# An Improved Byzantine Fault-tolerant Program for WSNs

Yi Tian
Shangluo University, Shangluo, China

***Abstract*—In order to increase the level of fault-tolerance of wireless sensor networks, thus to enhance the reliability and accuracy, we studies the traditional byzantine fault-tolerant program, and makes improvements in the environment of WSNs, reducing the number of rounds of message exchange between network nodes, thus improving the efficiency and reducing communication overhead and energy consumption. The simulation result shows that our program makes all normal network nodes reach an agreement, while the number of rounds of message exchange greatly decreases compared to the traditional byzantine program.**

***Index Terms*—WSNs; Fault Tolerance; Byzantine Fault-Tolerant Program; M-tree**

## I. INTRODUCTION

Wireless sensor networks (WSNs) are a distributed self-governance measurement and control network system composed of a lot of tiny sensors with communication and computation abilities. WSNs can make people get a large quantity of detailed and reliable information at any time, at any place, or under any condition; therefore, it is widely applied in national defense and military, environmental monitoring, traffic management, medical treatment and public health, and other fields.

Different application fields have different security requirements for WSNs. In most non-commercial applications such as environmental monitoring and air humidity, the security problem is not very important. However, in other fields such as monitoring the enemy's military deployment in the area occupied by the energy in terms of military, the things varying from data acquisition, data transmission to physical distribution of points have to be kept confidential to the enemy. In those security-sensitive applications, if the networks node in WSNs is captured or energy of the network node is exhausted, the consequence may be disastrous [1] [2] [3]. How to ensure the reliability of whole networks and the security of data transmission becomes the important content of security research on WSNs.

The traditional passive defense method such as encryption and identity authentication, can produce effective defense function on the attack from external part of WSNs. For example, it is unable to discover the originally normal network nodes controlled by the enemy by use of encryption and identity authentication, for there is private key in those network nodes. It is shown through previous research that there still exist loopholes in some network nodes which are vulnerable to the attack no matter how many security defense measures are taken. At the same time, those abovementioned defense measures basically can't confront the security threat from the internal part, such as network node failure [4] [5]. Therefore, the fault-tolerant program becomes the second barrier to guarantee the security.

Due to various limitations (such as computing power and storage space) on WSNs, the traditional fault-tolerant program can't be directly applied in WSNs; therefore, the fault-tolerant program must meet the specific application demand of WSNs. We hereby suggest designing a fault-tolerant program for WSNs by use of the advantages of the byzantine fault-tolerant program.

The byzantine fault-tolerant program originated from the byzantine agreement (BA) problem which was put forward by Lamport et al. in 1982 [6]. This problem is defined by Lamport [6] as follows.

(1) The messages sent out from network node can be correctly delivered.

(2) The receiver node knows the sender node who sends this message.

(3) When the number of network nodes is n, the number of network nodes in which the error happens will be (n-1)/3 at most.

Based on the above-mentioned hypothesis, BA problem can be solved [7], and the steps of solution are generally shown as below.

(1) Any network node is selected as the source network node, and then the source network node sends its initial value v(s) to all other network nodes.

(2) All normal network nodes make use of the message received from the source network node through exchange with other network nodes to check whether other normal network nodes have received the same value.

This solution can meet following requirements.

(1) A consensus can be reached among all normal network nodes.

(2) If the source network node is normal, the common value for which a consensus is reached among all normal network nodes shall be same with the initial value of the source network node.

Currently, the research on applicability of byzantine fault-tolerant program is mainly in the field of distributed computing system and wireless mesh networks [8]-[14]. Wang [14] et al. establishes a fault-tolerant network structure for wireless mesh networks according to traditional byzantine fault-tolerant program and improves


doi:10.4304/jnw.9.4.932-940

routing algorithm, thus improving the reliability of networks. However, for large number of rounds of message exchange between network nodes, it only establishes minimum message exchange unit from fixed topology for control. This is not applicable to WSNs with topological dynamic changes. Klempous [15] et al. applied traditional byzantine fault-tolerant program into WSNs and considered that the performance of fault-tolerance depended on the scope of message exchange between network nodes. If the scope of message exchange is too large, the large number of rounds of message exchange between network nodes will cause burden to communication; if the scope of message exchange is too small, normal network nodes cannot reach a consensus comprehensively, thus causing misinformation.

In view of this, we find that the major problem of the application of the byzantine fault-tolerant program into WSNs is that the number of rounds of message exchange between network nodes is too great. Generally, when the total number of network nodes is n and the maximum number of error network nodes is (n-1)/3, the number of rounds of message exchange required by traditional program is (n-1)/3+1 and the amount of message exchange in the ith round (i=1…(n-1)/3+1) is (n-1)(n-2)...(n-i). Which causes a huge communication overhead for WSNs with a large number of network nodes and meanwhile causes huge consumption for sensor with limited energy.

