IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 6, 2019, accepted July 1, 2019, date of publication July 4, 2019, date of current version July 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2926888

A Flexible Consensus Protocol for Distributed Systems

CHIEN-FU CHENG[1], (Member, IEEE), AND KUO-TANG TSAI[2]
[1]Department of Computer Science and Information Engineering, Tamkang University, New Taipei 25137, Taiwan
[2]AsusTek Computer Inc., Taipei 11259, Taiwan

Corresponding author: Chien-Fu Cheng (cfcheng@mail.tku.edu.tw)

The work of C.-F. Cheng was supported in part by the Ministry of Science and Technology of Taiwan, under Grant MOST 107-2221-E-032-006-MY2.

ABSTRACT This paper presents a new type of Consensus problem named the Consensus (n, m) with alternative plans, where n denotes the total number of processors in the network, m is the number of processors with an initial value, $n \geq 4$ and $1 \leq m \leq n$. Compared to the traditional Consensus problem, the Consensus (n, m) problem with alternative plans has two major features. First, each processor is no longer required to propose an initial value. It can flexibly choose to propose or not propose an initial value. This feature allows the Consensus problem to be flexibly applied in many new real-world applications of the distributed system. Second, the proposed protocol ensures that all correct processors always agree on a good plan from a correct processor and never on a bad plan. Compared to solutions of the traditional Consensus problem, which does not guarantee that all correct processors agree on a good plan, this feature ensures the rationality of the Consensus value. In other words, by solving the Consensus (n, m) problem with alternative plans, the fault tolerance and reliability of distributed systems can be improved.

INDEX TERMS Distributed systems, fault tolerance, reliability, Byzantine agreement, Consensus problem.

I. INTRODUCTION

Adistributed system is a software system where its components distributed across networked computers communicate and coordinate their actions by exchanging messages [1]. However, in the event of hardware error, software error or hacker attack, these components may have various types of errors, which will hamper the distributed system from accomplishing tasks requested by users [2], [3]. Therefore, the reliability and fault tolerance of distributed computer systems is an important topic [4]–[8].

A. BYZANTINE AGREEMENT

In fault-tolerant distributed systems, one of the well-known problems is the Byzantine Agreement (BA) problem [9], [10]. With Byzantine fault tolerance, the system is able to defend against Byzantine failure, which occurs when components of the system fail in arbitrary ways [11]. The applications of BA include the authentication protocol [12], the replicated file system [13], the leader election problem [14], [15], and etc.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Li.

In 1982, Lamport, Shostak and Pease [9] were the first scholars to pay attention to the BA problem. Lamport et al. defined the BA problem as follows: (1) A distributed system comprises n processors with a maximum number of Byzantine processors f_b, where $n \geq 4$ and $f_b \leq \lfloor (n-1)/3 \rfloor$; (2) Processors can communicate with each other directly; (3) Each processor can be uniquely identified; (4) One of the processors is assigned to be the commander that holds an initial value v_c; (5) The commander first sends the initial value v_c to other processors. On receipt of the value v_c, each processor exchanges the received value with other processors. After $\lfloor (n-1)/3 \rfloor + 1$ rounds of message exchange, an agreement value can be obtained. The term ''round'' denotes the interval of a message exchange between any two processors [16]. In the above definition, the first condition clearly indicates that the maximum allowed number of faulty processors is $\lfloor (n-1)/3 \rfloor$; the second indicates that the network topology is a fully connected network; the third is that the identification of each processor cannot be falsified; the fourth indicates that there is only one commander in the BA problem; The fifth indicates that each processor can reach a common agreement value after $\lfloor (n-1)/3 \rfloor + 1$ rounds of message exchange.

The difficulty of the BA problem lies in that there can be $\lfloor (n-1)/3 \rfloor$ Byzantine processors at most in the network. The worst case is that the commander is a Byzantine processor, so the Byzantine commander may send inconsistent v_c to different processors in the network. Any protocol designed to solve the BA problem must satisfy three requirements as follows [9], [10], [17]:

- BA_Agreement: All the correct processors agree on a common value.
- BA_Validity: If the commander processor is correct, then all the correct processors agree on the initial value sent by the commander processor.
- BA_Termination: All the correct processors eventually decide.

A review of related works of the BA problem published in recent years is provided as follows [18], [19]. With the continuous development of wireless networking, networks have evolved from being static to being dynamic. However, previous BA protocols can only be applied to static networks, such as fully connected networks [9], [10], broadcast networks [20], and non-fully connected networks [21]. Hence, Cheng and Tsai [18] propose the Eventual Byzantine Agreement Protocol for dynamic network with Byzantine faulty processors (EBA_{dynP}) as a solution for the BA problem in dynamic networks. The feature of the EBA_{dynP} protocol is that it can not only solve the BA problem in dynamic networks but also dynamically alter the number of rounds of message exchange depending on the interference of faulty processors.

On the other hand, it is assumed in the BA problem defined by Lamport et al. [9] that each processor in the distributed system can be uniquely identified. Hence, most of the BA algorithms assume that each processor has a unique identity. However, Delporte-Gallet et al. [19] argue that assuming each processor has a unique and unforgeable identifier might be a too restrictive assumption in practice. Therefore, they reexamine the BA problem in synchronous systems with homonyms. In their study, different processors may have the same authenticated identifier (a system of n processors sharing a set of l identifiers, where $1 \leq l \leq n$). They proposed a protocol named the Synchronous Byzantine Agreement Algorithm with Distribution $(n_1, .., n_l)$ (SBAAD in short). The SBAAD protocol can solve the BA problem with homonyms in synchronous systems after $2(\lfloor (n-1)/3 \rfloor + 1)$ rounds of message exchange.

B. CONSENSUS (BINARY-VALUED / MULTI-VALUED)

The Consensus problem is another well-known problem in fault-tolerant distributed systems. As mentioned earlier, the feature of the BA problem is that only one processor has the initial value. The difference between the BA problem and the Consensus problem is that every processor has an initial value of its own in the Consensus problem. There are two main types of the Consensus problem, namely the Binary-Valued Consensus (BVC) and the Multi-Valued Consensus (MVC) [22]–[26]. In the BVC problem, the agreement value is either 0 or 1. Any protocol designed to solve the BVC problem must satisfy three requirements as follows [23], [25]:

- BVC_Agreement: All the correct processors agree on a common value.
- BVC_Validity: If the initial values of all the correct processors are v, then all the correct processors shall agree on v.
- BVC_Termination: All the correct processors eventually decide.

