, and Hardy's methods assumed edges were unreliable without considering the case of node failure. This assumption is impractical for WSNs; WSN's sensors have limited power and are usually deployed in inaccessible terrains and even hostile environments. As a result, the sensors are prone to failure as a result of energy depletion or the natural factors such as earthquakes and landslides. We cannot ignore the impact of node fault on the reliability of the WSN. So these studies cannot be directly applied to WSNs.

Shrestha et al. [5] improved upon Yeh's algorithm [12] to analyze the reliability of WSNs on the multicast model under common cause failures, constructed the OBDD for the reliability function, and decreased the redundant computations from isomorphic subnetworks with the aid of a hash table.

In this paper, we focus on the WSN reliability analysis on the multicast model based on OBDD and node expansion. To evaluate the reliability, we have presented a symbolic algorithm named OBDD_Multicast. OBDD_Multicast finds the variable ordering of nodes using a breadth-first search (BFS), decomposes the WSN using node expansion to construct the OBDD using its "AND" and "OR" operations, and then combines these OBDDs using OBDD's "AND" operation to find the OBDD of the reliability function. OBDD_Multicast reduces redundant computations by recognizing the redundant paths of two adjacent nodes and s-t unconnected redundant paths in addition to identifying isomorphic subgraphs. Experiments show that OBDD_Multicast reduces the complexity of WSN reliability analysis by identifying the two types of redundant paths that lead to invalid expansions and has lower running time than Xing's OBDD-based algorithm.

2. Preliminaries

2.1. The Model of a WSN on the Multicast Model. The graph of a WSN should be drawn before constructing a model of the WSN. In this study, we assume all sensors belonging to a WSN were the same. We assumed if a sensor named A is in the communication range of a sensor B, then sensor B is also in the communication range of sensor A, in which case AboElFotoh's method [6] of transforming a WSN into an undirected graph is a better choice. Figure 1 shows the basic idea behind this method, where (a) is a wireless network and (b) is the corresponding graphic model. The figure shows a bidirectional edge exists between each node if they are in range of each other. The edge A-B in Figure 1(b) indicates that there is a communication path from A to B as well as from B to A. In this study, we use a unidirectional edge to represent the bidirectional edge.

Then, the WSN model on the multicast model is abstract to the following model based on an undirected graph obtained using AboElFotoh's method:

$$N = (G, P, s, T), \quad (1)$$

where

(1) $G = (V, E)$ is a graph with the set of nodes $V = (v_1, v_2, \ldots, v_n)$ and $E = (e_1, e_2, \ldots, e_m)$, where $E \subset V \times V$ (n is the number of nodes and m is the number of edges),

(2) $P = (p(v_1), p(v_2), \ldots, p(v_n))$ denotes the set of operational probability of nodes with $p(v_n)$ denoting the operational probability of node v_n and $0 \leq p(v_n) \leq 1$,

(3) s is the sink of the WSN,

(4) $T = (t_1, t_2, \ldots, t_n)$ is the set of target in the WSN on the multicast model, where n is the number of the target and $2 \leq n$.

2.2. The Problem of WSN Reliability. In this paper, we studied the infrastructure communication reliability of a WSN. Below are the assumptions made to evaluate reliability in our analysis:

(1) edges between two nodes are perfect;

(2) nodes failures are statistically independent;

(3) the fault-tolerant clustering protocol is executed to keep the topology.

With the assumption that the links are imperfect, this type of structure would rapidly increase the complexity in order to compute the reliability of WSNs. In this study, we assume that the links are perfect for the convenience of studying them; however, our future work will investigate the evaluation of WSN reliability with imperfect nodes as well as imperfect links. Based on the above assumptions some definitions are presented as follows.

Definition 1 (Reliability). The reliability is the probability that the special elements will accomplish a required task within a time threshold [15]. In this study, the reliability of a WSN is an important measure in evaluating the performance of a WSN.

Definition 2. The reliability of the WSN is the probability that there exists an operational path between the sink node and every node in the set of target nodes.

Definition 3. Given the WSN network $N = (G, P, s, T)$, using random vector $x = (x_0, x_1, \ldots, x_{n-1})$ and $y = (y_0, y_1, \ldots, y_{m-1})$ to, respectively, express the working state of the network's n nodes and m edges, and using (x, y), to express the state of network N, then the reliability of the network N's Boolean function is

$$f_r(x, y) = \begin{cases} 1, & \text{if } N \text{ is at the state } (x, y), \\ & \text{exists an operational path between} \\ & s \text{ and every node in } T \\ 0, & \text{else.} \end{cases} \tag{2}$$

The reliability of a WSN under the multicast model is

$$R_N = \sum_{(x,y)\in\Omega} \Pr\left(f_r(x, y) = 1\right), \tag{3}$$

where Ω is the Cartesian product $(x \times y)$, which is the state space of the network.

2.3. Ordered Binary Decision Diagram. An ordered binary decision diagram (OBDD) [10, 11] provides a compact, canonical, and efficiently manipulative representation for Boolean functions.

An OBDD for a nonconstant Boolean function $f(x_1, x_2, \ldots, x_n)$ is a directed acyclic graph. The characteristics of the OBDD can be summarized as follows:

(1) an OBDD has three types of nodes: root, terminal, and internal. The root node has no parent or is without input arcs, the terminal nodes have no child or out arcs, and the internal nodes have two outgoing arcs;

(2) the root node of an OBDD represents the function f;

(3) there are two terminal nodes "0" and "1," which represent constant Boolean functions 0 and 1;

(4) each nonterminal node has four attributes: f^u, var, low, and high, where f^u is the Boolean function of the vertex u and $f^u = f(x_1, \ldots, u, \ldots, x_n)$; var is the variable labeling the vertex u; low is the 0-node while var $= 0$, the arc connecting the vertex x and low is 0-edge, and low $= f^u|_{u=0}$ as well as $f^u|_{u=0} = f(x_1, \ldots, 0, \ldots, x_n)$; high is the 1-node while var $= 1$, the arc connecting the vertex u and high is 1-edge, and high $= f^u|_{u=1}$ as well as $f^u|_{u=1} = f(x_1, \ldots, 1, \ldots, x_n)$; then, $f^u = u' \cdot f^u|_{u=0} + u \cdot f^u|_{u=1}$;

(5) given the variable ordering $\pi : x_1 < x_2 < \cdots < x_n$, variables in an OBDD are ordered and all the paths in the OBDD keep the same variable ordering.

Given a Boolean function and any assignments to its variables, the function value is determined by tracing a path from the root to a terminal node following the appropriate branch from each node. The branch depends on the variable value of the assignments, and the function value under the assignments is determined by its path's terminal node.

For example, Figure 2 shows the binary and the OBDD for the Boolean function $f = x_1 \cdot x_2 + x_2' \cdot x_3$, in which the variable order is $\pi : x_1 < x_2 < x_3$. It is obvious that the OBDD is a directed acyclic graph and stores the same information more compactly. We trace the path ① → ② → ④ → ⑥ and reach the terminal node 1. Thus, the value of the function $f = x_1 \cdot x_2 + x_2' \cdot x_3$ of variable assignment (0, 0, 1) is 1.

3. A Solution Algorithm for the Problem of WSN Reliability

3.1. Symbolic Formulation of the WSN on the Multicast Model. We convert a WSN on the multicast model, $N = (G, P, s, T)$, to a symbolic OBDD form by encoding the vertices of G with a length-n binary number, where $n = \lceil \log_2 V_{\text{num}} \rceil$ and V_{num} was the number of the vertices. Each encoded vertex of G corresponded to a vector of binary variables $x = (x_0, \ldots, x_{n-1})$. The edge $(v_i, v_j) \in E$ of G can be represented by a binary vector $(x, y) = (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1})$, where $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{n-1})$ are the binary encodings of vertex i and vertex j, respectively. The connecting relation between two nodes can be represented using the following characteristic function:

$$C(x, y) = \begin{cases} 1, & (x, y) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

This function implies the relationship between the nodes and edges of the WSN.