In the research on the traditional byzantine program, two features as follows are noticed: 1) the number of normal network nodes is at least n-(n-1)/3. 2) The total number of messages received from normal network nodes is always higher than that of messages received from error network nodes. If the abovementioned 2 points can be fully utilized, the efficiency of the traditional fault-tolerant program can be effectively improved, that is, the energy consumption and communication traffic can be reduced through reducing number of rounds of message exchange between network nodes so that such program can be applicable to WSNs. In this paper, we propose an improved byzantine fault-tolerant program.

The remainder of this paper is organized as follows. Section II describes the proposed fault-tolerant program. The simulations and their results are analyzed in Section III. Finally, our conclusions and future work are presented in Section IV.

## II. IMPROVED BYZANTINE FAULT-TOLERANT PROGRAM

Firstly, the hypothesis of the proposed program is given.

(1) The messages sent out from network node can be correctly delivered.

(2) The receiver node knows the sender node who sends this message.

(3) When the number of network nodes is n, the number of network nodes in which the error happens will be (n-1)/3 at most.

(4) In the first round of message exchange, only one network node can send messages.

Furthermore, all error network nodes have abnormal behaviors, that is, the values sent by error network node to other network nodes are different or there is difference between normal value and value sent by error network node.

### *A. The M-tree Used by Fault-Tolerant Program*

In the process of message exchange, each network node uses a specific tree to store messages received from other network nodes after message exchange. This tree is named as M-tree (message tree). Fig. 1 shows its structure. First, the source network node sends its original value v(s) to other network nodes and itself. As the sender of message is recognizable, all normal network nodes can recognize that this message comes from the source network node and store this value in the root node of M-tree. However, this cannot judge whether the source network node is normal network node. Therefore, multi-round message exchange between network nodes is required so as to eliminate the influence of error source node. Starting from the second level of M-tree, each node is named with the sender of storing value and father node (the second level is not named with root node). For example, if the storing value of a node in the second level comes from network node x, this node is named as v(x); if the father node of a node in the third level is v(x) and its storing value comes from network node y, this node is named as v(xy); the rest can be done in the same manner. Nodes with continuously repetitive network nodes in the name (such as v(xx), v(yy) and v(xyy)) are not stored in M-tree so as to prevent error network nodes from sending error messages periodically which makes error messages stored in M-tree repeatedly and causes misjudgment.

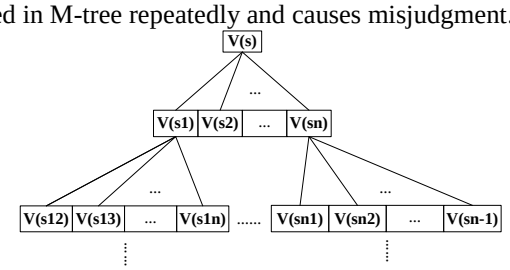


Figure 1. Structure of M-tree

### *B. Decision Process of Trusted Network Nodes*

For all network nodes, the decision process of trusted network nodes can be implemented after three rounds of message exchange. First, it is necessary to judge whether the storing value of each node (marked as v(cx)) in the second bottom level of M-tree is equal to the value in the majority (expressed as maj(cx)) among all its corresponding child node values or not. If it is, it is required to judge whether the total number of child nodes (expressed as cou_maj(cx)) with the same value of maj(cx) of each v(cx) is not less than (n-(n-1)/3-1), i.e. judge whether the total number of network nodes with the majority is not less than the minimum total number of normal network nodes. If both of abovementioned conditions are satisfied, the network nodes that sends the

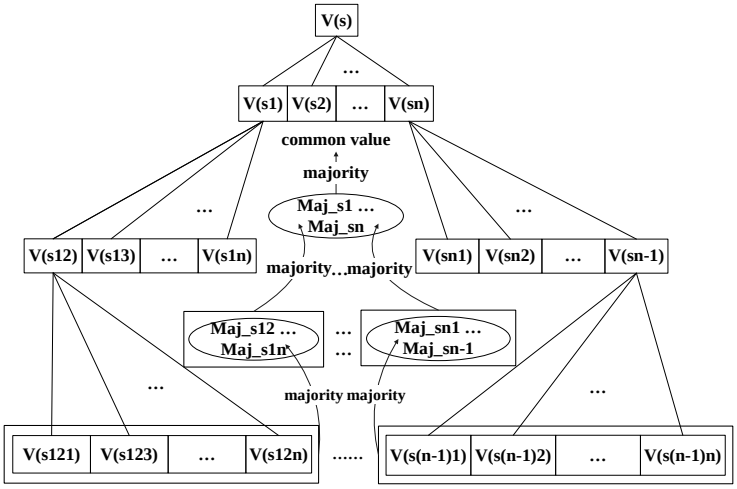


Figure 2. The process of getting common value

same value with maj(cx) to the node v(x) and its subtree are added into the reliable-like network point set (RNPS, recorded as RNPS(cx)). In general, judging whether a network node can be added into RNPS (cx) shall meet the following 2 conditions:

(1) v(cx) = maj(cx)

(2) cou_maj(cx) >= (n−(n−1)/3−1)

Then, the total number of each network node in all RNPS in the second bottom level (expressed as cou_RNP(z)) is counted. If cou_RNP(z) of a network node is not less than (n−(n−1)/3−1), this network node is defined as trusted network node.

### *C. Correction Process of Error Message*

After new trusted network node is added, each network node will correct messages from untrusted network node in M-tree to accelerate the election of common value. Subtrees of all nodes in the second bottom level of M-tree will be examined. If the storing value of its child node is different from the value in the majority (expressed as maj(RNP(cx))) among values sent by current trusted network node and the sender node of this value is not trusted network node, the value of child node will be replaced as maj(RNP(cx)).

### *D. Election Process of Common Value*

After all error messages are replaced, M-tree selects majority values from each subtree in the second bottom level and then selects majority values towards the upper level till the root. Then, the common value of this network node is obtained. The example diagram is shown in Fig. 2.

### *E. Overall Operational Process of Fault-Tolerant Program*

According to the design above, the minimum number of rounds of message exchange is 5. Starting from the 4th round, if no new trusted network node is found in each network node for two consecutive rounds, i.e. all normal network nodes reach a consensus, the operation of the program ends. Fig. 6 shows the details of fault-tolerant program. The main steps is described as below.

Step 1: First, necessary message exchange phase. In the first round of message exchange, the source network node sends its original value v(s) to itself and other network nodes and each network node stores this value in root node of M-tree. Then, go to step 2.

Step 2: In the 2nd round of message exchange, each network node sends root node value in M-tree to itself and all other network nodes and each network node stores the received value in the corresponding node of the second level of M-tree. Then, go to step 3.

Step 3: In the 3rd round of message exchange, each network node sends the storing value in the second level of M-tree to itself and all other network nodes and each network node stores the received value in the corresponding node of the third level of M-tree. Then, go to step 4.

Step 4: After the 3rd round of message exchange, each network node enters decision-making phase and the decision process of trusted network node starts. After the decision process, the condition that whether the cumulative variable null message exchange (cou_nullmess) is less than 2 needs to be judge and the initial value of cou_nullmess is 0. If judgment is false, the operation of the program ends. If judgment is true, then judge whether new trusted network node is found. If the result is true, cou_nullmess returns to 0, go to step 5; otherwise, cou_nullmess pluses 1, start the next round of message exchange and the new decision-making phase.

Step 5: After new trusted network node is added, each network node starts the correction process of error message and replace the message sent by untrusted network node in M-tree. Then, go to step 6.

Step 6: Each network node elects common value and the common value of all normal network nodes is obtained. Then start the next round of message exchange and the new decision-making phase.

### *F. Example of Applying Fault-Tolerant Program*

In this section, An example is given to explain how the proposed fault-tolerant program can make all normal network nodes reach an agreement. We assume in a seven-node WSNs, network nodes are marked as a, b, c, d, e, f, g. Node a is defined as source network node. To

check the performance of the proposed program, we design the worst-case scenario, that is Node c and Node e are both error network node. The messaging behavior of the error network nodes is shown in Table 1.

TABLE I. THE MESSAGING BEHAVIOR OF THE ERROR NETWORK NODES

| | a | b | d | f | g |
|---|---|---|---|---|---|
| a | 2 | 2 | 2 | 2 | 2 |
| c | 2 | 0 | 0 | 0 | 1 |
| e | 0 | 1 | 0 | 3 | 1 |

The beginning of the program is the necessary message exchange phase, the source network node a sends its initial value to itself and all other network nodes in the first round of the message exchange. Because the source network node a is normal network node, the sending values are the same and the value is 2. Then, each network node stores this value v(s) in the root of M-tree , as shown in Fig. 3. Here, the results of error network nodes do not need to be discussed, thus this example only shows the results of normal network nodes.