In terms of definition, the MVC problem is similar to the SVC problem, except in that processors can propose values with arbitrary length $v \in V$, where V is the domain of values that can be proposed. The protocol can decide one of the proposed values or a default value $\perp \notin V$. Any protocol designed for the MVC problem shall satisfy five requirements as follows [22], [24], [26]:

- MVC_Agreement: All the correct processors agree on a common value.
- MVC_Validity1: If the initial values of all the correct processors are v, then all the correct processors shall agree on v.
- MVC_Validity2: If a correct processor decides v, then v has been proposed by some processors or $v = \perp$.
- MVC_Validity3: If a value v is proposed by only Byzantine processors, then no correct processor agrees on v.
- MVC_Termination: All the correct processors eventually decide.

Some related works of the Consensus problem published in recent years [25], [27] are introduced as follows. The traditional Consensus protocols can tolerate $\lfloor (n-1)/3 \rfloor$ faulty processors in a network after $\lfloor (n-1)/3 \rfloor + 1$ rounds of message exchange [22], [23]. Due to the continuous improvement of software and hardware technologies, the actual number of faulty processors (f_{act}) in a network has become very small (in practice, $f_{act} << \lfloor (n-1)/3 \rfloor$). Nevertheless, the traditional Consensus protocols still need $\lfloor (n-1)/3 \rfloor + 1$ rounds of message exchange to solve the Consensus problem even if there is no faulty processor. Hence, Cheng and Tsai [25] propose the Recursive Byzantine-Resilient (RBR) protocol to address this issue. In this paper, the authors do not address how to let the protocol tolerate a greater number of faulty processors but focus on how to drastically reduce its space complexity and time complexity by slashing the number of faulty processors that can be tolerated. Moreover, they also discuss how to achieve tolerance of more faulty processors through iterative execution of the RBR protocol.

On the other hand, several variants of the Consensus problem have been proposed for various applications. Using the approximate Consensus protocol to solve the clock synchronization problem in mobile ad hoc networks is an example of these variants. Clock synchronization is crucial for many applications in mobile ad hoc networks. For instance, sensors must perform clock synchronization before

TABLE 1. Four variations of the problem.

	Equal BP sets	Possibly different BP sets
Equal GP sets	[1] $\forall i,j$, (correct(P_i) $\wedge$ correct(P_j)) => (GP_i=GP_j $\wedge$ BP_i=BP_j)	[2] $\forall i,j$, (correct(P_i) $\wedge$ correct(P_j)) => (GP_i=GP_j)
Possibly different GP sets	[3] $\forall i,j$, (correct(P_i) $\wedge$ correct(P_j)) => (BP_i=BP_j)	[4] $\forall i,j$, (correct(P_i) $\wedge$ correct(P_j)) => No restriction on the relations between GP_i, GP_j, BP_i, and BP_j

sleep/wake scheduling. Monitoring tasks may also require identification of a total order among the events observed on different sensors. However, mobile processors move in a random walk pattern in mobile ad hoc networks. Hence, previous approximate Consensus protocol cannot be applied in mobile ad hoc networks. To address this issue, Li et al. [27] attempt to solve the clock synchronization problem in mobile ad hoc networks using an approximate Consensus protocol. They propose a protocol named Clock Synchronization (CS). This CS protocol can solve the clock synchronization problem in mobile ad hoc networks containing Byzantine mobile processors.

C. ALTERNATIVE PLANS

In traditional agreement related problems (i.e. BA and Consensus problems), all the correct processors shall eventually accept a commonly agreed value. However, whether the agreement (plan) is good or bad is not considered in previous research. Therefore, Correia et al. [22] propose a variant of the agreement problem called the Byzantine Generals with Alternative Plans problem (BGAP). In BGAP, processors can devise several good plans and a set of bad plans. The protocol for BGAP is to make all the correct processors agree on a good plan that is proposed by one of them.

Four variations of BGAP have been noted, and they differ by the restrictions to the relation between the processors' good plans (GP set) and bad plans (BP set). The four variations of the problem are explained in Table 1. According to Correia et al., Variation 4 does not set any restriction to the relation between good plans and bad plans, so it cannot be solved. Protocols designed for BGAP shall satisfy four requirements as follows [22]:

- BGAP_Agreement: No two correct processors decide differently.
- BGAP_Validity1: If there is a value v such that for any correct processor P_i, $v \in GP_i$, then any correct processor that decides will decide a value v' such that $v' \in GP_j$ for a correct processor P_j.
- BGAP_Validity2: No correct processor P_i decides a value v if there is a correct processor P_j with $v \in BP_j$.
- BVC_Termination: All the correct processors eventually decide.

D. RELIABLE CHANNELS

In most BA and Consensus problems, processors are fully connected by reliable channels. This ensures that messages from the sender will be eventually received by the intended receiver without being tampered [9], [10]. However, in reliable channels, the Byzantine sender can still send inconsistent messages to different receivers. For example, the Byzantine commander may send an attack instruction to some of the generals and send a retract instruction to the others. Therefore, Correia et al. [22] use a reliable broadcast protocol to ensure that different messages with the same identifier cannot be delivered. Using this reliable broadcast protocol can avoid Byzantine senders from sending incongruous messages to different processors. This reliable broadcast protocol has three requirements as follows [28]:

- RB Validity: If a correct processor broadcasts a message *Msg*, then some correct processors will eventually receive *Msg*.
- RB Agreement: If a correct processor receives a message, then all the correct processors will ultimately receive *Msg*.
- RB Integrity: For any identifier *ID*, each correct processor receives only one message *Msg* with identifier *ID*; if the sender of message *Msg* is correct, then *Msg* has been previously broadcast by the sender of *Msg*.

E. CONTRIBUTION (CONSENSUS (n,m) PROBLEM WITH ALTERNATIVE PLANS)

As mentioned previously, only one processor can serve as the commander in the BA problem. In other words, only one processor is allowed to propose the suggestion (i.e. only one processor has the initial value). In contrast to the BA problem, all processors in the Consensus problem can serve as the commander. That is, each processor must propose its suggestion (as mentioned in Section I.B). However, requesting all processors to propose suggestions or allowing only one processor to propose a suggestion is not flexible and practical for real-world applications. For example, demanding everyone to vote for someone and not abstain in the election is unreasonable. Therefore, the Consensus problem is reexamined in this paper. A new type of Consensus problem called Consensus (n, m) is defined, where n is the total number of processors in the network, m is the number of processors with an initial value, $n \geq 4$ and $1 \leq m \leq n$. Based on this definition, the traditional BA problem can be expressed as Consensus(n,1), and the traditional Consensus problem can be expressed as Consensus (n, n). The difference between the traditional BA/Consensus problem and the Consensus (n, n) problem is that the processors in the Consensus (n, n) problem can flexibly choose to propose or not propose an initial value. That is, the value of m can flexibly vary between $1 \sim n$. A greater m means there are more processors having an initial value. However, the value of m does not affect the number of rounds of message exchange and the number of messages that are required by the proposed protocol. This feature allows

the Consensus problem to be flexibly applied in many new real-world applications of the distributed system. Moreover, the proposed Consensus (n, m) problem will be combined with alternative plans and solve the Consensus (n, m) problem with the four variations in Table 1. Therefore, the Consensus (n, m) problem is also a variation of the MVC problem.