In this study, we assume that the operational probability of all the nodes was 0.9. Then, P in N do not need to be

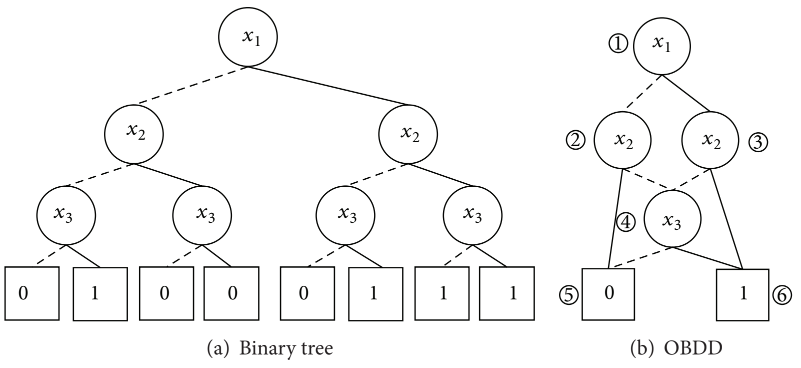


Figure 2: OBDD for Boolean function $f = x_1 \cdot x_2 + x_2' \cdot x_3$.

represented by the Boolean function. Sensor reliability is closely related with the sensing subsystem, the processing subsystem, the communication subsystem, and the power supply subsystem of the WSN. Modeling and evaluating the sensor node reliability will be performed in our future works and will not be discussed in this study.

The source node, namely, the sink node of the WSN, can be represented by the following characteristic function:

$$s(x) = \begin{cases} 1, & x = s, \\ 0, & \text{otherwise}. \end{cases} \tag{5}$$

Similarly, the target nodes can be represented by the following characteristic function:

$$T(y) = \begin{cases} 1, & y \in T, \\ 0, & \text{otherwise}. \end{cases} \tag{6}$$

Thus, a WSN $N = (G, P, s, T)$ can be formulated using $N(C(x, y), s(x), T(y))$. Then the characteristic functions are Boolean functions and can be compactly formulated using OBDDs. For example, OBDDs for the WSN in Figure 3(a) are shown in Figures 3(A), 3(B), and 3(C). Node 1 is the source node, and nodes 3 and 4 are the target nodes. Figure 3(b) gives the encoding of vertices and edges and the adjacency matrix of Figure 3(a), where the encoding below "encoding" is encoding of nodes in Figure 3(a). In the adjacency matrix, x_0 and x_1 are used to sign the starting node of an edge in the network, and y_0 and y_1 are used to sign the subsequent node. Using "0" is to express there is no edge between two nodes and using "1" is to express there is an edge between two nodes. For example, the first line and the second column in adjacency matrix have a value of 1, expressing there is an edge between node 1 and node 2.

3.2. *The Symbolic OBDD_Multicast Algorithm.*

The size of an OBDD is sensitive to variable ordering. It also dictates the performance of subsequent operations and the actual reliability calculation, also known as the order in which the WSN nodes are taken into account when calculating the reliability. However, finding the best ordering is an intractable task. Therefore, heuristics have been used to obtain reasonably good sequences. The BFS heuristic [14] typically outperforms other heuristics; in this study, such a method was used to order the variables.

The key ideas supporting the OBDD_Multicast algorithm are as follows: the variable ordering for the WSN nodes is obtained by BFS, the OBDD of the WSN's reliability on the multicast model is constructed, and the reliability is computed by traversing the OBDD.

3.2.1. *Ordering the Variables in a Breadth-First Manner.*

In this step, a WSN $N(C(x, y), s(x), T(y))$ is visited in a BFS in order to obtain the variable ordering of the WSN. Pseudocode 1 shows the high-level pseudocode used to obtain the variable ordering. This step is started by initializing the OBDD *reached*$(x) = 0$, which is used to store the vertices that are not visited, initializing the vertices of the first layer *Node*$[0](x) = s(x)$, and initializing the OBDD $S(x) = s(x)$. This step then iterates through a sequence of sweeps. A single sweep consists of the following steps.

(1) Store the vertices and edges on each layer by visiting the WSN starting from the source node $s(x)$ with the function Store_Vertices_Edges_on_Layer (Line 6).

(2) Order the source node $s(x)$ using the function Order_Vertices_s (Line 7).

(3) Select all the edges connected with $S(x)$, whose successor nodes are not visited using the function Select_Edges (Line 11).

(4) Order the vertices connected with $S(x)$, that is, not the source node by the function Order_Vertices_Connected (Line 13).

In Store_Vertices_Edges_on_Layer, the WSN $N(C(x, y), s(x), T(y))$ is visited in a BFS manner, starting from a given source vertex $s(x)$, storing the vertices reached for the first time on the ith step of BFS in the OBDD *Node*$[i](x)$, $i = 0, 1, 2, \ldots$, and storing the edges search on each layer in the OBDD $U[i](x, y)$. The OBDD *reached*(x) is used to store the nodes that are visited. Given $S(x), T(x), C(x, y), Node[i](x)$,

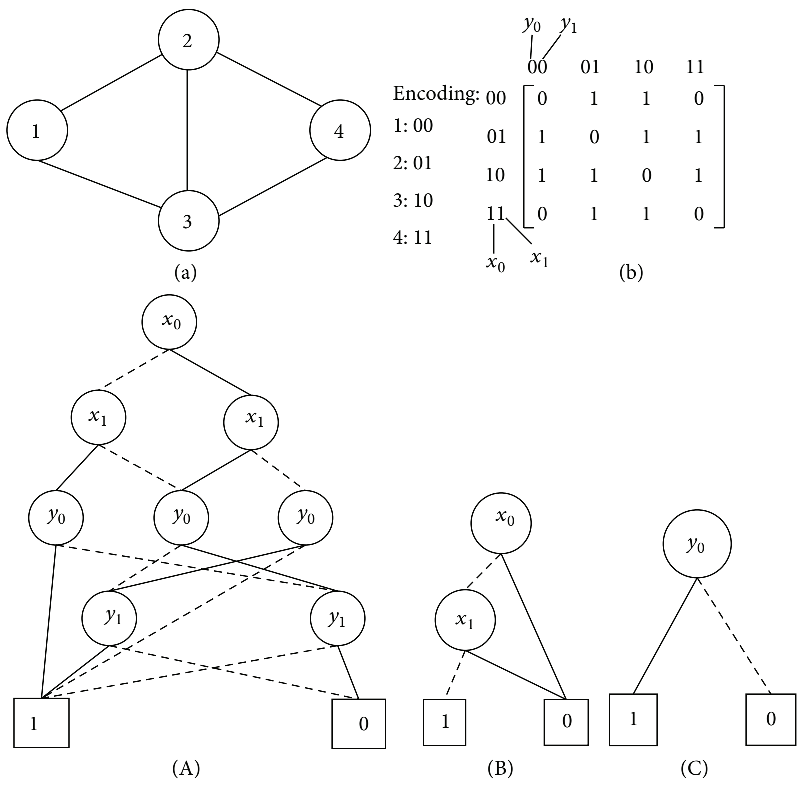


Figure 3: OBDDs for a WSN: (a) a WSN; (b) adjacency matrix of the WSN; (A) OBDD for $C(x, y)$; (B) OBDD for $s(x)$; (C) OBDD for $T(y)$.

```
Get_Variable_Ordering($s(x)$; $T(y)$,$C(x, y)$){
(1)    $reached(x) = 0$;
(2)    $Node[0](x) = s(x)$;
(3)    $S(x) = s(x)$;
(4)    for ($i = 0$;;$i$++){
(5)        if ($S(x) == 0$){
                break;
           }
(6)        ($U[i](x)$, $Node[i+1](x)$, $reached(x)$, $S(x)$) = Store_Vertices_Edges_on_Each_Layer($S(x)$, $T(x)$, $C(x, y)$, $i$, $reached(x)$);
       }
(7)    $nodeOrder[k]$ = Order_Vertices_s($s(x)$);
(8)    $order = 2$;
(9)    for ($j = 1$; $j < i$; $j$++){
(10)       while ($Node[j-1](x)$ != 0) {
(11)           ($Sel(x, y)$, $Node[j-1](x)$) = Select_Edges($U[j-1](x, y)$, $Node[j-1](x)$);
(12)           while ($Sel(x, y)$ != 0)
(13)           ($nodeOrder[k]$, $Sel(x, y)$, $order$) = Order_Vertices_Connected($Sel(x, y)$, $order$);
           }
       }
       return $nodeOrder$;
}
```

Pseudocode 1: Pseudocode for obtaining the variable ordering of the WSN.