**Node a, V(a) = 2**

**Node b, V(a) = 2**

**Node d, V(a) = 2**

**Node f, V(a) = 2**

**Node g, V(a) = 2**

Figure 3. The value stored in each normal network node's M-tree

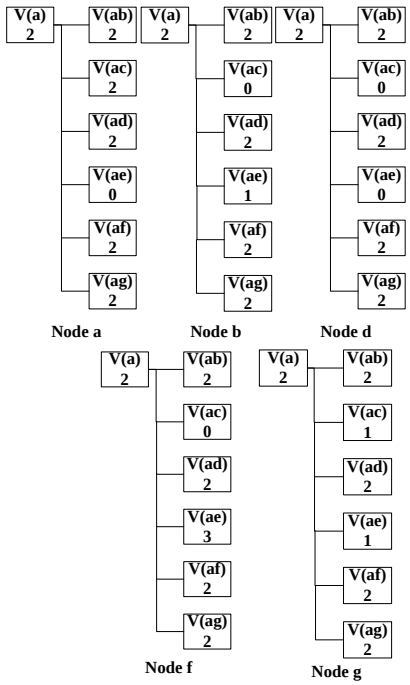


Figure 4. The result of storing values in each normal network node's M-tree after the second round of message exchange

In the 2nd round of message exchange, each network node sends the root value in M-tree to itself and all other network nodes. Similarly, each network node stores the received value in the corresponding node of the second level of M-tree. The results for the second round of message exchange are shown in Fig. 4.

In the 3rd round of message exchange, each network node sends the storing value in the second level of M-tree to itself and all other network nodes and each network node stores the received value in the corresponding node of the third level of M-tree. The results of all normal network nodes' M-trees are shown in Fig. 5.

After the 3rd round of message exchange, the program accesses the Decision-making phase, and the first process of this phase is the decision process of trusted network. The process specifics of Node d is shown in Fig. 7. For example, the Node a, b, d, f , g should be added in RNPS(ab), that the following conditions are satisfied:

(1) v(ab) = maj(ab)

(2) cou_maj(ab) >= (n−(n−1)/3−1)

(3) v(abx) = maj(ab){such as, v(aba), v(abd), v(abf), v(abg) = maj(ab) = 2}.

Next, the program counts the number (cou_RNP(z)) of each network node occurring in all RNPS in the second bottom level of M-tree and computes whether cou_RNP(z) is not less than (n−(n−1)/3−1). For example, Node b is appeared in RNPS(ab), RNPS(ac), RNPS(ad), RNPS(af), RNPS(ag). Obviously, cou_RNP(b) is 5, which is greater than (n−(n−1)/3−1) = 4. Hence, Node d is defined as trusted network node. Correspondingly, Node a, d, f, g are also trusted network node.

The program checks whether the cumulative variable null message exchange (cou_nullmess) is less than 2. If the result is false, the program ends. On the contrary, if new trusted network node is found and added, the correction process of error message can be executed, and cou_nullmess returns to 0. If no new trusted network node is found and added, cou_nullmess pluses 1, and the program starts the next round of message exchange and the new decision-making phase. In this example, Node a, b, d, f, g are all new trusted network node and cou_nullmess remains initial state, the value is 0, so the program runs the correction process.

In this process, the values received from the untrusted network nodes in each subtree of the second bottom level of M-tree must be replaced by the majority value of the trusted network node in each subtree. For example, as shown in Fig. 8, Node c and e are untrusted network nodes in the subtree of v(ab). Furthermore, v(abc) and v(abe) is not equal to the majority value of the trusted network node set of v(ab){v(aba), v(abd), v(abf), v(abg)}. Thus the value of v(abc) and v(abe) must be replaced by 2 (maj(RNP(ab)) = 2).

Finally, the majority values from each subtree in the second bottom level of M-tree can be elected. Then the election process can be executed upwards layer by layer till the root. So far, the decision-making phase has been completed, the program starts the new round of message exchange and repeats the decision-making phase. In this example, all the normal network nodes have reached a agreement in the 3th round of message exchange. So the total number of rounds of message exchange used by this example is 5. This value is equal to the minimum design number of rounds.

