Workshop on Computer Systems

Reach Unanimity Problem on an Unknown Network

S.C. Wang
Department of Information Management
scwang@cyut.edu.tw

K.Q. Yan
Department of Business Administration
kqyan@mail.cyut.edu.tw

C.F. Cheng
Department of Information and Communication of Engineering
s9027601@mail.cyut.edu.tw

Chaoyang University of Technology
168 Gifeng E. Rd., Wufeng,
Taichung County, Taiwan 413, R.O.C.
TEL: 886-4-2332-3000 ext. 3071
Fax: 886-4-2374-2319

ABSTRACT

The Byzantine Agreement is an important topic in the reliable distributed system because the system can cope with the influences from faulty components only when the agreement is achieved. In the literature concerned, the BA problem has been well formulated in a Fully Connected Network (FCN), a Generalized Connected Network (GCN), and a MultiCasting Network (MCN) under the assumption that each processor in the network has the common knowledge of the graphic information about the entire network structure. However, in the real world, each processor may not have the common knowledge of the graphic information about the entire network structure. That is, the processors may only have the partial knowledge as to their own graphic information. In this paper, the Byzantine Agreement problem will be visited in an Unknown Network (UNet) to increase the capability of fault tolerance by allowing faulty processors with dual failure mode. The proposed protocols, the Unknown Agreement Protocol (UAP) and the Relay Fault-tolerance Channel (RFC), use the minimum number of rounds of message exchange and can tolerate the maximum number of faulty processors allowed.

Keywords: Byzantine Agreement, Distributed system, Fault tolerance, Unknown network, Dual failure mode.

1. INTRODUCTION

In order for the reliability of the fault-tolerant distributed system, reaching a common agreement at the presence of malfunctioning components is the central issue. Such an agreement problem was first studied by Lamport [7], who named it the Byzantine Agreement (BA). In the BA problem, there is a transmitter that transmits the messages at the first round. After the message exchange, each healthy processor should agree on the same value. The BA problem in the distributed system has been defined as follows:

(1) There are n processors in a distributed system, where n is a constant.

(2) Each processor can communicate with each other through reliable network.

(3) One or more of the processors may be failed, so a faulty processor may transmit incorrect message(s) to other processors.

(4) After message exchange, all healthy processors should reach a common agreement, if and only if the number of malicious faulty processors f_p is less than one-thirds of the total number of processors in the network ($f_p \leqq \lfloor (n-1)/3 \rfloor$).

(5) The number of rounds of message exchange is f_p+1.

The BA is achieved when the following constraints are met:

(Agreement): All healthy processors in the network agree on a single common value;

(Validity): If the transmitter is healthy, then all healthy processors in the network should agree on the transmitter's initial value.

In previous works [2,4,5,6,7,8,9,10,11,12], all the results with the BA problem were built upon the same assumptions that the network structure is well defined and that each processor in the network has the common knowledge of the graphic information of the entire network structure. That is, each processor knows what the total number of processors in the network is, knows what the connection state of all the processors in the network is like, and can identify each processor in the network. However, in real-world applications, each processor may probably only know the connection state of its own without the knowledge of the connection state of the others. Therefore, most previous results may unfortunately be un-applicable in a UNet.

There are three kinds of symptoms a faulty component may have. They are crash, omission and

malicious fault [8]. A crash fault happens when the component is broken, and the behavior of an omission fault is that the component fails to transmit or receive a message on time or at all. The symptoms having to do with crash and omission can be detected by a fault-free processor if the protocol appropriately encodes a transmitted message in either the Non-Return-to-Zero code or the Manchester code [12] before transmission, so we call them dormant faults. On the other hand, a malicious fault happens when the behavior of a faulty component is totally unpredictable and arbitrary. Therefore, the most serious influence a faulty component can have on the system is a malicious fault. In previous results [2,4,5,6,7,11], the researchers visited the BA problem with fallible components on malicious fault only, and we should consider it a step forward if we can classify failures into dormant faults and malicious faults before trying to reach the Byzantine agreement.

To make our new protocols work in practical, real-life applications, we must visit the BA problem with dual failure mode in an UNet. As we will point out later, our new protocols use the minimum number of rounds of message exchange and can tolerate the maximum number of faulty processors allowed with dual failure mode in an UNet.

The rest of this paper is organized as follows. Section 2 proposes the detail descriptions of the proposed protocols RFC and UAP. Section 3 gives an example of executing RFC and UAP. Section 4 proves the correctness and the complexity of our protocols. Finally, Section 5 gives the conclusion.

2. PROPOSED PROTOCOLS

In this section, we shall introduce the proposed protocols RFC and UAP to solve the BA problem with dual failure mode for the processors in a UNet. The assumptions and parameters of our protocols are listed as follows:

- The processors in the underlying network are assumed to be fallible.
- A processor that transmits a message is called a sender processor.
- There is only one transmitter who transmits the message(s) at the first round in solving the BA problem.
- Let m be the maximum number of malicious faulty processors.
- Let d be the maximum number of dormant faulty processors.