$i = 0, 1, 2, \ldots, m$, and $reached(x) = \sum_{i=0}^{i=m} Node[i](x)$, the vertices and edges on each layer are stored as follows:

$$U[i](x, y) = S(x) \cdot C(x, y) \cdot \overline{reached(y)},$$
$$Node[i+1](x) = \exists y U[i](y, x),$$
$$S(x) = Node[i+1](x) \cdot \overline{T(x)},$$
$$reached(x) = reached(x) + Node[i+1](x), \tag{7}$$

where $S(x) \cdot C(x, y) \cdot \overline{reached(y)}$ means that the edges are selected from $C(x, y)$ and the start node of the selected edges is $S(x)$; meanwhile, the successor nodes of the edges selected have not been visited. $\exists y U[i](y, x)$ indicates abstracting the successor nodes of the edges $U[i](y, x)$, and $Node[i+1](x) \cdot \overline{T(x)}$ indicates deleting the target nodes $T(x)$ from $Node[i+1](x)$. After Store_Vertices_Edges_on_Layer, every vertex in every set $Node[i+1](x)$ as well as every edge in every set $U[i](x, y)$ is reachable by a path from the source.

In Order_Vertices_s, the variable labeling the OBDD $s(x)$ is obtained, which is used as the subscript of the array *nodeOrder*; the number 1 is then assigned to this array *nodeOrder* with the subscript; that is,

$$\begin{aligned} k &= index\,(s\,(x))\,, \\ nodeOrder\,[k] &= 1, \end{aligned} \tag{8}$$

where the function *index* is to obtain the mark number of the OBDD $s(x)$.

After Order_Vertices_s, the vertex $s(x)$ has been ordered, and its variable number becomes 1.

In Select_Edges, in order to obtain the variable ordering of the vertices on level j, namely, $Node[j](x)$, one vertex is selected from the sets $Node[j-1](x)$. The edges of the sets $U[j-1](x, y)$, from which all edges connected with the vertex selected from $Node[j-1](x)$, are selected and stored in $Sel(x, y)$; that is,

$$\begin{aligned} oneN\,(x) &= PickOneTerm\,(Node\,[j-1]\,(x))\,, \\ Node\,[j-1]\,(x) &= Node\,[j-1]\,(x) - oneN\,(x)\,, \\ Sel\,(x, y) &= oneN\,(x) \cdot U\,[j-1]\,(x, y)\,, \end{aligned} \tag{9}$$

where the function $PickOneTerm(f)$ represents the selection of one minimum term from the function f.

After Select_Edges, all edges that connect with the vertex selected from $Node[j-1](x)$ are selected, as well as those whose successor nodes have not been visited. Then, using the function Order_Vertices_Connected, we can obtain the variable ordering of the vertices from the sets $Node[j-1](x)$ to which the edges selected by Select_Edges are connected.

The key of Order_Vertices_Connected is to select one vertex and order this vertex. A priority function is used in order to solve the node-matching problem for each layer of the WSN. The priority function is as follows: $\Pi(x, y, z) : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$. The first argument is the bias, and the other two arguments are the nodes to be compared. For each choice of x, Π returns 1 if the second argument precedes the third; otherwise, 0 is returned. In this study, the first priority function $\Pi(x, y, z) = \|y-x\| < \|z-x\|$ is called the relative proximity function and returns the result of testing $\|y-x\| < \|z-x\|$, where $\|x-y\|$ is defined as follows:

$$\|x-y\| = \sum_{i=0}^{n-1} |x_i - y_i|\, 2^{n-i-1}, \tag{10}$$

n is the number of elements in the vectors x, y, and z.

Order_Vertices_Connected starts from a given edge $Sel(x, y)$ and a given order number, namely, *order*. The order number of the vertex selected by the prior function is stored in the array *nodeOrder*. Then, the vertices are computed as follows:

$$\begin{aligned} E\,(x, y) &= Sel\,(x, y) \cdot \overline{\exists z\,(Sel\,(x, z) \cdot \Pi\,(x, z, y))}, \\ Sel\,(x, y) &= Sel\,(x, y) - E\,(x, y)\,, \\ one\,(x) &= \exists y E\,(y, x)\,, \\ k &= index\,(one\,(x))\,, \\ nodeOrder\,[k] &= order, \\ order &= order + 1, \end{aligned} \tag{11}$$

where $E(x, y)$ is the subgraph of $Sel(x, y)$ and each vertex of $E(x, y)$ has most one outgoing edge. The successor node of the edge $E(x, y)$ is obtained using the function $\exists y E(y, x)$.

If $Node[j-1](x) = 0$ is satisfied in some layers, all nodes in $Node[j](x)$ are ordered, and a new loop j is started.

3.2.2. Construct the OBDD of the Reliability Function. This step uses the variable ordering which has been obtained using the function Get_Variable_Ordering. Pseudocode 2 shows the high-level pseudocode of Construct_OBDD_Reliability. The Construct_OBDD_Reliability is begun by initializing the variable $node\_s(x) = s(x)$ and implemented through a series of recursive operations. Briefly, one recursion consists of the following steps.

(1) Construct the OBDD of the current subnetwork when the current source vertex $s(x)$ belongs to the sets of targets, namely, $T(x)$, by procedure Construct_OBDD_t (Line 2).

(2) Remove the redundant vertices to avoid the inefficient manipulation (Line 4). A vertex other than the source and target vertex is named a redundant vertex if only one edge is connected to it.

(3) Select all vertices connected to the variable $node\_s(x)$, belonging to the set of target vertices (Line 7).

(4) Execute node expansion, and obtain subnetwork $G_{s\rightarrow v}$ (Line 8). To perform this expansion, the current source is merged into its adjacent node, which belongs to the sets $T(x)$.

(5) Construct the OBDD of $G_{s\rightarrow v}$ obtained using Node_Expansion_t, and return this OBDD (Line 9).

(6) Execute Boolean operations on OBDD to construct the OBDD of the current network by Construct_OBDD_Subpath (Line 10).

(7) Remove the redundant paths of two adjacent vertices by Remove_Redundant_Path_Two_Adjacent_Vertices (Lines 11 and 16). This step can avoid invalid extensions.