Correia et al. [22] point out that if the correct processors have possibly different good plans and possibly different bad plans (Variation 4 in Table 1), then the problem cannot be solved. In this paper, how to make all the correct processors in the network have the same set of good plans and the same set of bad plans is discussed. This is a problem of reducing Variation 4 to Variation 1. If Variation 4 can be reduced to Variation 1, Variation 4 of the Consensus (n, m) problem can be solved. In addition, the proposed Consensus (n, m) protocol does not take advantage of the reliable broadcast protocol to transmit messages in reliable channels. Therefore, Byzantine senders can send differing messages to different processors. Nevertheless, the proposed protocols can still solve the Consensus (n, m) with alternative plans problem without a reliable broadcast protocol in reliable channels. The main contributions of this paper are summarized below.

1) We define a new type of Consensus problem named the Consensus(n,m) with alternative plans. This problem features a relaxed number of processors that are allowed to have an initial value.

2) We integrate the concept of alternative plans into the proposed algorithm. We integrate the Plan_Consensus Protocol (PCP) to ensure that all correct processors obtain a consistent good plan set and a consistent bad plan set.

3) Under the influence from Byzantine processors, the proposed algorithm can still solve the Consensus(n,m) with alternative plans problem without using the reliable broadcast protocol.

This paper comprises six sections, and remainder is as follows. Section 2 describes the definitions and conditions for the proposed protocol. Section 3 describes the concept and approaches. Section 4 shows how the proposed protocol runs in an example problem. Section 5 evaluates the correctness and complexity. Finally, the conclusion is presented in the Section 6.

II. DEFINITIONS AND CONDITIONS

In this study, the Consensus (n, m) problem is discussed in an asynchronous network with oracle or a timing assumption [22] to circumvent the limitation of strictly asynchronous systems. Moreover, it is assumed that processors are fully connected by reliable channels. The definition of reliable channel is given in Section I.D. To begin with solving the Consensus (n, m) problem, the system model must be established and problem definition must be clearly defined.

A. SYSTEM MODEL

In the Consensus problem, the number of allowable faulty processors depends on the failure type of faulty processors as well as the total number of processors in the network. Common failure types of faulty processors include crash [29], omission [30] and Byzantine [31]. The most damaging type of failure is Byzantine failure. For instance, a Byzantine processor may send bogus messages, send messages at the wrong time or send different messages to different processors. Besides, Byzantine processors may work with other faulty processors to prevent correct processors from reaching a common agreement.

So, if the Consensus (n, m) problem can be solved with Byzantine processors, the Consensus (n, m) problem can also be solved when there are other types of failures. Therefore, the proposed protocol is designed to solve the Consensus (n, m) problem with Byzantine processors. Fischer et al. [16] show that when using oral messages to cope with f_b Byzantine processors, there must be at least $3f_b + 1$ processors. Therefore, $3f_b + 1$ is also a constraint on our system model. The assumptions and parameters of our model are listed as follows:

- Let N be the set of all the processors in the network and $|N| = n$, where n denotes the total number of processors in the network.
- Each processor can be uniquely identified, and the underlying network is an asynchronous network with reliable channels.
- All messages are transmitted through reliable channels. Intermediate components cannot falsify any message from a sender to its receivers, but the Byzantine sender may send different messages to different processors.
- Let f_b be the maximum number of Byzantine processors allowed in the network, where $f_b = \lfloor (n-1)/3 \rfloor$. These faulty processors cannot break down a workable network.
- Correct processors do not know the faulty status of other processors in the network.
- Processors can propose values with arbitrary length $v \in V$, where V is the domain of values that can be proposed.

B. PROBLEM DEFINITION

In the Consensus (n, m) problem with alternative plans, there are a total of n ($n \geq 4$) processors in the network, where m ($1 \leq m \leq n$) processors have their own initial values. The maximum number of Byzantine processors in the network is f_b ($f_b \leq \lfloor (n-1)/3 \rfloor$). During the execution of the Consensus (n, m) protocol, each of the m processors shall propose its initial value (i.e. suggestion) selected from its good plans set. The goal of the Consensus (n, m) protocol is to make all the correct processors agree on a common value with no interference by Byzantine processors. The common value is selected from good plans. That is, the protocol for the Consensus (n, m) problem with alternative plans shall satisfy the following requirements:

- Agreement: All the correct processors agree on a common value.
- Validity1: If there is a value v such that for any correct processor P_i, $v \in GP_i$, then any correct processor

Protocol: Plan Consensus Protocol (PCP) //for each processor P_i
Input: IB_i, IG_i, *ID*
Output: GP_i and BP_i

Initialization:
1: Set GP ← *null*;
2: Set BP ← *null*;
3: Vector *TemGP* ← *null*;
4: Vector *TemBP* ← *null*;
5: active task(Consistent_Plan, Msg_Record) ;

Task Consistent_Plan:
6: **for** $v \in IB_i$ **do**
7: *BA*(BP, *v*, P_i);
8: **end**
9: **for** $v \in IG_i$ **do**
10: *BA*(GP, *v*, P_i);
11: **end**
12: **wait until** ($\forall j$, $1 \le j \le n$: IB_j have been received by BA function)
13: **for** $v \in V$ **do**
14: **if** $|TemBP|_v \ge \lfloor (n\text{-}1)/3 \rfloor + 1$ **then**
15: BP=BP$\cup\{v\}$;
16: **end**
17: **wait until** ($\forall j$, $1 \le j \le n$: IG_j have been received by BA function)
18: **for** $v \in V$ **do**
19: **if** $|TemGP|_v \ge \lfloor (n\text{-}1)/3 \rfloor + 1$ **then**
20: GP=GP$\cup\{v\}$;
21: **end**
22: GP= GP - BP;
23: **return** (GP, BP);

Task Msg_Record:
24: **when** the value of BA function is returned **do**
25: **if** (*TYPE*, *APlan*, *ID*) with *TYPE=BP* **then**
26: **add** *APlan* into *TemBP*
27: **if** (*TYPE*, *APlan*, *ID*) with *TYPE=GP* **then**
28: **add** *APlan* into *TemGP*

FIGURE 1. Plan_Consensus Protocol (PCP) [24].

that decides will decide a value $v' \in GP_j$, for correct processor P_j.

- Validity2: No correct processor P_i decides a value v if there is a correct processor P_j with $v \in BP_j$
- Termination: All the correct processors eventually decide.