- Let n be the total number of processors in the network, where $n > \lfloor (n-1)/3 \rfloor + 2m + d$.
- Let f_n be the maximum number of malicious and dormant faulty processors, $f_n = m + d$, $f_n \geqq f_p$.
- Let r be the minimum number of rounds of messages exchange, where $r = f_p + 1$ and $f_p \leqq \lfloor (n-1)/3 \rfloor$.
- Each processor in the network can be identified.
- Each processor knows the total number of processors in the network.
- Each processor in the network does not know other processors' connection state.
- Let c be the lowest bonded connectivity of the UNet, where $c > 2m+d$
- Let c_i be the connectivity of processor P_i in the network where $c_i \geq c$, *for* $1 \leq i \leq n$
- A processor does not know the fault status of the other processors, while dormant faulty processors can be detected.

In UAP, RFC is used to transmit messages, and the number of rounds of UAP operations is $f_p + 1$ ($f_p \leqq \lfloor (n-1)/3 \rfloor$). RFC can provide a reliable channel to help the processors to transmit messages to each other, and using RFC can make an un-fully connected network act just like a fully connected network without the common knowledge of the graphic information of the whole network structure. The definition of the protocol RFC is shown in Figure 1. In a UNet, each processor only has the partial knowledge of its own graphic information. For example, in Figure 2(a), P_1 and P_3 only have the information of the connection state between them two. So, it is impossible for P_1 to transmit a message to P_3 directly, and the reason is that P_1 does not know the location of P_3. In this study, the idea of our modified transmitter protocol RFC comes from the concept of virtual link by F.J. Meyer and D.K. Pradhan [8]. The concept of virtual link is graceful; however, it needs the common knowledge of the network structure and the connection state of each processor to create virtual links, and therefore it cannot be used in a UNet. So, in this study, we shall modify it and make it more formal and practical. Our modified protocol can enable a sender processor to transmit a message to its destination processor without the location information of the destination processor.

UAP can tolerate m malicious faulty processors and d dormant faulty processors, where $n > \lfloor (n-1)/3 \rfloor + 2m + d$ and $c > 2m+d$. The definition of protocol UAP is shown in Figure 3. There are two phases in protocol UAP, which are the message exchange phase and the decision making phase.

In the message exchange phase, each processor exchanges messages with others to get enough

information through RFC, which needs f_p +1 rounds of message exchange. An example of executing RFC is shown in Figure 2. In Figure 2(a), the destination processor can receive messages from *c* physical links. By using RFC, the sender processor will transmit the message to all the other processors with physical links to the sender processor, and each intermediate processor will also broadcast the message to other processors. So, the destination processor can receive many copies of a message from these *c* physical links, the destination processor can match them to be *c* node-disjoint paths [3] through which each processor can identify the sender processor of any given message. For example, in Figure 2(a), the sender processor is P_1 and the destination processor is P_3. By using RFC, the sender processor will transmit the message to P_2, P_4, P_5 and P_6 (but the destination processor does not know the connection state of the sender processor P_1), and the destination processor P_3 can receive messages from P_2, P_4, P_5 and P_7. Therefore, in this case, there are four node-disjoint paths (as shown in Figure 2(b)), which are $P_1 \rightarrow P_2 \rightarrow P_3$, $P_1 \rightarrow P_4 \rightarrow P_3$, $P_1 \rightarrow P_5 \rightarrow P_3$ and $P_1 \rightarrow P_6 \rightarrow P_7 \rightarrow P_3$. At the end of each round of message exchange, the destination processor uses function VMAJ (shown in Figure 1) to process its received message(s), by way of RFC to get a single value and to store the majority value in its mg-tree (the detailed description of the mg-tree is presented in Appendix I). The structure of an mg-tree is shown in Figure 4(c). So, when a sender processor wants to transmit its message(s) to its destination processor, the destination processor can get the information through physical links by RFC.

In the decision making phase, all the healthy processors turn their mg-trees into their corresponding ic-trees (the detailed description of the ic-tree is presented in Appendix II) by deleting the vertices with repeated processor names. The structure of an ic-tree is shown in Figure 4(e). Then the processors use function VOTE (shown in Figure 3) to obtain the common value.

Protocol RFC

Definition:

- Each processor has the partial knowledge of itself graphic information G=(E,Þ), where Þ is the set of processors in the network and E is a set of processor pairs (P_i,P_j) indicating a physical link between processor P_i and processor P_j ,where $1 \le i,j \le n$.
- There are at least c (c>2m+ d) paths to each processor.
- These c paths from sender processor to destination processor are node-disjoint paths.
- Each intermediate node on these c paths should not be passed through more than once.