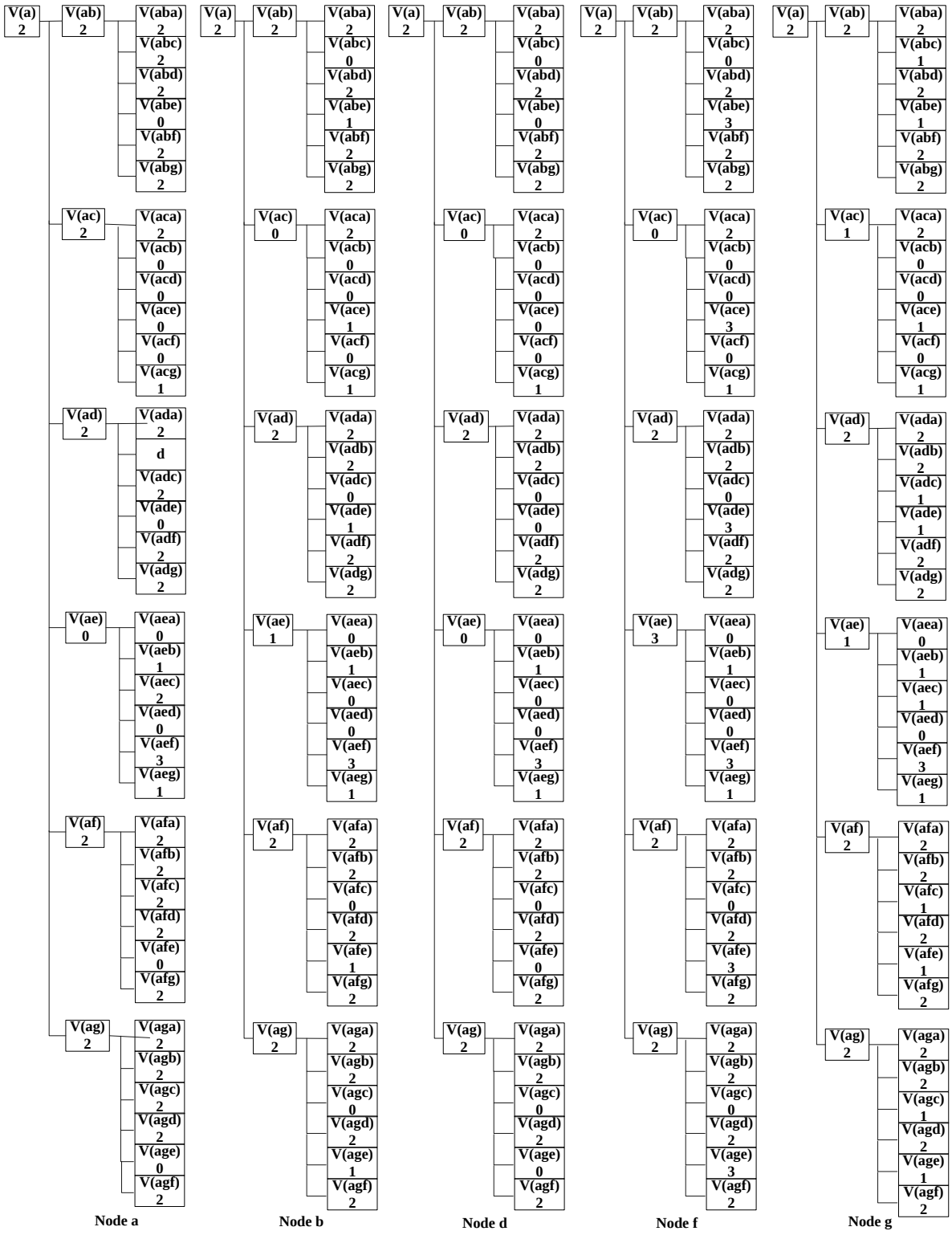


Figure 5. The result of M-tree's storing values in network node a, b, d, f, g after the third round of message exchange

## III. SIMULATIONS

To study the performance of the proposed fault-tolerant program and the traditional byzantine fault-tolerant program in the number of rounds of message exchange, NS2 is used as basic simulation platform. The basic setting of simulation scene: 80 network nodes are distributed randomly in 500m×500m area. The covering radius of sensor is 100m. There are 2 independent variables in simulation scene: the number of error network nodes {5, 10, 15, 20, 25} and whether error network node is the source network node. There are 10 scenes in total and each scene is simulated for 10 times. Each simulation ends till the completion of fault-tolerant program. The final data is the average value of ten simulations.