We set the object function of the Consensus (n, m) problem with alternative plans as expressed in Equation (1), where v_i is the consensus value that processor P_i decides. Equation (2) is used to limit the number of processors and its relationship with the number of Byzantine processors. Equation (3) and Equation (4) are used to ensure that all the correct processors have the same set of good plans and the same set of bad plans. Equation (5) is used to limit that the consensus value comes from the good plan set.

$$v_i = v_j, \quad \forall \text{ correct processor } P_i,\ P_j \in N \text{ where } i \neq j \tag{1}$$

subject

$$f_b \le t, \quad \text{where } t = \lfloor (n-1)/3 \rfloor \tag{2}$$

$$GP_i = GP_j, \quad \forall \text{ correct processor } P_i,\ P_j \in N \text{ where } i \neq j \tag{3}$$

$$BP_i = BP_j, \quad \forall \text{ correct processor } P_i,\ P_j \in N \text{ where } i \neq j \tag{4}$$

$$v_i \in GP_i, \quad \forall \text{ correct processor } P_i \in N \tag{5}$$

III. CONCEPT AND APPROACHES

To solve the Consensus (n, m) problem with alternative plans in a network containing Byzantine processors, the proposed protocol must be able to cope with the four variations of the Consensus (n, m) problem as shown in Table 1. As mentioned earlier, Variation 4 does not set any restriction to the relation between good plans and bad plans, so it cannot be solved (Correia et al. [22]).

To tackle this issue, we propose a protocol that makes all the correct processors have the same set of good plans and the same set of bad plans (to reduce Variation 4, 3 and 2 to Variation 1) [24]. This protocol is called Plan_Consensus Protocol (PCP). The pseudocode of the PCP protocol is shown in Fig. 1. Readers interested in understanding how the PCP works are advised to check out the reference [24].

If the alternative plans model [22] is not considered in the problem, all correct processors can still reach a common consensus value through the proposed ECP_{AP} protocol. That is, the proposedECP_{AP} protocol can deal with the raw consensus problem. However, without the consideration of the alternative plans model, the proposed ECP_{AP} protocol cannot ensure that the consensus value comes from the good plan set. In the following subsection, how to solve the Consensus (n, m) problem with alternative plans in a network with Byzantine processors will be discussed.

A. ELASTIC CONSENSUS(n,m) PROTOCOL WITH ALTERNATIVE PLANS (ECP_{AP})

The proposed protocol for the Consensus (n, m) problem with alternative plans in a network with Byzantine processors is called the Elastic Consensus (n, m) Protocol with Alternative Plans (ECP_{AP}). ECP_{AP} has two phases, including Msg_Collecting Phase and Dec_Making Phase. Msg_Collecting Phase is a phase in which all the processors collect messages from other processors. In ECP_{AP}, a round of message exchange is defined in the same way as in Fischer et al. [16], so the number of rounds of message exchange is also $t + 1$ in this phase. In the Msg_Collecting Phase, each processor will store the received messages into an ECP-tree. ECP-tree is conceptually similar to ic-tree [24]. The difference between ic-tree and ECP-tree is that ic-tree starts from Level 1 and ECP-tree starts from Level 0. After $t + 1$ rounds of message exchange, all the processors will

Protocol: Elastic Consensus(n,m) Protocol with Alternative Plans (ECP$_{AP}$) //for each processor P_i and v_i is the initial value of P_i, $v_i \in GP \cup \{\eta\}$.
Input: v_i
Output: *ConsensusPlan*

```
/*Msg_Collecting Phase */
1:   crt_vertex(0, root, v_i);
2:   for q∈N do /*Round = 1*/
3:     begin
4:       sndpkt = encp(0, θ);
5:       send_RC(q, sndpkt);
6:     end
7:   for q∈N do
8:     begin
9:       rcv_RC(q, sndpkt);
10:      crt_vertex(1, q, v_q);
11:    end
12:  for r=2 to t+1 do /*Round 2 to Round t+1 */
13:    begin
14:      sndpkt=encp(r-1, θ);
15:      for q∈N do send_RC(q, sndpkt);
16:    end
17:  for q∈N do
18:    begin
19:      rcv_RC(q, sndpkt);
20:      for σ∈sndpkt do
21:        begin
22:          if chk_rpt(σq) = legal then
23:            v= val(σ);
24:            if v= η^j then
25:              v = η^{j+1}(0≤j≤⌊(n-1)/3⌋) ;
26:            crt_vertex(r, σq, v);
27:        end
28:    end
/*Dec_Making Phase */
29:  ConsensusPlan= vote_AC(Lv_t+1 ECP-tree);
30:  val(root)=ConsensusPlan;
31:  return val(root);
```

FIGURE 2. Elastic Consensus (n, m) Protocol with Alternative Plans (ECP$_{AP}$).

enter the Dec_Making Phase. In this phase, influence from Byzantine processors will be removed by using the $vote_{AC}$ function. A detailed explanation of $vote_{AC}$ function is provided in Section III.A.2. The formal description of ECP$_{AP}$ is shown in Fig. 2.

As mentioned in Section I.D, a reliable broadcast protocol can avoid Byzantine senders to send inconsistent messages to different processors. In this study, ECP$_{AP}$ does not take advantage of reliable broadcast protocol. Even though Byzantine senders may send inconsistent messages to different processors, ECP$_{AP}$ can still solve the Consensus (n, m) problem with alternative plans in a network with Byzantine processors. This is because if the amount of Byzantine processors is smaller than f_b ($f_b = \lfloor (n-1)/3 \rfloor$, where n is the total number of processors), the impact of one Byzantine processor can be fixed after one round of message exchange [32]. Given at most f_b Byzantine processors in the network, after $\lfloor (n-1)/3 \rfloor + 1$ rounds of message exchange in the *Msg_Collecting Phase*, the impact of f_b Byzantine processors can be fixed. Using the $vote_{AC}$ *function* in the *Dec_Making Phase*, the impact of the Byzantine processors can be further excluded through a majority rules procedure (Theorem 2).

1) Msg_Collecting PHASE OF ECP$_{AP}$

The main purpose of the Msg_Collecting Phase is to enable all the processors to gather sufficient messages. In the Msg_Collecting Phase, each processor has to compute how many rounds of message exchange are needed first ($t+1$, where $t = t = \lfloor (n-1)/3 \rfloor$) and then construct an ECP-tree and set the value of root (Level 0) as its initial value (Line 1 of ECP$_{AP}$). Later, all processors begin to exchange messages. Each of them will send its initial value to all the processors in the network (Line 2-6 of ECP$_{AP}$), which will then store the obtained initial values in their ECP-trees (Level 1) respectively (Line 7-11 of ECP$_{AP}$). In the following rounds (Round r, $2 \leq r \leq t+1$), each processor will send the messages it has received in the previous round (Round $r-1$) to other processors and store the messages delivered from other processors at Level r of its ECP-tree (Line 12-28 of ECP$_{AP}$). This process continues until Round $t+1$ is completed.