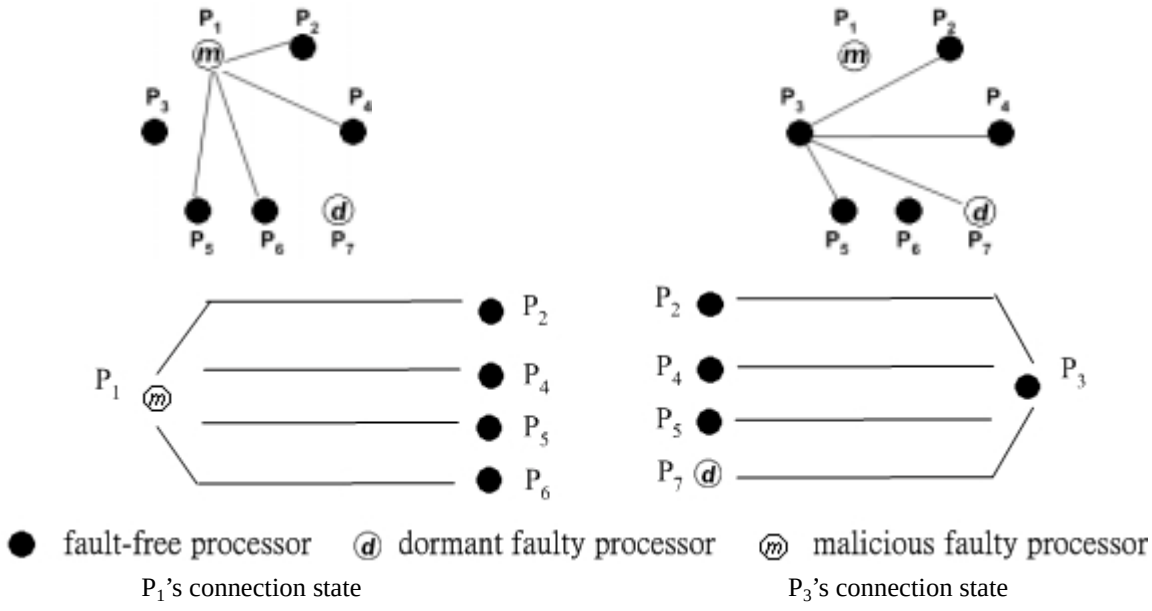
Steps:

1. The sender processor P_i ($1 \le i \le n$) transmits the initial v_i to all processors which have connection(s) with the sender processor.
2. If the node-disjoint path from sender processor to destination processor passes through any dormant faulty processor or if the sender processor has dormant faults, then replace λ.
3. The destination processor constructs the vector $V_i = [v_{path\ 1}, v_{path\ 2},\ldots, v_{path\ c-1},\ldots, v_{path\ c}]$ for c>2m+d.
4. The destination processor applies VMAJ on vector V_i.

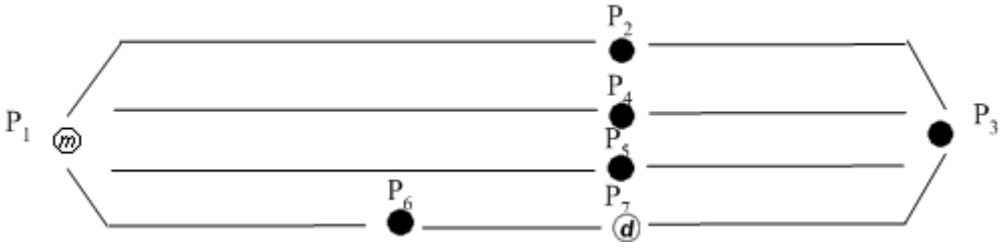
The function VMAJ(V)=
1. The majority value[#] in the vector $V_i = [v_{path\ 1}, v_{path\ 2},\ldots, v_{path\ c-1},\ldots, v_{path\ c}]$.
2. A default value ϕ is chosen, otherwise.

[#] If the number of the set $\{v_1,\ldots,v_i\ldots,v_c\}$ is the majority value. For instance, the majority value of the set {0,1, 0, λ, λ,1,1 } is 1.

Figure 1. The proposed protocol RFC



(a) The connection state of the sender processor P_1 and that of the destionation processor P_3



(b) The node-disjoint paths from sender processor P_1 to destination processor P_3

Figure 2. An example of executing RFC

Protocol UAP (for each processor in a UNet)

Compute the number of rounds required r :

$$r = \lfloor (n-1)/3 \rfloor + 1$$

Message Exchange Phase:

If $r = 1$, then:

1. The transmitter transmits its initial value v_t to the other processors and itself through the physical links directly.
2. If the transmitter is a processor in dormant fault, then replace the value with λ.
3. Each processor stores its received value in the root t of its mg-tree.

For $r > 1$, do:

1. Each processor transmits the values at level r-1 in its mg-tree to the others and itself through the physical links directly.
2. If the messages are from any processor in dormant fault, then replace the value with λ.
3. Store the received values at level r of its mg-tree.

Decision Making Phase:

Step 1: Turn the mg-tree into its corresponding ic-tree by deleting the vertices with repeated names.
Step 2: Use function VOTE to process root t of each processor's ic-tree and to obtain the common value VOTE(t).

The function VOTE(α)=

1. val(α), if α is a leaf.
2. The majority value* in the set of {VOTE(αi)| $1 \le i \le n$,and vertex αi is a child of vertex α }, if such majority values exist.
3. A default value ϕ is chosen, otherwise.