(8) Select all vertices connected to the variable $node\_s(x)$ by Select_Vertices_Connected_s (Line 12). This step is similar to the procedure Select_Vertices_in_T_Connected_s (Line 7). The only difference is that vertices selected by Select_Vertices_Connected_s do not belong to the set $T(x)$.

```
Construct_OBDD_Reliability($s(x)$, $T(y)$, $C(x, y)$, $nodeOrder$){
(1)   $node\_s(x) = s(x)$;
      if ($node\_s(x) \in T(x)$){
(2)       $bdd\_result[i]$ = Construct_OBDD_t($node\_s(x)$, $nodeOrder$)
(3)       return $bdd\_result[i]$;
      }
(4)   ($C(x, y)$) = Remove_Redundant_Vertices($node\_s(x)$, $C(x, y)$, $T(x)$);
      if (there is the isomorphic sub-network in the sub-network hash table)
(5)       return the OBDD of the sub-network;
(6)   $N(x, y) = C(x, y)$;
      if (there is at least one vertex connected to $node\_s(x)$, which is in $T(x)$){
(7)       $T\_v(x)$ = Select_Vertices_in_T_Connected_s($node\_s(x)$, $C(x, y)$, $T(y)$);
          do{
(8)           ($ns(x)$, $C(x, y)$, $T\_v(x)$, $k$) = Node_Expansion_t($node\_s(x)$, $C(x, y)$, $T\_v(x)$);
(9)           $result[i]$ = Construct_OBDD_Reliability($ns(x)$, $T(y)$, $C(x, y)$, $nodeOrder$);
(10)          $bdd\_result[i]$ = Construct_OBDD_Subpath($result[i]$, $k$);
(11)          $N(x, y)$ = Remove_Redundant_Path_Two_Adjacent_Vertices($N(x, y)$, $ns(x)$);
          }while ($T\_v(x)$ != 0)
      }
      if (not all vertices connected to $s(x)$ have been expanded){
(12)      ($R(x)$, $nE(x, y)$) = Select_Vertices_Connected_s($s(x)$, $N(x, y)$);
          do{
(13)          ($node\_s(x)$, $C_1(x, y)$, $R(x)$, $k$) = Node_Expansion($s(x)$, $N(x, y)$, $R(x)$, $nE(x, y)$);
(14)          $result[i]$ = Construct_OBDD_Reliability($node\_s(x)$, $T(y)$, $N(x, y)$, $nodeOrder$);
(15)          $bdd\_result[i]$ = Construct_OBDD_Subpath($result[i]$, $k$);
(16)          $N(x, y)$ = Remove_Redundant_Path_Two_Adjacent_Vertices($N(x, y)$, $node\_s(x)$);
          }while ($R(x)$ != 0)
      }
(17)  result $bdd\_result[i]$;
}
```

PSEUDOCODE 2: Pseudocode for constructing the OBDD of the reliability function.

(9) Execute node expansion, and obtain subnetwork $G_{s\rightarrow v'}$ (Line 13). This step is similar to the procedure Node_Expansion_t (Line 8). However, we merge the source of the current network into its adjacent node v', which does not belong to the set $T(x)$.

(10) Construct the OBDD of the subnetwork $G_{s\rightarrow v'}$ obtained using Node_Expansion and return this OBDD (Line 14). This step is similar to the corresponding steps (Line 9). The only difference is that the incoming arguments, namely, the variable $node\_s(x)$ and the subnetwork $C(x, y)$, are different.

(11) Execute Boolean operations on OBDD to construct the OBDD of current network (Line 15). This step is similar to the procedure (Line 10).

In Construct_OBDD_t, the variable number of $node\_s(x)$ is searched from the array *nodeOrder*, and then the OBDD of this variable number is obtained; that is,

$$i = index\,(node\_s\,(x))\,,$$
$$order = nodeOrder\,[i]\,, \tag{12}$$
$$bdd\_result\,[i] = obdd\,(order)\,.$$

The function of Remove_Redundant_Vertices is to remove the redundant nodes. Pseudocode 3 shows the high-level pseudocode in the function. The function is started by selecting the vertices that are neither the source node of the current subnetwork nor the target nodes, and iterates through a loop program to determine whether the node is redundant and whether to delete the redundant node. Remove_Redundant_Vertices consists of the following main steps.

(1) Select the vertices that are neither the source node of the current subnetwork nor the target nodes by Select_Vertices_not_s_T (Line 3.1 in Pseudocode 3). In Select_Vertices_not_s_T, the vertices, which are neither the source node of the current subnetwork nor the target nodes, are computed as follows:

$$V\,(x) = \overline{node\_s(x)} \cdot (\exists y C\,(x, y)) \cdot \overline{T\,(x)}. \tag{13}$$

(2) Determine whether the vertices selected by Select_Vertices_not_s_T are the redundant vertices (Line 3.2 in Pseudocode 3). If $Edge(x, y)$ is 0, the $V\_s(x)$ is the redundant vertex. Then the vertex is removed using the function Remove_Vertices (Line 3.3 in Pseudocode 3). The operations are iterated by the following equations until the variable $V(x)$

```
Remode_Redundant_Vertices($node\_s(x)$, $C(x, y)$, $T(x)$){
(3.1)      $V(x)$ = Select_Vertices_not_s_T($node\_s(x)$, $C(x, y)$, $T(x)$);
           do{
(3.2)          ($V\_s(x)$, $Edge(x, y)$) = Determin_Redundant_Vertices($V(x)$, $C(x, y)$);
               if ($Edge(x, y)$ == 0){ // $V\_s(x)$ is the redundant vertex if $Edge(x, y)$ == 0.
(3.3)              $C(x, y)$ = Remove_Vertices($V\_s(x)$, $C(x, y)$);
           }while ($V(x)$ != 0)
           return $C(x, y)$;
}
```

Pseudocode 3: Pseudocode for removing the redundant vertices of the WSN.

reduces to 0.In Determine_Redundant_Vertices, the equations will be implemented:

$$\begin{aligned} V\_s(x) &= PickOneTerm(V(x)), \\ Edge(x,y) &= V\_s(x)\cdot C(x,y), \\ Etemp(x,y) &= PickOneTerm(Edge(x,y)), \\ Edge(x,y) &= Edge(x,y) - Etemp(x,y), \\ V(x) &= V(x) - V\_s(x). \end{aligned} \tag{14}$$

In Remove_Vertices, the equations will be implemented:

$$\begin{aligned} E\_v(x,y) &= V\_s(x)\cdot C(x,y), \\ C(x,y) &= C(x,y) - E\_v(x,y). \end{aligned} \tag{15}$$

After Remove_Redundant_Vertices, the redundant vertices have been deleted, and the simplified network $C(x, y)$ is obtained.

In Select_Vertices_in_T_Connected_s, all vertices, namely, $T\_v(x)$, which are connected to the variable $node\_s(x)$ and belong to the set of target vertex $T(x)$, are obtained through the following equations:

$$\begin{aligned} nE(x,y) &= node\_s(x)\cdot C(x,y)\cdot T(y), \\ T\_v(x) &= \exists y nE(y,x). \end{aligned} \tag{16}$$

Then the node expansion is executed, and the subnetwork $G_{s\to v}$ of the current network is obtained using the procedure Node_Expansion_t. First, Node_Expansion_t computes the mark number of the variable $node\_s(x)$. Second, all edges connected to $node\_s(x)$ are searched. Then these edges are deleted to obtain the subnetwork $G_{s\to v}$. Finally, the source node of $G_{s\to v}$ is selected from the set $T\_v(x)$ obtained by Select_Vertices_in_T_Connected_s. The subnetwork $G_{s\to v}$ is computed as follows:

$$\begin{aligned} k &= index(node\_s(x)), \\ E\_s(x,y) &= node\_s(x)\cdot C(x,y), \\ C(x,y) &= C(x,y) - E\_s(x,y), \\ ns(x) &= PickOneTerm(T\_v(x)), \\ T\_v(x) &= T\_v(x) - ns(x). \end{aligned} \tag{17}$$

After Node_Expansion_t, we can obtain the new subnetwork $G_{s\to v}$, namely, $C(x, y)$, whose source node is $node\_s(x)$. We then construct the OBDD of subnetwork $G_{s\to v}$ using the recursive function Construct_OBDD_Reliability. What this function returns is stored in the OBDD $result[i]$.