In the Consensus (n,m) problem, some processors (n-m) may not have any suggestion (i.e. initial value). For processors whose initial value is *null*, η^0 will be used to represent their messages. In other words, value η is used to report the absence of suggestions. Besides, for processors to better identify that value η comes from a preceding processor, ECP$_{AP}$ has the following design: if a processor receives η^j ($0 \leq j \leq (\lfloor (n-1)/3 \rfloor)$ in Round $2 \sim t+1$ of message exchange, it will store η^{j+1} in its ECP-tree (Line 24-26 of ECP$_{AP}$). This process continues iteratively until Round $t+1$ is completed.

2) Dec_Making PHASE OF ECP$_{AP}$

The Dec_Making Phase enables all the correct processors to compute a common good plan. After the Msg_Collecting Phase, every processor has built an ECP-tree of $t+1$ levels. The correct processors will apply the $vote_{AC}$ function to their ECP-tree to compute a common good plan.

First of all, each processor will bring its ECP-tree into the $vote_{AC}$ function to derive the function values at the first level (Level 0) and uses this function to remove the influence from Byzantine processors. To put the matter simply, after bringing the messages stored at each leaf (Level $t+1$) into the $vote_{AC}$ function, a $vote_{AC}$ function value of each leaf (Level $t+1$) can be obtained. Later, the $vote_{AC}$ function value of each node at Level t can be computed based on the values of leaves (Level $t+1$) under the same parent node. The $vote_{AC}$ function value of each node is computed from the last level (Level $t+1$) to the first level (Level 0). Finally, each correct processor can get the same good plan by $vote_{AC}$ function (Line 29-31 of ECP$_{AP}$).

Function $vote_{AC}$
Input: α
Output: *FPlan*
1: **if** the α is a leaf **then**
2: return $val(\alpha)$; ---(1)
3: **else**
4: **if** $\#_\eta 0(\alpha) \geq 3\cdot(t\text{-}i\text{+}1)+[(n\text{-}1) \bmod 3]$ **then**
5: return $val(\alpha)$; ---(2)
6: **if** $maj(\alpha) = \eta^i$, where $1 \leq i \leq \lfloor (n\text{-}1)/3 \rfloor\text{-}1$ **then**
7: return η^{i-1}; ---(3)
8: **if** $maj(\alpha) = v$ and $v \in$ GP **then**
9: return v; ---(4)

*: Function $vote_{AC}$ only counts the non-η value(excluding the last level of the ECP-tree). Function $maj(\alpha)$ is used to compute the majority value of α's child nodes and $val(\alpha)$ is the value of the α. Function $\#_x(\alpha)$ is used to count the number of occurrences of x in α's child nodes.

FIGURE 3. $vote_{AC}$ **function.**

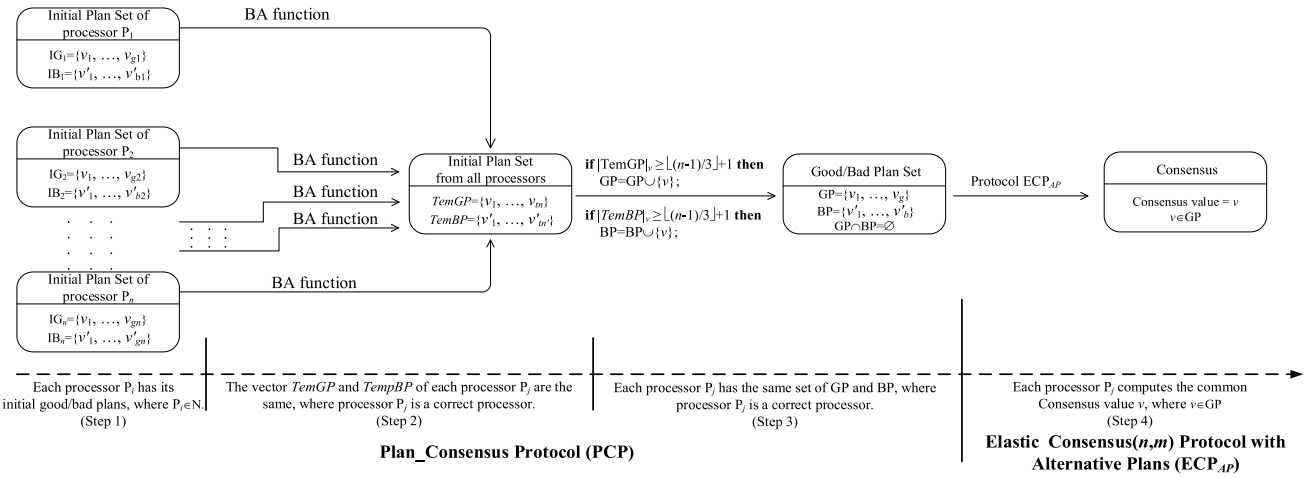


FIGURE 4. **The flow chart of solving the Consensus (*n*, *m*) problem with alternative plans.**

The $vote_{AC}$ function has four conditions as follows: (1) If vertex α is a leaf, there is only one value in vertex α. Thus, the majority value is the value of vertex α; (2) It is used to remove the influence from Byzantine processors. (3) It is used to identify the processors with no suggestion; (4) It is used to find the majority value. The formal description of $vote_{AC}$ function is shown in Fig. 3.

B. THE FLOW CHART OF THE APPROACH

The flow chart of the approach is shown in Fig. 4. In order to solve the Consensus (n, m) problem with alternative plans in a network with Byzantine processors, the PCP protocol is utilized to ensure that all the correct processors have the same set of good plans (GP) and the same set of bad plans (BP). As shown in the left of Fig. 4 (step 1), each processor P_i has a set of its Initial Good plans $IG_i = \{v_1, \ldots, v_{gi}\}$ and a set of its Initial Bad plans $IB_i = \{v'_1, \ldots, v'_{bi}\}$ in the beginning, where $IG_i \subseteq V$, $IB_i \subseteq V$, and $IG_i \cap IB_i = null$. Subsequently, the BA function ensures that all the correct processors get the same message (i.e. the same initial good/bad plan set from all the processors, Step 2 in Fig. 4). Then, using the following rule to compute the good/bad plan set: If plan v is recognized as a good plan by $\lfloor (n-1)/3 \rfloor + 1$ processors, plan v will be placed in the set of GP; in contrast, if plan v is recognized as a bad plan by $\lfloor (n-1)/3 \rfloor + 1$ processors, plan v will be placed in the set of BP. Hence, every correct processor can compute the same good/bad plan set because with the same input and the same computing rule, the same output will be obtained (Step 3 in Fig. 4). That is, the role of the PCP protocol is important. It is used to reduce the Variation 2, Variation 3 and Variation 4 to Variation 1 (Table 1). Finally, the proposed ECP_{AP} protocol is used to make all the correct processors agree on the same good plan (Step 4 in Fig. 4).