*The majority value of the set $\{v_1,\ldots,v_i\ldots,v_n\}$ is v_i if the number of v_i's present in the set is greater than $\lfloor n/2 \rfloor$. For instance, the majority value of the set {0,1,0,1,0 } is 0.

Figure 3. The proposed protocol UAP

3. AN EXAMPLE OF EXECUTING RFC and UAP

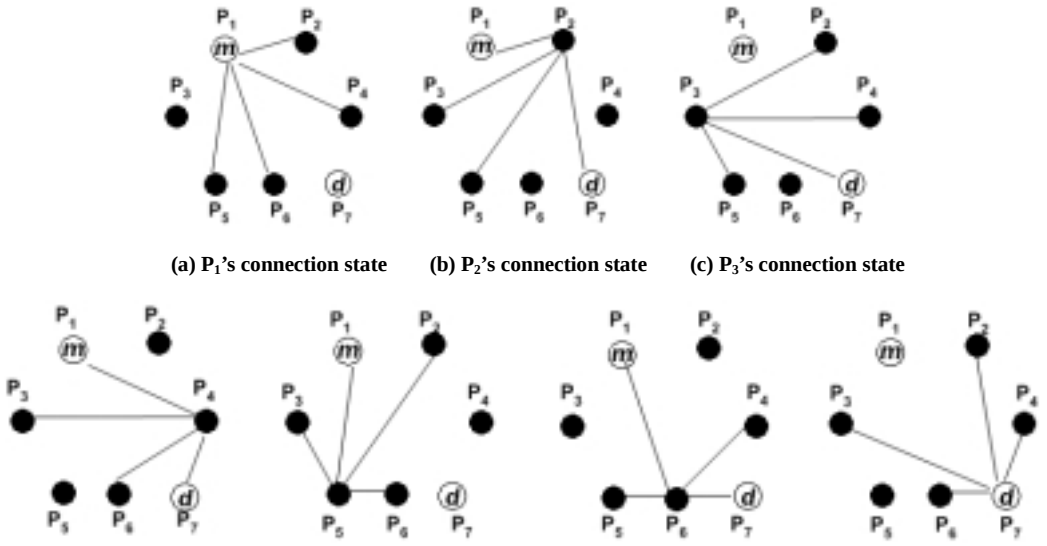
In this section, we shall give an example of how RFC and UAP are executed in Figure 5. Figure 4 shows the connection state of each processor, where the transmitter is P_1, the dormant faulty processor is P_7, and the malicious faulty processor is P_1.

The worst case of the BA problem is that the transmitter is a malicious faulty processor [7]. For example, assume P_1, the malicious faulty processor, is our transmitter is this case, so the transmitter may transmit different values to different processors. In order to reach a common agreement among healthy processors in our example, UAP needs 3 ($\lfloor (n-1)/3 \rfloor$+1) rounds of message exchange.

In the first round of the message exchange phase, the transmitter processor, P_1, uses RFC to transmit messages to other processors on the right-hand side of Figure 5(b). Figure 5(a) shows the messages sent by the transmitter processor P_1. The messages stored by healthy processors P_2, P_3, P_4, P_5 and P_6 in the first round of message exchange are illustrated on the left-hand side of Figure 5(b). Never mind the messages stored by

faulty processors. In the *r*-th (*r* >1) round of message exchange, all the processors use RFC to transmit the values at the $(r\text{-}1)^{th}$ level in their mg-trees to each other and itself. Using RFC, the destination processor can receive *c* messages from the sender processor. Then, using the function VMAJ to process the received values, it can get a single value and store the value at level *r* in its mg-tree. An example of processor P_2 at the 2_{nd} round message exchange is in Figure 5(c), and an example of processor P_2 at the 3_{rd} round message exchange is in Figure 5(d).

In the decision making phase, each healthy processor turns its mg-tree into a corresponding ic-tree by deleting the vertices with repeated names. An example of the ic-tree is illustrated in Figure 5(e). Finally, using the VOTE function to root the value *t* for each processor's ic-tree, a common value 1 is obtained (as shown in Figure 5(f)). So, the common root value of the healthy processors P_2 and P_3 is replaced by 1.

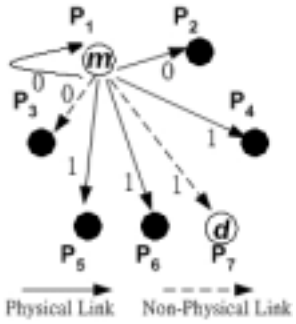


(a) P_1's connection state **(b) P_2's connection state** **(c) P_3's connection state**

(d) P_4's connection state **(e) P_5's connection state** **(f) P_6's connection state** **(g) P_7's connection state**

The transmitter is P_1, malicious faulty processor is P_1 and dormant faulty processor is P_7