**Byzantine fault-tolerant program**

**Definition**
n: The number of network nodes in WSNs.
v(cx): The node in M-tree.
maj(cx): The value in the majority among values stored in child nodes of node v(cx) in M-tree.
cou_maj(cx): The frequency of values stored in child nodes of node v(cx) in M-tree equal to maj(cx).
RNPS(cx): The reliable-like network point set of node v(cx) in M-tree.
cou_RNP(z): The total number of network node z occurring in all reliable-like network point sets in certain level of M-tree.
RNP(cx): The trusted network node set of node v(cx) in M-tree.
maj(RNP(cx)): The value in the majority among values stored in trusted network node of node v(cx) in M-tree.
cou_nullmess: The total number of rounds of message exchange without new trusted node found, and its initial value is 0.
**Necessary message exchange phase**
The 1st round
The source network node sends its original value v(s) to itself and other network nodes and each node stores this value in root node of M-tree.
The 2nd round
Each network node sends root node value in M-tree to itself and all other network nodes and each node stores the received values in the second level of M-tree.
The 3rd round
Each network node sends the storing value in the second level of M-tree to itself and all other network nodes and each node stores the received values in the third level of M-tree.
**Decision-making phase**

```
//Decision process of trusted network node
foreach all nodes in certain level of M-tree
{
If(v(cx) = maj(cx) and cou_maj(cx) >= (n−(n−1)/3−1)) then
{
 Add network node x to RNPS(cx);
 foreach child node of v(cx)
 {
 If (v(cxy) = v(cx)) then
 {
 Add network node y to RNPS(cx);
 }
 }
}
foreach network node z of this network
{
 Count cou_RNP(z) in all RNPS in this level of M-tree;
 If (cou_RNP(z) >= (n−(n−1)/3−1)) then
 {
 Network node z is defined as trusted network node;
 }
}
//Judge whether to stop the program
If (cou_nullmess < 2) then
{
 If (new trusted network node is added in RNP(cx)), then
 {
 cou_nullmess = 0;
 //Correction process of error message
 foreach all nodes in certain level of M-tree
 {
 foreach child node of v(cx)
 {
 If (the sender of v(cxy) is not in RNP(cx) and v(cxy) ≠ maj(RNP(cx)))), then
 {
 v(cxy) = maj(RNP(cx));
 }
 }
 }
 //Election process of common value
 Elect majority value level by level starting from the subtree of node in the second bottom level of M-tree and obtain common value;
 }
 else
 {
 cou_nullmess = cou_nullmess + 1;
 }
 //Start the new round of message exchange
 Each network node sends the value stored in the bottom level of M-tree to itself and all other network nodes;
 Each node stores the received values in the new bottom level of M-tree;
 //Start decision-making phase
 Goto decision-making phase;
}
```

Figure 6. Improved byzantine fault-tolerant program

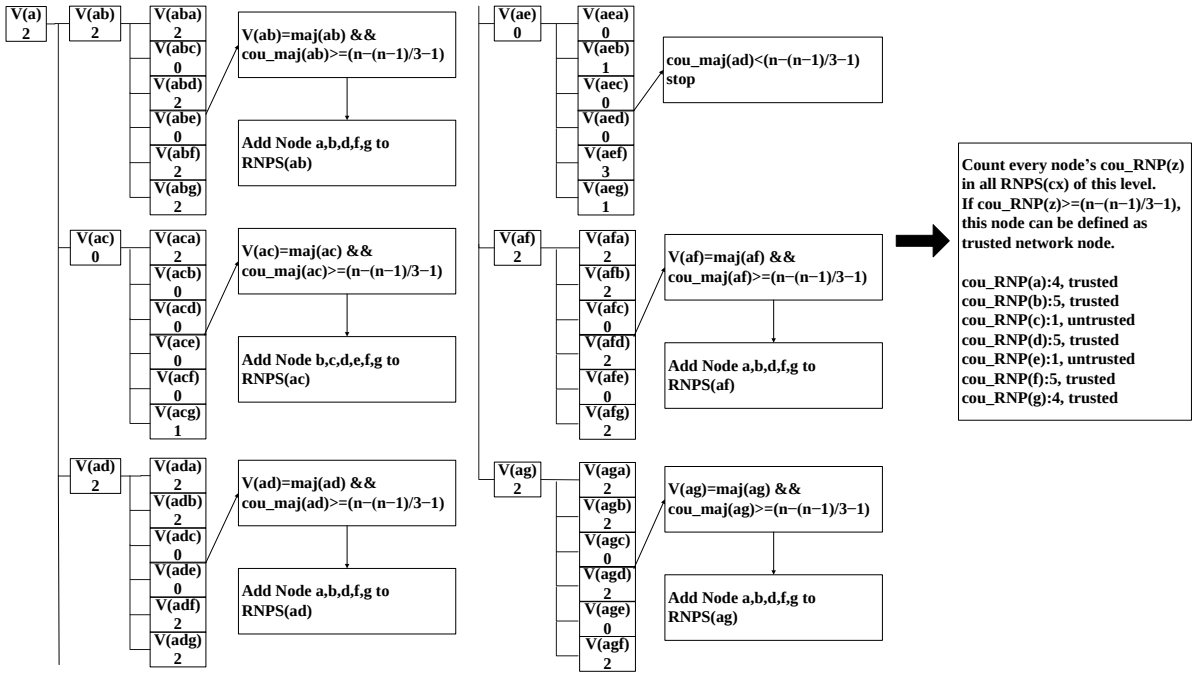


Figure 7. The decision process of trusted network (for Node d)