IV. AN EXAMPLE OF EXECUTING THE PROPOSED PROTOCOLS

This section shows how the proposed protocols work. This example is given as follows: A network consists of eight processors, with each processor P_i having some initial good plans (IG_i set) and initial bad plans (IB_i set), as shown in Fig. 5 and Table 2. Among these processors, P_3 and P_7 are Byzantine processors.

After executing the PCP protocol, all the correct processors obtain a consistent BP set and a consistent GP set, where $BP = \{\zeta, \psi\}$ and $GP = \{\tau, \rho, \omega\}$. How to use the ECP_{AP} protocol to allow all the correct processors to obtain a final

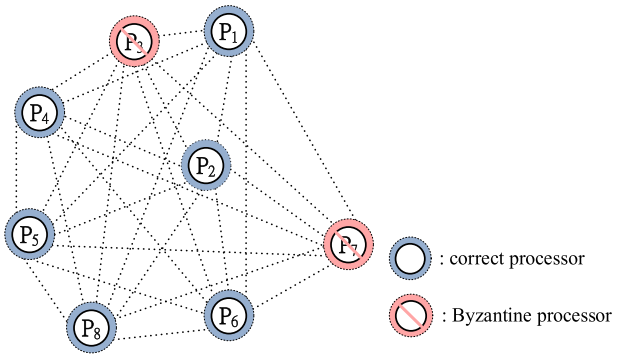


FIGURE 5. An example of the network with Byzantine processors.

TABLE 2. The IG set and IB set of each processor.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
IB set	$\{\zeta\}$	$\{\zeta\}$	$\{\tau\}$	$\{\zeta, \psi\}$	$\{\psi\}$	$\{\}$	$\{\tau\}$	$\{\zeta, \psi\}$
IG set	$\{\tau, \rho\}$	$\{\tau, \omega, \rho\}$	$\{\zeta\}$	$\{\omega, \rho\}$	$\{\tau, \omega, \rho, \varphi\}$	$\{\}$	$\{\zeta\}$	$\{\omega, \rho\}$

TABLE 3. The initial value of each processor.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Initial value	τ	τ	*arbitry*	ρ	*null*	*null*	*arbitry*	ω

TABLE 4. The initial value sends by Byzantine processors P_3 and P_7 in the 1st round.

receiver / sender	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_3	ρ	τ	---	τ	τ	ρ	---	ρ
P_7	ρ	τ	---	τ	τ	ρ	---	ρ

common value that comes from the GP set is explained as follows.

First of all, each processor will obtain a 0∼1 value from the GP set to be its initial value and store it in the root of its ECP-tree. Table 3 shows the initial value of each processor. P_5 and P_6 do not have a suggestion, so their initial values are set as *null*. P_3 and P_7 are Byzantine processors, so they may arbitrarily alter their initial values any time. In the worst case, these two processors conspire to prevent the correct processors from reaching a consensus. In this example, the ECP_{AP} protocol will execute 3 ($\lfloor(8-1)/3+1$) rounds of message exchange in the Msg_Collecting Phase. The operation of the ECP_{AP} protocol is explained from P_1's viewpoint.

In the first round of the Msg_Collecting Phase of ECP_{AP}, each processor sends its initial value to other processors in parallel. For processors whose initial value is *null*, η^0 will be used to represent their messages. Later, each processor will store the initial value from other processors in its ECP-tree. Assume that Byzantine processors P_3 and P_7 work together to send ρ to P_1, P_6 and P_8 and τ to P_2,P_4 and P_5, as shown in Table 4. After the first round of the Msg_Collecting Phase, P_1 will have an ECP-tree as shown in Fig. 6(a), and P_4 will have an ECP-tree as shown in Fig. 6(b).

In the second round of the Msg_Collecting Phase of ECP_{AP}, each processor will send the value it has stored at Level 1 of its ECP-tree to all the other processors and store the messages it has collected at Level 2 of its ECP-tree. Fig. 6(c) shows the ECP-tree of P_1 after 2 rounds of message exchange. Take P_4 in the second round of the Msg_Collecting Phase as an example. P_4 will send the messages it has collected at Level 1 to P_1 (i.e. val(P_1), val(P_2), val(P_3), val(P_4), val(P_5), val(P_6), val(P_7), val(P_8) in Fig. 6(b)). P_1 will stores the received message from processor P_4, denoted as val(P_1P_4), val(P_2P_4), val(P_3P_4), val(P_5P_4), val(P_6P_4), val(P_7P_4), val(P_8P_4) in vertex P_1P_4, P_2P_4, P_3P_4, P_5P_4, P_6P_4, P_7P_4, P_8P_4 of its ECP-tree (Fig. 6(c)). To help readers better understand the operation of the proposed protocol, we highlight the ECP-tree of P_1 after the 2nd round of Msg_Collecting Phase (Fig. 6(c)) that is received from P_4 (Fig. 6(b)).

It should be noted that if any processor receives η^j ($0 \le j \le (\lfloor(n-1)/3\rfloor)$ from other processors, it will change it into η^{j+1} as val(P_5P_1) $= \eta^{0+1} = \eta^1$ as shown in Fig. 6(c). In the third round of the Msg_Collecting Phase of ECP_{AP}, each processor will send the messages stored at Level 2 of its ECP-tree to other processors and store the messages it has collected at Level 3 of its ECP-tree. An example with the ECP-tree of P_1 after 3 rounds of message exchange is shown in Fig. 6(d).

After 3 rounds of message exchange, each processor will enter the Dec_Making Phase. Finally, each processor will use the $vote_{AC}$ function to calculate the common values from the leaf nodes to the root. In this example, correct processor P_1 obtains τ as the final common value and is confident that this value matches with the values obtained by other correct processors. Below is a portion of the computation process.