Figure 4. An example of each processor's connection state



(a) The message is sent by transmitter

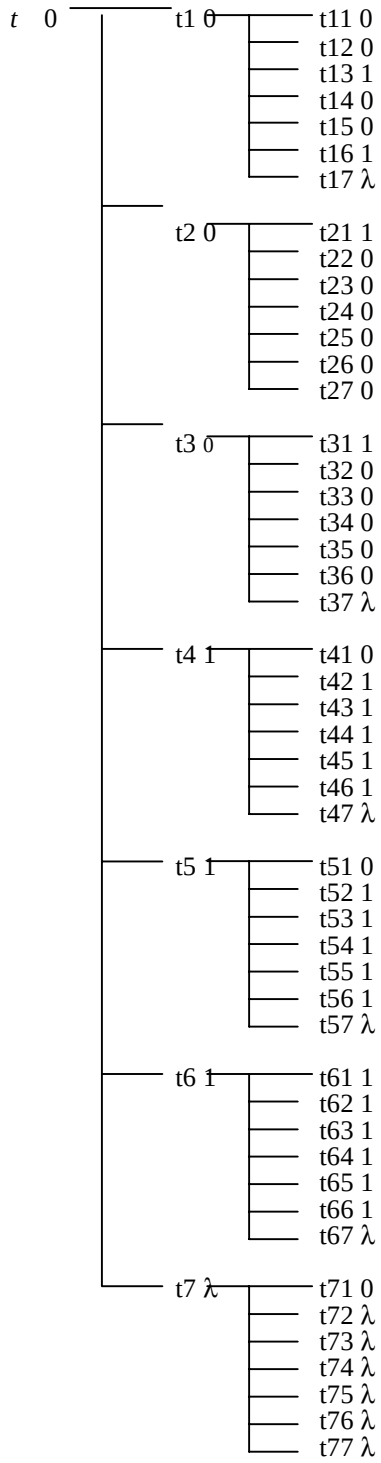
	level1 root	Using function VMAJ on received values
Healthy processor P_2	0	← (0,0,0,λ)
Healthy processor P_3	0	← (0,0, λ,0
Healthy processor P_4	1	← (1,1,1,λ)
Healthy processor P_5	1	← (1,1,1,1)
Healthy processor P_6	1	← (1,1,1,λ)
(mg-tree)		From a faulty transmitter by RFC

(b) The mg-tree of each processor in the 1st round of the message exchange phase

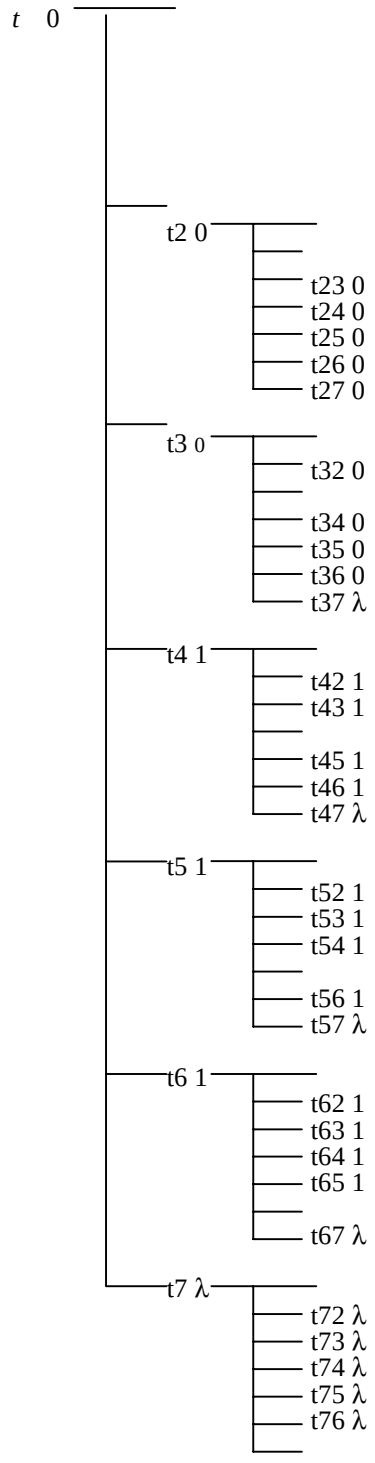
level 1 root	level 2		Using function VMAJ on received values	
t 0	t1 1	←	(1,0,λ,1)	from P_1
	t2 0	←	(0,0,0,λ)	from P_2
	t3 0	←	(0,0,λ,1)	from P_3
	t4 1	←	(0,1,1,λ)	from P_4
	t5 1	←	(1,1,1,λ)	from P_5
	t6 1	←	(0,1,λ,1)	from P_6
	t7 λ	←	(λ,λ,λ,0)	from P_7
(mg-tree)			(received messages from RFCs)	

(c) The mg-tree of processor P_2 in the 2nd round of the message exchange phase

Figure 5. An example of reaching a common agreement in an UNet (Cont'd.)