In Construct_OBDD_Subpath, the OBDD of the current subnetwork is computed, starting with the OBDD $result[i]$ and the mark number k, obtaining variable number of k, combining the OBDDs of the nodes in the same path by the "AND" operation of OBDD, and combining the OBDDs of the nodes in different paths by the "OR" operation of OBDD; that is,

$$\begin{aligned} order &= nodeOrder[k], \\ bddTemp[i] &= bdd\_and(obdd(order), bdd\_result[i]), \\ bdd\_result[i] &= bdd\_or(bdd\_result[i], bddTemp[i]), \end{aligned} \tag{18}$$

where $order$ is the variable number of k, bdd_and is the "AND" operation of OBDD, and bdd_or is the "OR" operation of OBDD.

In Remove_Redundant_Path_Two_Adjacent_Vertices, the redundant paths between $s(x)$ and $node\_s(x)$ are deleted as follows:

$$\begin{aligned} E_1(x,y) &= ns(x)\cdot N(x,y), \\ E_2(x,y) &= N(x,y)\cdot ns(y), \\ N(x,y) &= N(x,y) - E_1(x,y) - E_2(x,y). \end{aligned} \tag{19}$$

In Select_Vertices_Connected_s, all edges connected to the source vertex $s(x)$ are computed, and then the successor nodes of these edges are obtained; that is,

$$\begin{aligned} nE(x,y) &= s(x)\cdot N(x,y), \\ R(x) &= \exists y nE(y,x), \end{aligned} \tag{20}$$

where $nE(x, y)$ are the edges connected to the vertex $s(x)$ and $R(x)$ are the successor nodes of these edges.

After Select_Vertices_Connected_s, the operations of node expansion are then executed using Node_Expansion. We obtain the set of vertices, namely, $R(x)$. To execute this expansion, one vertex v' is selected from $R(x)$, and the source

```
Construct_OBDD_Reliability_WSN($bdd\_result$){
    $relBDD = 0$;
    $i = 0$;
    do{
(4.1)       $relBDD =$ bdd_and($relBDD$, $bdd\_result[i]$);
    }while ($i > n$); // $n$ is the number of the target vertices of the WSN
    return $relBDD$;
}
```

Pseudocode 4: Pseudocode for constructing the OBDDs of paths from the original source vertex to every target vertex.

vertex is then merged into the vertex v'; next, all edges incident to the source node are deleted; that is,

$$\begin{aligned} i &= index\left(s\left(x\right)\right), \\ node\_s\left(x\right) &= PickOneTerm\left(R\left(x\right)\right), \\ R\left(x\right) &= R\left(x\right) - node\_s\left(x\right), \\ C_1\left(x, y\right) &= C\left(x, y\right) - nE\left(x, y\right). \end{aligned} \tag{21}$$

After Node_Expansion, we obtain the variable number k of source node $s(x)$, the new source vertex $node\_s(x)$, and the new subnetwork $G_{s\to v'}$, namely, $C_1(x, y)$. Then, we construct the OBDD of subnetwork $G_{s\to v'}$ using the recursive function Construct_OBDD_Reliability. What this function returned is then stored in the OBDD $result[i]$. We have the same operations of Construct_OBDD_Subpath to construct the OBDD of paths; the operations in Line 15 are the same as in Line 10.

Next, we delete the redundant paths between two adjacent vertices. This work is similar to the operations in Line 11.

When $T\_v(x)$ is 0, we are able to obtain the OBDD of the current subnetwork, namely, $bdd\_result[i]$. Finally, the array bdd_result, returned using the Construct_OBDD_Reliability function, stores the OBDD of each path from source vertex of original network to every target node.

The OBDD (the $relBDD$ in Pseudocode 4) of the reliability of the WSN on the multicast model is obtained by applying the "AND" operation of OBDD to the array bdd_result. Pseudocode 4 shows the high-level pseudocode of these operations.

3.2.3. Compute the Reliability of the WSN on the Multicast Model. After the OBDD for the WSN's reliability on the multicast model is obtained, a breadth-first is used to visit the OBDD and to compute the reliability. The reliability is computed starting from the root node, and the probability value of the parent node is multiplied with the probability of the node itself and assigned to each child node. The probability of the root node itself is initialized to 1. We use the following equation to compute the reliability:

$$P\left(F\right) = \begin{cases} 1, & \text{if } F \text{ is the root node}, \\ P\left(F\right) + \left(1 - p\right) \times P\left(node\right), & \text{if } F \text{ is 0-node of } node, \\ P\left(F\right) + p \times P\left(node\right), & \text{if } F \text{ is 1-node of } node, \end{cases} \tag{22}$$

where p is the operational probability of the vertex of the WSN and $P(node)$ represents the probability of the node "*node*" itself. We traverse each node only once so the complexity was $O(n)$, where n equals the number of the nodes. Pseudocode 5 shows the high-level pseudocode of the algorithm.

3.3. An Illustrated Example. To illustrate the basic principles of the proposed approach for the WSN reliability analysis, a benchmark network is considered. Figure 4 shows the graph of the sample WSN system. The sink vertex is node 4, and the target vertices are nodes 7 and 8.

The process of constructing the OBDD of the reliability function by OBDD_Multicast is shown as follows.

Step 1. The WSN is formulated using $N(C(x, y), s(x), T(y))$, represented by the OBDDs with binary encoding of the vertices. Figure 5 shows the OBDDs.

Step 2. The function of Store_Vertices_Edges_on_Layer (Line 6 in Pseudocode 1) is implemented to store the vertices and edges on each layer. Then, we are able to obtain $Node[0](x) = \{4\}$, $Node[1](x) = \{2, 5, 6\}$, $Node[2](x) = \{1, 3, 7, 8\}$, $U[0](x, y) = \{e_4, e_6, e_7\}$, $U[1](x, y) = \{e_1, e_3, e_5, e_8, e_9, e_{10}\}$, and $U[2](x, y) = \{e_2, e_{11}\}$.

Step 3. To order source node 4, the function Order_Vertices_s (Line 7 in Pseudocode 1) is used. Then, the variable number of node 4 is 1.

Step 4. To select all the edges connected with node 4, whose successor nodes have not been visited, the function Select_Edges (Line 11 in Pseudocode 1) is used. We are then able to obtain the edges, which are $\{e_4, e_6, e_7\}$.

```
Compute_Reliability_WSN(OBDD relBDD){
    Queue.insert(relBDD); // Queue is a queue which is used to store the visited nodes of the relBDD
    P(relBDD) = 1;
    while (Queue is not empty){
        node = Queue.getHead();
        $F_0$ = low(node); // $F_0$ is the 0-node of node
        if ($F_0$ is the terminal node which is 1)
            reliability = reliability + $(1 - p) \times P(node)$; // p is the operational probability of node
        else
            $P(F_0) = P(F_0) + (1 - p) \times P(node)$;
        Queue.insert($F_0$);
        $F_1$ = high(node); // $F_1$ is the 1-node of node
        if ($F_1$ is the terminal node which is 1)
            reliability = reliability + $p \times P(node)$;
        else
            $P(F_1) = P(F_1) + p \times P(\text{node})$;
        Queue.insert($F_1$);
    }
    return reliability;
}
```

Pseudocode 5: Pseudocode for computing the reliability with the OBDD obtained by Construct_OBDD_Reliability_WSN.

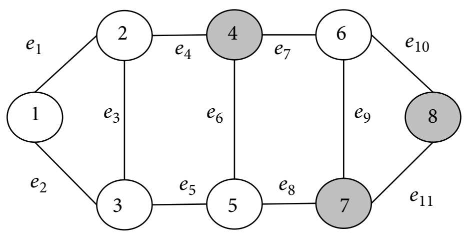


Figure 4: A benchmark network.