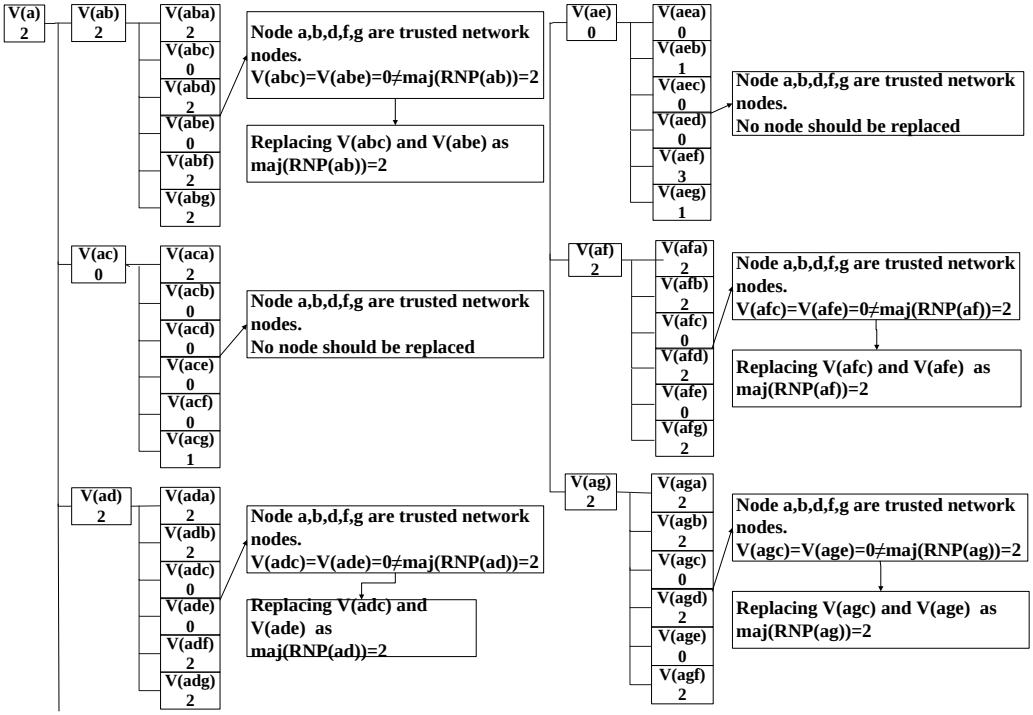


Figure 8. Correction process of error message (for Node d)

Fig. 9 shows the comparison of the number of rounds of message exchange in the proposed program and the traditional byzantine program when the source network node is not error network node. Compared to traditional byzantine program, the proposed program has obvious advantages. When the number of error network nodes is small, the number of rounds of message exchange in the proposed program is always the minimum design number of rounds. It slightly increases when the number of error network nodes approaches the maximum value of network design. This is because that network nodes with the normal value in the whole network are always in the majority, i.e. this normal value is always in the majority in M-tree of each network node, thus making the final common value equal to this value. It is unnecessary to find error nodes to determine the final common value through at least (n− 1)/3 +1 rounds of message exchange in traditional byzantine program, thus reducing the number of rounds of message exchange.

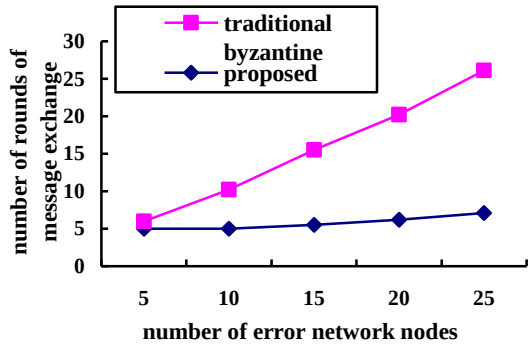


Figure 9. Comparison of the number of rounds of message exchange in the proposed program and traditional byzantine program (source network node is not error network node)

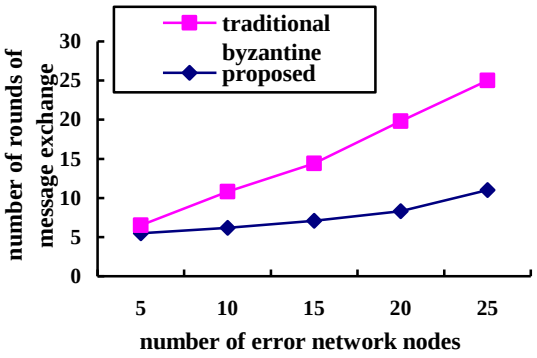


Figure 10. Comparison of the number of rounds of message exchange in the proposed program and traditional byzantine program (source network node is error network node)

Fig. 10 shows the comparison of the number of rounds of message exchange in the proposed program and the traditional byzantine program when the source network node is error network node. Compared to Fig. 9, in the proposed program, the number of rounds of message exchange required when the source network node is error network node is generally higher than that when the source network node is not error network node under the condition of same number of error network nodes, because when the source network node is error network node, the original value v(s) sent to all network nodes will be different, which increases the difficulty of determining common value of normal network nodes. The number of rounds of message exchange increases correspondingly. However, compared to traditional byzantine program, the number of rounds of message exchange required in the proposed program is still small.