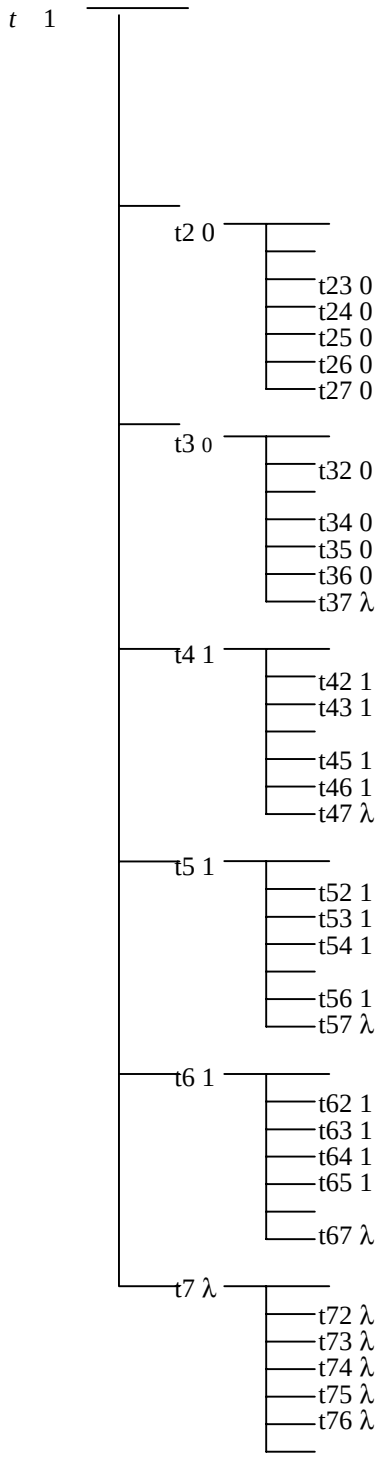


(d) The mg-tree of P_2 in the 3rd round of the message exchange phase



(e) The ic-tree of P_2

Figure 5. An example of reaching a common agreement in an UNet (Cont'd.)



(f) Vote(*t*)=1 for P_2

By step 2 in the decision making phase, the function VOTE value 1 is chosen

Figure 5. An example of reaching a common agreement in an UNet

4. THE CORRECTNESS AND COMPLEXITY OF UAP

The following lemmas and theorems are used to prove the correctness and complexity of our protocol.

Correctness of UAP

To prove the correctness of our protocol, a vertex α is called common [11] if each healthy processor has the same value for α. That is, if vertex α is common, then the value stored in vertex α of each healthy processor's mg-tree or ic-tree is identical. When each healthy processor has the common initial value of the transmitter processor in the root of its ic-tree, if the root t of the ic-tree of a healthy processor is common and the initial value received from the transmitter processor is stored in the root of the tree structure, then an agreement is reached because the root is common. Thus, the constraints, (Agreement) and (Validity), can be rewritten as:

(Agreement'): Root t is common, and

(Validity'): VOTE(t) = v_t for each healthy processor, if the transmitter processor is healthy.

To prove that a vertex is common, the term common frontier [11] is defined as follows: When every root-to-leaf path of the tree (an mg-tree or an ic-tree) contains a common vertex, the collection of the common vertices forms a common frontier. In other words, every healthy processor has the same messages collected in the common frontier if a common frontier does exist in a healthy processor's tree structure (ms-tree or ic-tree); subsequently, using the same majority voting function to compute the root value of the tree structure, every healthy processor can compute the same root value because the same input (the same collected messages in the common frontier) and the same computing function will cause the same output (the root value).

Since UAP can solve the BA problem, the correctness of UAP should be examined by the following two terms.

(1) Correct vertex: Vertex αi of a tree is qualified as a correct vertex if processor P_i (the last processor name in vertex αi's processor name list) is healthy. In other words, a correct vertex is a place to store the value received from a healthy processor.

(2) True value: For a correct vertex αi in the tree of a healthy processor P_j, val(αi) is the true value of vertex αi. In other words, the stored value is the true value.

According to the definition of a correct vertex, its stored value is received from a healthy processor, and

a healthy processor always transmits the same value to all processors; therefore, the correct vertices of such an mg-tree are common. After turning the mg-tree into its corresponding ic-tree by deleting the vertices with repeated processor names, the values stored on the correct vertices of an ic-tree will be the same. As a result, all the correct vertices of an ic-tree are also in common. Again, by the definition of a correct vertex, a common frontier does exist in the ic-tree. Thus, the root can be proven to be a common vertex [(Agreement') is true] due to the existence of a common frontier, regardless of the correctness of the transmitter processor. An agreement on the root value can now be reached. Next, we need to check the validity of (Validity'). When the transmitter processor is failed, (Validity') is true, and the reason is that the proposition [(P➔Q)] means (NOT(P) OR Q), hence (NOT(P) OR Q) or (P➔Q) is true when P is false, where P implies "the transmitter processor is healthy" and (P➔Q) implies (Validity's) [1]. Conversely, root *t* is a correct vertex by the definition of a correct vertex if the transmitter processor is healthy. If all the correct vertices' true values can be computed by UAP, then the true value of the root can also be computed because the root is a correct vertex. By definition, the true value of the root is the initial value of the transmitter processor if the transmitter processor is healthy. In short, each healthy processor's root value is the initial value of the transmitter processor if the transmitter processor is healthy; therefore, (Validity') is true when the transmitter processor is healthy. Since (Agreement') and (Validity') are both true no matter whether the transmitter processor is healthy or failed, the BA problem is solved.