Step 5. For every edge in $\{e_4, e_6, e_7\}$, use the function Order_Vertices_Connected (Line 13 in Pseudocode 1) to obtain their successor nodes and order these nodes. The successor nodes are nodes 2, 5, and 6; the variable numbers of these nodes are 2, 3, and 4, respectively.

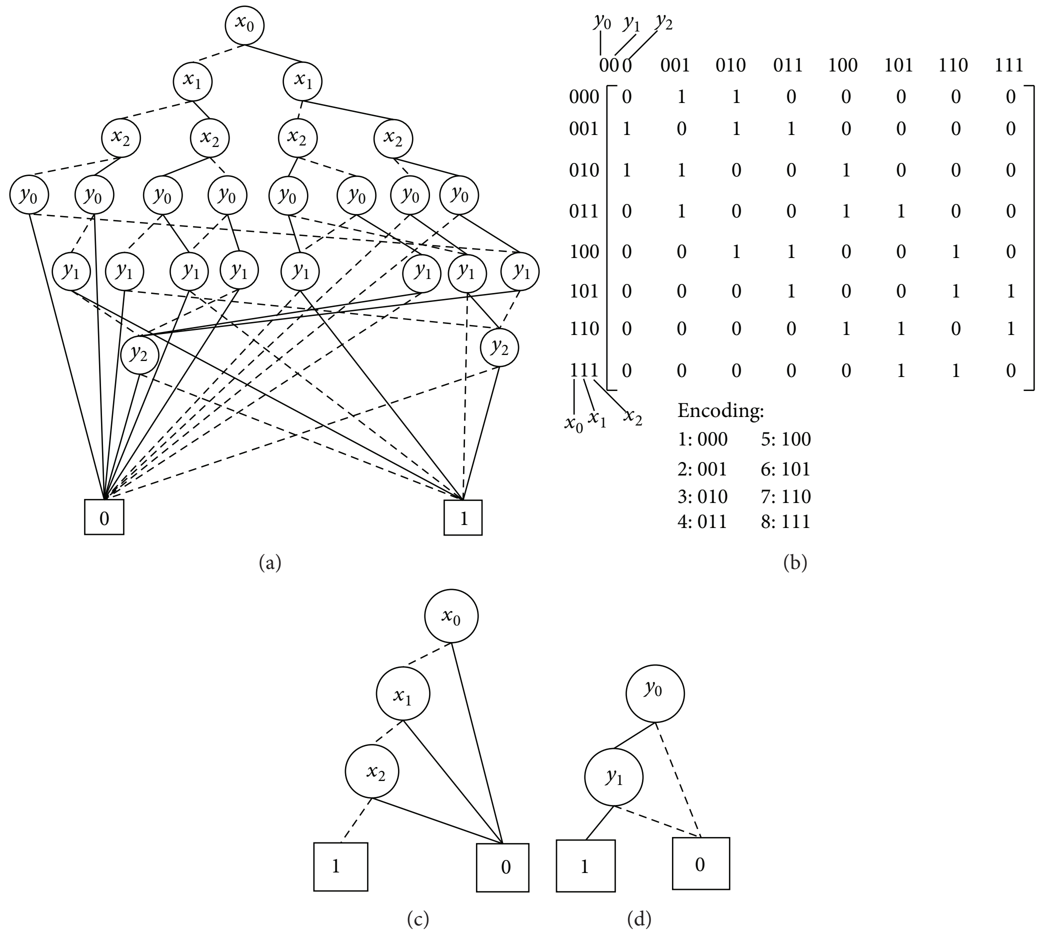
Step 6. For every node in $Node[1](x) = \{2, 5, 6\}$, select one vertex from this set and obtain all edges, whose successor nodes are not visited, connected with the vertex. First, select node 2, and $\{e_1, e_3\}$ is obtained using the Select_Edges (Line 11 in Pseudocode 1). Next, obtain the successor nodes $\{1, 3\}$, and order these nodes using Order_Vertices_Connected (Line 13 in Pseudocode 1). Then, the variable numbers of nodes 1 and 3 are 5 and 6, respectively. Second, node 5 is selected, and e_8 is obtained using Select_Edges. Next, obtain successor node 7, and order these nodes using Order_Vertices_Connected (Line 13 in Pseudocode 1). Then, the variable number of node 7 is 7. Finally, select node 6, and e_{10} is obtained using Select_Edges. Then, obtain successor node 8, and order these nodes by Order_Vertices_Connected (Line 14 in Pseudocode 1). The variable number of node 8 is 8. Finally, the variable ordering is obtained; that is, $\Pi 4 < 2 < 5 < 6 < 1 < 3 < 7 < 8$.

Step 7. Figure 6 shows the network decomposition of the benchmark network when the OBDD of the reliability function is constructed. The network is then decomposed starting from node 4 by node expansion. Apply "AND" of OBDD to the nodes on the same path from source vertex to the same target node, and apply "OR" of OBDD to the nodes on different paths from source vertex to the same target node. From Figure 6, we can see that there are two paths from node 4 to node 7, namely, $4 \to 6 \to 7$ and $4 \to 5 \to 7$, and there is one path from node 4 to node 8, namely, $4 \to 6 \to 8$. The operations for these nodes on these paths are shown in Figure 7.

Step 8. To obtain the OBDD of the reliability function, "AND" is applied for the OBDDs in (c) and (d) of Figure 7. The result OBDD is shown in Figure 8.

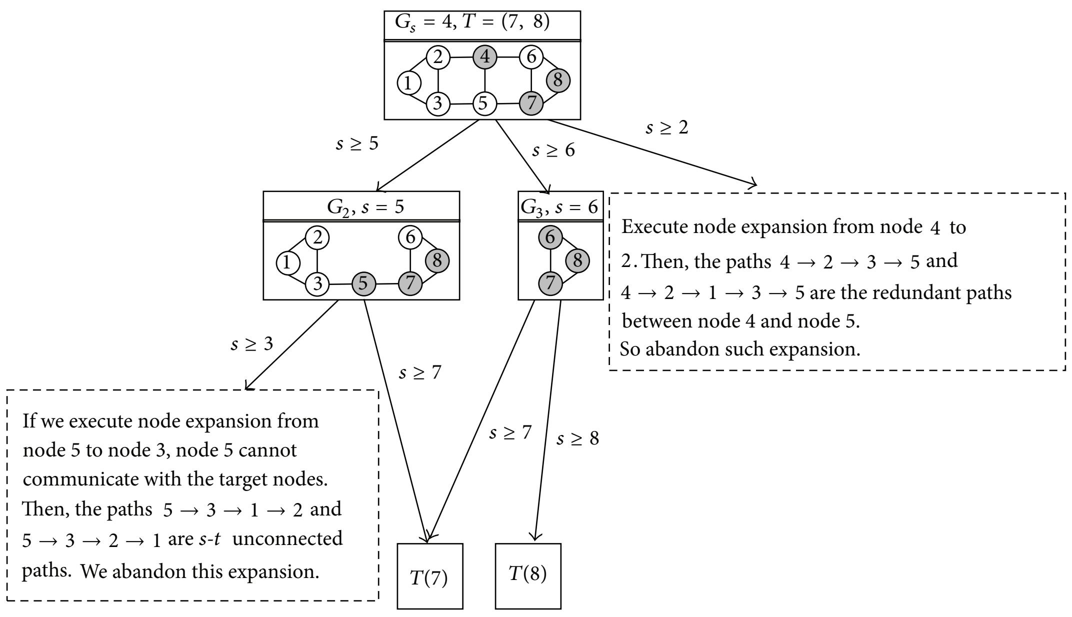
We can then compute the reliability with the result OBDD. For convenience of calculations, we have assumed that the operational probability of all sensors is 0.9. The reliability of sensors is closely related with the sensing subsystem, the processing subsystem, the communication subsystem, and the power supply subsystem of the WSN. Modeling and evaluating the reliability of the sensor node are the work we will explore in our future and will not be discussed in this paper. The process of computing the reliability is attached to Figure 8. From this figure, we can see that reliability = 0.6561. To obtain the reliability of the WSN, each node of the OBDD is traversed only once. The complexity is $O(n)$, where n is the number of the nodes of the OBDD.