$vote_{AC}(P_3P_4P_1) = vote_{AC}(\text{val}(P_3P_4P_1) = vote_{AC}(\tau) = \tau$
Condition 1 in $vote_{AC}$ function

$vote_{AC}(P_3P_4)$ = $vote_{AC}(\text{val}(P_3P_4P_1)$, val($P_3P_4P_2$), val($P_3P_4P_5$), val($P_3P_4P_6$), val($P_3P_4P_7$),val($P_3P_4P_8$),) = $vote_{AC}(\tau,\tau,\tau,\tau,\omega,\tau) = \tau$
Condition 4 in $vote_{AC}$ function

$vote_{AC}(P_3)$ = $vote_{AC}(\text{val}(P_3P_1)$, val($P_3P_4$), val($P_3P_5$), val($P_3P_6$), val($P_3P_7$), val($P_3P_8$)) = $vote_{AC}(\rho,\tau,\tau,\tau,\rho,\tau,\rho) = \tau$
Condition 4 in $vote_{AC}$ function

$vote_{AC}(P_5P_6)$ = $vote_{AC}(\text{val}(P_5P_6P_1)$, val($P_5P_6P_2$), val($P_5P_6P_3$), val($P_5P_6P_4$), val($P_5P_6P_7$),val($P_5P_6P_8$),) = $vote_{AC}(\eta^2,\eta^2,\eta^2,\eta^2,\eta^2,\eta^2) = \eta^1$
Condition 3 in $vote_{AC}$ function

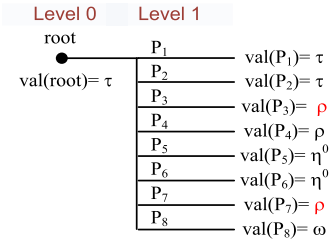
$vote_{AC}(\text{root})$ = $vote_{AC}(\text{val}(P_1)$, val($P_2$), val($P_3$), val($P_4$), val($P_5$), val($P_6$), val($P_7$) ,val($P_8$)) = $vote_{AC}(\tau,\tau,\tau,\rho,\eta^0,\eta^0,\rho,\omega) = \tau$
Condition 4 in $vote_{AC}$ function

V. CORRECTNESS AND COMPLEXITY

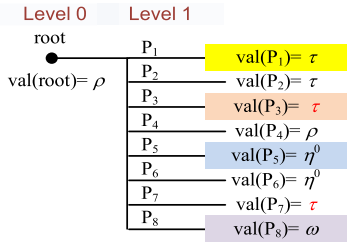
In this section, the correctness of the ECP_{AP} in solving the Consensus (n, m) with alternative plans problem is proved, and the complexity of the ECP_{AP} is evaluated. Moreover, whether the number of rounds of message exchange required is minimal is also examined.

A. CORRECTNESS

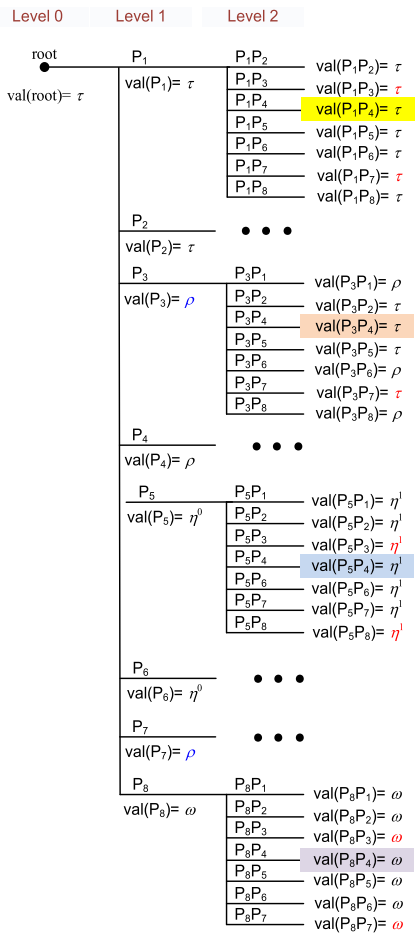
In proving the correctness of the proposed protocols, we define that a vertex α is called *common* [33] if the values stored at α of all the correct processors' ECP-trees



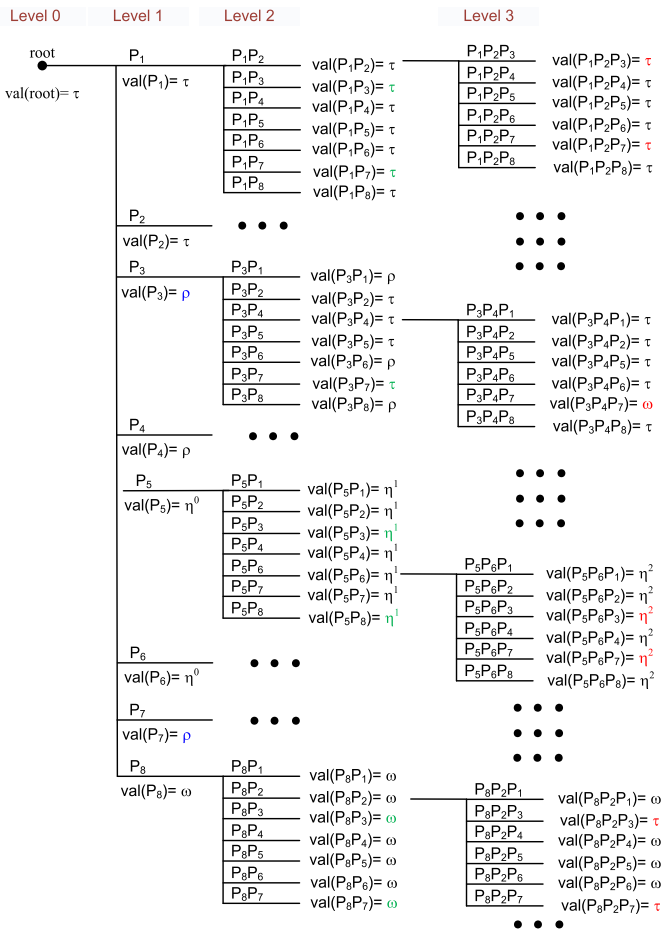
(a) The ECP-tree of P_1 after the 1st round of Msg_Collecting Phase of ECP_{AP}



(b) The ECP-tree of P_4 after the 1st round of Msg_Collecting Phase of ECP_{AP}



(c) The ECP-tree of P_1 after the 2nd round of Msg_Collecting Phase of ECP_{AP}



(d) The ECP-tree of P_1 after the 3rd round of Msg_Collecting Phase of ECP_{AP}

FIGURE 6. **An example of executing ECP_{AP}.**

are identical. That is, if each correct processor has a common value stored at its root, then a consensus is reached since the root is common. Besides, if every root-to-leaf path of an ECP-tree contains a common vertex, and then the collection of the common vertices forms a *common frontier* [33]. Subsequently, by using the same vote function (i.e. $vote_{AC}$) to compute the root value of the tree structure, every correct processor can obtain the same root value because the same input (i.e. common frontier) and the same computing function (i.e. $vote_{AC}$) generate the same output (i.e. a common good plan). The following lemmas and theorems are proposed to prove the correctness of the proposed ECP_{AP}.