Lemma 1: The destination processor can receive messages from healthy sender processors by using RFC, if $c>2m+d$.

Proof: Healthy sender processor sends *c* copies of a message to destination processor. In the worst case, a healthy destination processor can receive $c-d$ messages transmitted by a healthy sender processor because dormant fault components can be detected. Since $c-d>2m$, a healthy destination processor can confirm the message from the sender processor by taking majority value from the same sender processor.

Lemma 2: The healthy receiver processor can detect dormant faults.

Proof: The healthy receiver processor can detect dormant faults if the protocol appropriately encodes a transmitted message by using either the Non-Return-to-Zero code or the Manchester code [12] before transmission.

Theorem 1. The healthy destination processor can remove faulty influences from dormant faulty processors, if $c>2m+d$.

Proof: By Lemma 1 and Lemma 2, the theorem is proved.

Lemma 3. All correct vertices of an ic-tree are common.

Proof: After reorganization, no repeated vertices are in an ic-tree. At level f_p+1 or above, the correct vertex α has at least $2f_p+1$ children, out of which at least f_p+1 children are correct. The true values of these f_p+1 correct vertices are in common, and the majority value of vertex α is common. The correct vertex α is common in the ic-tree if the level of α is less then f_p+1. As a result, all correct vertices of the ic-tree are common.

Lemma 4. The common frontier exists in the ic-tree.

Proof: There are f_p+1 vertices along each root-to-leaf path of an ic-tree in which the root is labeled by the transmitter name, and the others are labeled by a sequence of processor names. Since at most f_p processors can be failed, there is at least one correct vertex along each root-to-leaf path of the ic-tree. By Lemma 3, the correct vertex is common, and the common frontier exists in each healthy processor's ic-tree.

Lemma 5. Let α be a vertex, α is common if there is a common frontier in the subtree rooted at α.

Proof: If the height of α is 0 and the common frontier (α itself) exists, then α is common. If the height of α is r, the children of α are all in common under the induction hypothesis with the height of the children being r-1.

Corollary 1: The root is common if the common frontier exists in the ic-tree.

Theorem 2: The root of a healthy processor's ic-tree is common.

Proof: By Lemma 3, Lemma 4, Lemma 5 and Corollary 1, the theorem is proved.

Theorem 3: Protocol UAP solves the BA problem in an UNet.

Proof: To prove the theorem, UAP must meet the constraints (Agreement') and (Validity')

(Agreement'): Root t is common.

By Theorem 2, (Agreement') is satisfied.

(Validity'): VOTE(t) = v for all healthy processors, if the initial value of the transmitter is v_t say $v=v_t$

Most processors are healthy. The value of the correct vertices for all the healthy processors' mg-trees is *v*. When the mg-tree is turned into an ic-tree, the correct vertices still exist. As a result, each correct vertex of the ic-tree is common (Lemma 3), and its true value is *v*. By Theorem 2, this root is common. The computed value VOTE(*t*) = *v* is stored in the root for all the healthy processors. Therefore, (Validity') is satisfied.

Complexity of UAP

The complexity of UAP is defined in terms of 1) the minimum number of rounds and 2) the maximum number of faulty components allowed.

Theorem 4: UAP requires f_p +1 rounds to solve the BA problem with dual failure mode in an UNet if $n > \lfloor (n-1)/3 \rfloor + 2m + d$ and $c > 2m+d$, where $f_p \leqq \lfloor (n-1)/3 \rfloor$, and f_p +1 is the minimum number of rounds of message exchange.

Proof: The message passing is required in the message exchange phase only. In the message exchange phase, UAP requires f_p +1 rounds, and no rounds are required during the decision making phase; therefore, UAP requires f_p +1 rounds, which is the minimum number of rounds of message exchange [6].

Theorem 5: The total number of faulty components allowed by UAP is *m* malicious faulty processors and *d* dormant faulty processors, where $n > \lfloor (n-1)/3 \rfloor + 2m + d$ and $c > 2m+d$.

Proof: A protocol for the BA problem with dual failure mode does exist if the constraints on failures, namely $n > \lfloor (n-1)/3 \rfloor + 2m + d$ and $c > 2m+d$, hold. Otherwise, an agreement cannot be reached. If $m+d$ is not the maximum number of faulty processors there can be, then other constraints on failures should exist, namely $n \leqq \lfloor (n-1)/3 \rfloor + 2m + d$ or $c \leqq 2m+d$. However, this contradicts Siu et al.[10]. Thus, the theorem is proven.

5. CONCLUSION

In the literature concerning the BA problem, researchers tend to assume each processor in the network has the common knowledge of the network structure, of the total number of processors in the network, as

well as of the connection state of each processor [2,4,5,6,7,8,9,10,11]. However, in the real world, each processor may not know the connection state of each processor. So, in this study, we loosen the constraint to revisit the BA problem. The new assumption under which the BA problem is to be solved is that each processor may not know the connection state of the others.