Figure 6 shows the network decomposition by OBDD_Multicast and Figure 9 shows the network decomposition using Xing's method. By comparing the two figures, we can see that OBDD_Multicast avoids the invalid expansion, which reduces the number of subnetworks by identifying the



$$\begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
001 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
010 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
011 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
100 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
101 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
110 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
111 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}$$

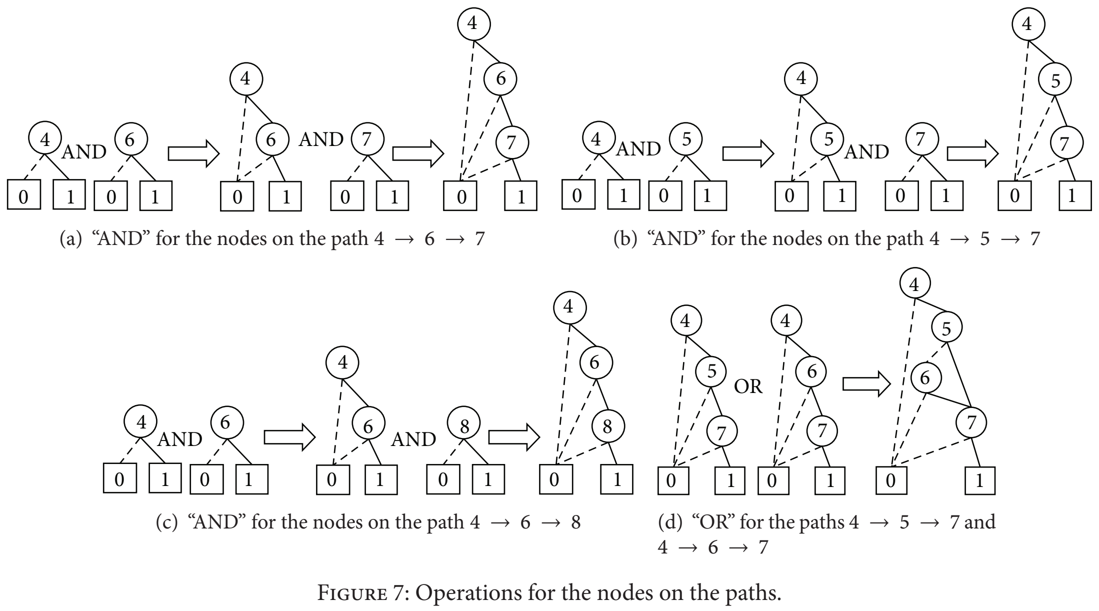
FIGURE 5: OBDDs for the benchmark network: (a) OBDD of $C(x, y)$ and the encoding of the vertices; (b) adjacency matrix of the benchmark network; (c) OBDD of $s(x)$; (d) OBDD of $T(y)$.



Execute node expansion from node 4 to 2. Then, the paths $4 \rightarrow 2 \rightarrow 3 \rightarrow 5$ and $4 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 5$ are the redundant paths between node 4 and node 5. So abandon such expansion.

If we execute node expansion from node 5 to node 3, node 5 cannot communicate with the target nodes. Then, the paths $5 \rightarrow 3 \rightarrow 1 \rightarrow 2$ and $5 \rightarrow 3 \rightarrow 2 \rightarrow 1$ are s-t unconnected paths. We abandon this expansion.

FIGURE 6: Network decomposition using OBDD_Multicast.



(a) "AND" for the nodes on the path 4 $\rightarrow$ 6 $\rightarrow$ 7

(b) "AND" for the nodes on the path 4 $\rightarrow$ 5 $\rightarrow$ 7

(c) "AND" for the nodes on the path 4 $\rightarrow$ 6 $\rightarrow$ 8

(d) "OR" for the paths 4 $\rightarrow$ 5 $\rightarrow$ 7 and 4 $\rightarrow$ 6 $\rightarrow$ 7

Figure 7: Operations for the nodes on the paths.

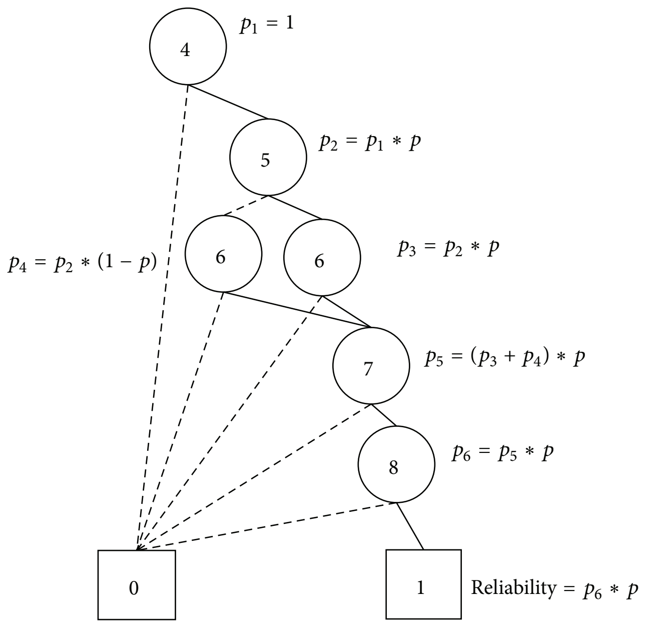


Figure 8: The OBDD of the reliability function and the process of computing reliability.

redundant paths of two adjacent nodes and *s*-*t* unconnected paths.

4. Experimental Results

The symbolic algorithm proposed in this paper is implemented in Windows XP and the software package CUDD [16], developed by the University of Colorado. CPU time is in seconds at 2.80 GHz with 3 GB of memory.

In the experiments, the OBDD_Multicast is compared with Xing's OBDD-based algorithm. We chose $3 \times n$ networks (there are three lines, with n nodes in each line) as benchmark networks. Figure 10 shows this type of network. Gray nodes indicate the source and target nodes, specifically where node 3 is the source node and the other gray nodes are target nodes. The results are shown in Table 1, where "reliability" in column 3 denotes the reliability of these benchmark networks, which is obtained using Xing's algorithm and OBDD_Multicast, "XOBDD" denotes the running time for Xing's algorithm, and "OBDD_Multicast" denotes the running time for OBDD_Multicast. "0" in Table 1 denotes that the running time was lower than 0.000001 s, and "—" denotes that the running time was longer than half an hour. The reliability computed using Xing's algorithm and OBDD_Multicast is the same. The experiments indicate the fact that OBDD_Multicast is correct; furthermore, it can be observed that the symbolic OBDD_Multicast outperformed Xing's algorithm. However, on smaller scale networks (such as 3×3 and 3×4 networks in Figure 10), the running times for Xing's algorithm and OBDD_Multicast were the same. However, with the increase in the scale of the networks, the disparity of the running times likewise grew. This is especially true for a 3×9 network; the time cost of Xing's algorithm exceeds half an hour, while the OBDD_Multicast's cost is only 113.14 s. OBDD_Multicast avoids the invalid expansion, which would have reduced the number of subnetworks

Table 1: The reliability of the WSNs and their computing time.