## IV. CONCLUSIONS

We study and improve the traditional byzantine fault-tolerant program and design a fault-tolerant program applied in WSNs. The proposed program uses normal network nodes in the network as measurement standard of common value, which not only ensures the reliability and correctness of the whole network, but also solves the problem of huge energy consumption and communication overhead caused by large number of rounds of message exchange in traditional byzantine fault-tolerant program. The subsequent simulation proves that the number of rounds of message exchange in the proposed program decreases greatly compared to traditional byzantine fault-tolerant program whether or not source network node is error network node. In the future research work, the real-time problem of the proposed program will be considered. We will optimize the code and process of the program, thus making the time spend in reaching all normal network nodes a consensus as short as possible.

## ACKNOWLEDGMENT

This work is supported by a grant from Shanxi's Department of Education fund (2013JK1160)

## REFERENCES

[1] Y. Zhou, Y. Fang, Y. Zhang, "Securing wireless sensor networks: a survey," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 3, pp.6-28, 2008.

[2] I. Bekmezci, F. Alagöz, "Energy efficient, delay sensitive, fault tolerant wireless sensor network for military monitoring." *International Journal of Distributed Sensor Networks*, vol. 5, no. 6, pp.729-747, 2009.

[3] X. Chen, K. Makki, K. Yen, N. Pissinou, "Sensor network security: a survey." *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 2, pp.52-73, 2009.

[4] J. Yick, B. Mukherjee, D. Ghosal, "Wireless sensor network survey." *Computer networks*, vol. 52, no. 12, pp.2292-2330, 2008.

[5] T. Kavitha, D. Sridharan, "Security vulnerabilities in wireless sensor networks: A survey." *Journal of information Assurance and Security*, vol. 5, no. 1, pp.31-44, 2010.

[6] L. Lamport, R. Shostak, M. Pease, "The Byzantine generals problem." *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp.382-401, 1982.

[7] M. Correia, N. A. Bessani, P. Verísimo, "On Byzantine generals with alternative plans." J*ournal of Parallel and Distributed Computing*, vol. 68, no. 9, pp.1291-1296, 2008.

[8] M. B. Kapron, D. Kempe, V. King, J. Saia, V. Sanwalani, "Fast asynchronous byzantine agreement and leader election with full information." *ACM Transactions on Algorithms (TALG)*, vol. 6, no. 4, pp.68, 2010.

[9] V. King, J. Saia, "Breaking the O(n2) bit barrier: scalable byzantine agreement with an adaptive adversary." *Journal of the ACM (JACM)*, vol. 58, no. 4, pp.18, 2011.

[10] M. A. AlZain, B. Soh, E. Pardede, "A Survey on Data Security Issues in Cloud Computing: From Single to Multi-Clouds" *Journal of Software*, vol. 8, no. 5, pp. 1068-1078, 2013.

[11] L. Zhang, Q. Zhu, A. Chen, "Fast Message Dissemination Tree and Balanced Data Collection Tree for Wireless Sensor Network" *Journal of Software*, Vol. 8, No. 6, pp. 1346-1352, 2013.

[12] S. S. Wang, Q. K. Yan, C. S. Wang, "Achieving efficient agreement within a dual-failure cloud-computing environment." *Expert Systems with Applications*, vol. 38, no. 1, pp.906-915, 2011.

[13] L. M. Chiang, "Eventually Byzantine Agreement on CDS-based mobile ad hoc network" *Ad Hoc Networks*, vol. 10, no. 3, pp.388-400, 2012.

[14] J. Wang, Y. Zhang, J. Wu, "Byzantine Fault Tolerant Network Structure and Algorithm in WMN." *Computer Engineering*, vol. 37, no. 20, pp.83-86, 2011.

[15] R. Klempous, J. Nikodem, L. Radosz, N. Raus, "Byzantine Algorithms in Wireless Sensors Network" *in Information and Automation, ICIA 2006*, pp.319-324, 2006

**Yi Tian** received his M.S. degrees in 2011, from Northwest University of Information Technology. He is now an lecturer in Shangluo University. His research interests include computer system architecture, internet of things.