Many previous protocols now cannot solve the BA problem any more under the new assumption. In Table 1, we have summarized the application domain of various protocols. For example, Siu et al. [9] have used FTVC to transmit messages in the message exchange phase; however, without the common knowledge of the network structure and the connection state of each processor, then FTVC becomes useless. Under such circumstances, our UAP can still solve the BA problem. Therefore, UAP is more practical and applicable than the existing protocols when the processors do not know the network structure or the connection state.

In this study, we have proposed a new protocol UAP, which can solve the BA problem with dual failure mode for processors in a UNet. UAP can tolerate the maximum number of faulty components allowed while making all the healthy processors reach a common agreement at the cost of f_p +1 rounds of message exchange.

Table 1. The application domain of various protocols.

	Network topology			
	FCN	BCN	GCN	UNet
Dasgupta [2]	◆			
D.Dolev [4]	◆			
Lamport et al. [7]	◆			
Meyer and Pradhan [8]	◆	◆		
Siu et al.[9]	◆	◆	◆	
Wang et al. [11]	◆	◆	◆	
UAP	◆	◆	◆	◆

6. REFERENCES

[1] D. R. Broug, Logic Programming: New Frontiers, Boston Dorecht :Kluwer Academic, 1992.

[2] P. Dasgupta, “Agreement Under Faulty Interfaces,” *Information Processing Letters*, 65, pp.125-129, 1998.

[3] N. Deo, Graph Theory with Applications to Engineering and Computer Science, Englewood Cliffs, N.

J.:Prentice-Hall, 1974.

[4] D. Dolev, “The Byzantine Generals Strike Again,” *Journal of Algorithms,* vol. 3, no. 1, pp. 14-30,1982.

[5] D. Dolev, and R. Reischuk, “Bounds on Information Exchange for Byzantine Agreement,” *Journal of ACM*, vol. 32, no. 1, pp. 191-204, January 1985.

[6] M. Fisher, and N. Lynch, “A Lower Bound for the Assure Interactive Consistency,” *Information Processing Letters*, vol. 14, no. 4, pp. 183-186, June 1982.

[7] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382-401, July 1982.

[8] F.J. Meyer and D.K. Pradhan, “Consensus with Dual Failure Modes,” *IEEE Trans. Parallel and Distributed Systems*, vol. 2, no. 2, pp. 214-222, April 1991.

[9] H.S. Siu, Y.H. Chin, W.P. Yang, “Byzantine Agreement in the Presence of Mixed Faults on Processors and Links,” *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 4, pp. 980-986, April 1998.

[10] H. S. Siu, Y.H. Chin, W.P. Yang, “A Note on Consensus on Dual Failure Modes,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 225-229, March 1996.

[11] S.C. Wang, Y.H. Chin, and K.Q. Yan, “Byzantine Agreement in a Generalized Connected Network,” *IEEE Trans. on Parallel and Distributed System*, vol. 6, no. 4, pp. 420-427, April 1995.

[12] K.Q. Yan, S.C. Wang and Y.H. Chin, “Consensus Under Unreliable Transmission,” *Information Processing Letters*, vol. 69, pp.243-248, March 1999.

APPENDIX

I. The Message Gathering Tree (mg-tree)

The structure of an mg-tree is in Figure 5(d). Each healthy processor maintains such an mg-tree during the execution of UAP. At the first round, transmitter processor P_1 uses RFC to transmit its initial value to the other processors. We assume that each receiver processor can always identify the sender of a message. When a healthy processor receives the message sent from the transmitter processor, it stores the received value, denoted as *val*(*t*), at the root of its mg-tree as shown in Figure 5(b). At the second round, each processor uses RFC to transmit the root value of its mg-tree to the other processors. If processor P_1 sends message *val*(t) to

P_2, then processor P_2 stores the value of the VMAJ function of the received messages from processor P_1, denoted as *val*(t1), in vertex t1 of its mg-tree. Similarly, if processor P_2 sends message *val*(t1) to P_1, then the value of the VMAJ function is *val*(t12) and stored in vertex t12 of processor P_1's mg-tree as presented in Figure 5(d). Generally speaking, message *val*(t1….g), stored in the vertex t1….g of an mg-tree, implies that the message just received was sent through the transmitter processor, processor P_1,……, processor P_g, where processor P_g is the latest processor to pass the message. When a message is transmitted through a processor more than once, the name of the processor will be repeated correspondingly. For instance, the appearance of message *val*(t11) in vertex t11 in Figure 5(d) indicates that the message is sent from t to processor P_1 and to somewhere else and then to P_1 again; therefore, processor name P_1 appears twice in vertex name t11.

In summary, the root of an mg-tree is always named *t* to denote that the stored message is sent from the transmitter processor at the first round, and the vertex of an mg-tree is labeled with a list of processor names. The processor name list contains the names of the processors through which the stored message was transferred.

II. The Information Collecting Tree (ic-tree)

An ic-tree is reorganized from a corresponding mg-tree by removing the vertices with repeated processor names in order to avoid the repeated influences from faulty processors in an ic-tree. In Figure 5(e), there is an example of an ic-tree created by deleting the repeated processors name of the original mg-tree.