$3 \times n$ WSN	Reliability	XOBDD (s)	OBDD_Multicast (s)
3×3	0.633537	0	0
3×4	0.634993	0	0
3×5	0.628447	0.015	0
3×6	0.624206	0.031	0.015
3×7	0.61987	0.266	0.172
3×8	0.615535	4.109	3.671
3×9	0.612250	—	113.140

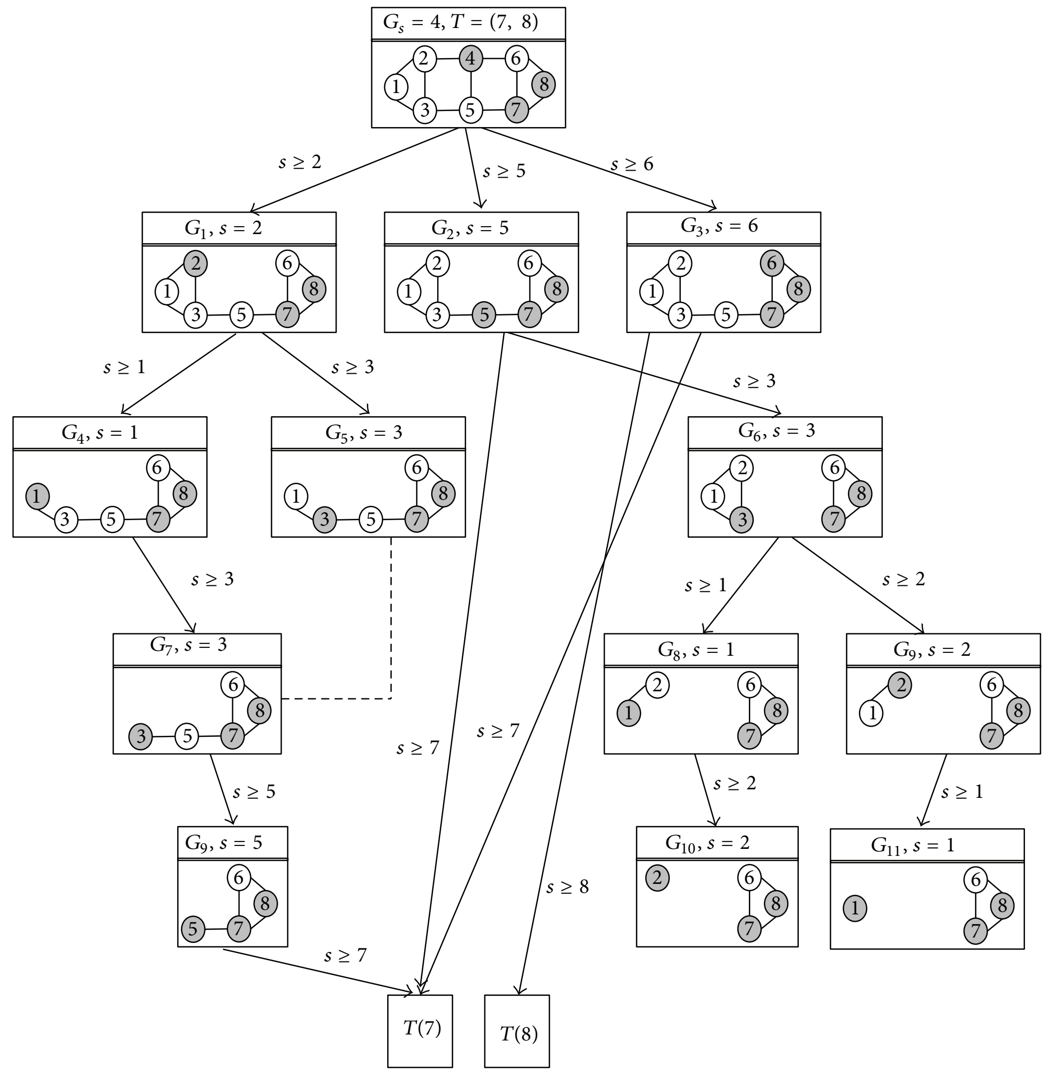


FIGURE 9: Network decomposition by Xing's method.

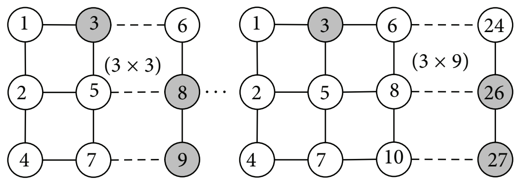


FIGURE 10: $3 \times n$ WSN.

by identifying the redundant paths of two adjacent nodes and *s-t* unconnected paths. This comparison shows that OBDD_Multicast reduces the complexity of WSN reliability analysis and had a lower running time than Xing's OBDD-based algorithm, and OBDD_Multicast could efficiently evaluate the reliability of the WSN.

5. Conclusions

In this paper, an efficient OBDD-based algorithm, OBDD_Multicast, is proposed to evaluate the reliability of the WSN on the multicast model. The OBDD_Multicast's advantages lie in that the OBDD_Multicast avoids invalid expansion by reducing the number of subnetworks through identifying the redundant paths of two adjacent nodes and *s-t* unconnected paths. Experiments shows OBDD_Multicast has a lower running time compared to Xing's OBDD-based algorithm; and OBDD_Multicast was also efficient for the reliability evaluation in larger scale WSNs. In this paper, we assume the operational probability of sensors was 0.9 and focus on the reliability of the WSN at the system-level. Modeling and evaluating the reliability of the sensor, namely, evaluating the WSN at the component-level, are part of our future work, which also includes evaluating the reliability of WSNs with imperfect nodes as well as imperfect links.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (no. 61363070 and no. 61262074) and Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology) (no. kx201401).

References

[1] K. Sharma and M. K. Ghose, "Wireless sensor networks: an overview on its security threats," *International Journal of Computer Applications*, no. 1, pp. 42–45, 2010.

[2] C. Wang, L. Xing, V. M. Vokkarane, and Y. L. Sun, "A phased-mission framework for communication reliability in WSN," in *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS '14)*, pp. 1–7, Colorado Springs, Colo, USA, January 2014.

[3] "National Science Foundation Where Discoveries Begin," http://www.nsf.gov/awardsearch/showAward?AWD_ID=1112947&HistoricalAwards=false.

[4] S. J. Park and R. Sivakumar, "Sink-to-sensors reliability in sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 27–28, 2003.

[5] A. Shrestha, L. Xing, Y. Sun, and V. M. Vokkarane, "Infrastructure communication reliability of wireless sensor networks considering common-cause failures," *International Journal of Performability Engineering*, vol. 8, no. 2, pp. 141–150, 2012.

[6] H. M. F. AboElFotoh, S. S. Iyengar, and K. Chakrabarty, "Computing reliability and message delay for cooperative wireless distributed sensor networks subject to random failures," *IEEE Transactions on Reliability*, vol. 54, no. 1, pp. 145–155, 2005.

[7] T. Gu and Z. Xu, "The symbolic algorithms for maximum flow in networks," *Computers and Operations Research*, vol. 34, no. 3, pp. 799–816, 2007.

[8] T. Gu, Z. Xu, and Z. Yang, "Symbolic OBDD representations for mechanical assembly sequences," *Computer Aided Design*, vol. 40, no. 4, pp. 411–421, 2008.

[9] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509–516, 1978.

[10] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[11] R. E. Bryant, "Symbolic Boolean manipulation with ordered Binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992.

[12] F.-M. Yeh, S.-K. Lu, and S.-Y. Kuo, "OBDD-based evaluation of k-terminal network reliability," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 443–451, 2002.

[13] M. Ghasemzadeh, C. Meinel, and S. Khanji, "K-terminal network reliability evaluation using binary decision diagram," in *Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications (ICTTA '08)*, pp. 1–5, Damascus, Syria, April 2008.

[14] G. Hardy, C. Lucet, and N. Limnios, "K-terminal network reliability measures with binary decision diagrams," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 506–515, 2007.

[15] MIL-STD-721B, *Definition of Effectiveness Terms for Reliability, Maintainability, Human Factors, and Safety*, Department of Defense, Washington, DC, USA, 1966.

[16] "F. S. Cudd: Cu decision diagram package release 2.3.1," http://vlsi.colorado.edu/.