Lemma 1: All the correct vertices of an ECP-tree are common if the number of Byzantine processors $f_b \leq t$, where $t = \lfloor (n-1)/3 \rfloor$.

Proof: To prove that a common value can be obtained by the ECP_{AP}, we define two terms, "*correct vertex*" and "*true value*", as follows:(1) *Correct vertex*: Vertex αi of a tree is considered a correct vertex if processor P_i (the last processor name in vertex αi, where α is a sequence of processor *ID* and i is a single processor id) is correct. In other words, a correct vertex stores the value from a correct processor. (2) *True value*: For a *correct vertex*αi in the tree of a correct processor P_j, val(αi) is the *true value* of vertex αi. In other

words, if processor P_j is correct, then the stored value of *correct vertex* αi is called the *true value*. There are no vertices with repeated names in an ECP-tree. At Level $t+1$ or above, the correct vertex α has at least $2t+1$ children, of which at least $t+1$ children are correct. The true values of these $t+1$ correct vertices are in common, and the majority value of vertex α is common. The correct vertex α is common in the ECP-tree if the level of α is lower than $t+1$. As a result, all the correct vertices of the ECP-tree are common.

Lemma 2: The common frontier exists in the ECP-tree.

Proof: There are $t+1$ vertices along each root-to-leaf path of an ECP-tree in which the root is labeled by *root*, and the others are labeled by a sequence of processor names. Since at most t Byzantine processors are likely to fail, there is at least one correct vertex along each root-to-leaf path of the ECP-tree. By Lemma 1, the correct vertex is common, and the common frontier exists in each correct processor's ECP-tree.

Lemma 3: Let α be a vertex; α is common if there is a common frontier in the sub-tree 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 ECP-tree.

Theorem 1: The root of a correct processor's ECP-tree is common.

Proof: By Lemma 1, Lemma 2, Lemma 3, and Corollary 1, the theorem holds.

Theorem 2: The proposed ECP_{AP} protocol can solve the Consensus(n,m) with alternative plans problem if the number of Byzantine processors f_b is smaller than t, where $f_b = \lfloor (n-1)/3 \rfloor$.

Proof: The value of the correct vertices for the ECP-tree of all the correct processors is v. As a result, each correct vertex of the ECP-tree is common (Lemma 2), and its true value is v. By Theorem 1, this root is common. The computed value $vote_{AP}$(ECP-tree)$= v$ is stored in the root for all the correct processors. Therefore, the validity is confirmed.

B. COMPLEXITY

The complexity of the ECP_{AP} protocol is evaluated by two criteria, including (1) the number of rounds of message exchange (i.e. time complexity), and (2) the number of messages required (i.e. space complexity). The following theorems are used to evaluate the complexity of the ECP_{AP} protocol.

Theorem 3: The proposed ECP_{AP} protocol solves the Consensus(n,m) with alternative plans problem by using a minimum number of rounds (t+1 rounds) of message exchange.

Proof: In message exchange, oral messages (non-encrypted message) are used; in estimating the frequency of message exchange, the operation is counted by round [16]. Fischer and Lynch [16] point out that when the failure type of processors in the network is the worst Byzantine fault, for a network consisting of n processors and relying on oral messages for message exchange, the network can tolerate a maximum of $\lfloor (n-1)/3 \rfloor$ Byzantine processors. In such network, the number of rounds of message exchange is $t+1$ $(t = \lfloor (n-1)/3 \rfloor)$. Moreover, Fischer and Lynch also prove that the minimum number of rounds of message exchange is $t+1$. In this paper, a similar setting as used by Fischer and Lynch is adopted. Because the number of rounds of message exchange of ECP_{AP} is $t+1$, and this number is the minimum as shown by Fischer and Lynch, the number of rounds required by the ECP_{AP} protocol is also the minimum.

Theorem 4: The number of messages required by the proposed ECP_{AP} is minimum ($(t+1)*n^2$).

Proof: According to Theorem 3, the number of rounds of message exchange is $t+1$. In each round, each processor sends its message (the value stored at the last level of its ECP-tree tree) to all the other processors in the network. The total number of processors in the network is n, so at most n^2 messages will be generated in each round. Therefore, the total number of messages required by ECP_{AP} protocol is $(t+1)*n^2$. By Theorem 3, $t+1$ rounds is the minimum number of rounds of message exchange. Hence, $(t+1)*n^2$ is also the minimum number of messages required. That is, the number of messages required by the ECP_{AP} protocol is the minimum.

Further, the performance of the ECP_{AP} protocol in terms of the number of allowable Byzantine processors, the number of rounds required, and the number of messages required are evaluated. The number of allowable Byzantine processors depends on the number of processors in the network. The number of allowable Byzantine processors with varied numbers of processors in the network (30∼80 processors) is shown in Fig. 7. As shown in Fig. 7, with the increase of the number of processors, the number of allowable Byzantine processors also increases. This is because the ECP_{AP} protocol takes advantage of the distributed feature of the system. It relies on adequate cooperation between processors to overcome the attacks from Byzantine processors. Therefore, when there is a greater number of processors in the system, the number of Byzantine processors that can be tolerated

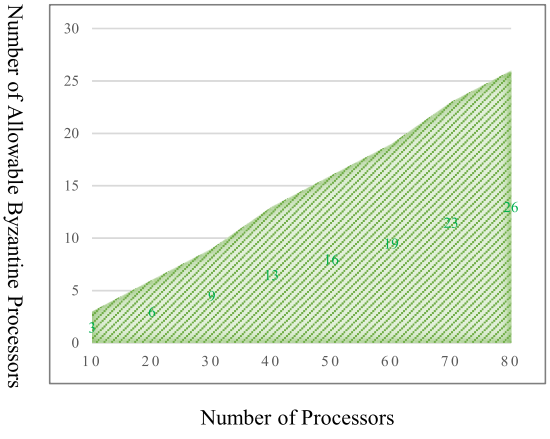


FIGURE 7. The number of allowable Byzantine Processors.

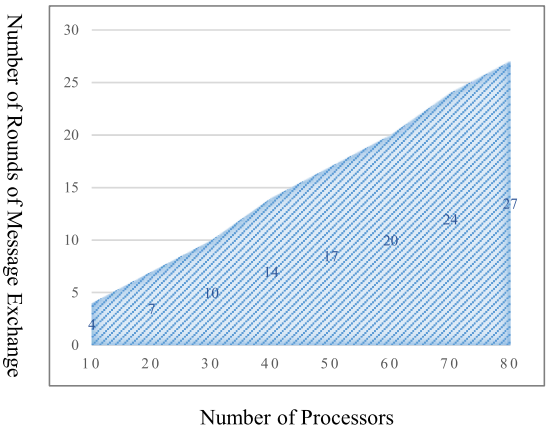


FIGURE 8. The number of rounds of message exchange.

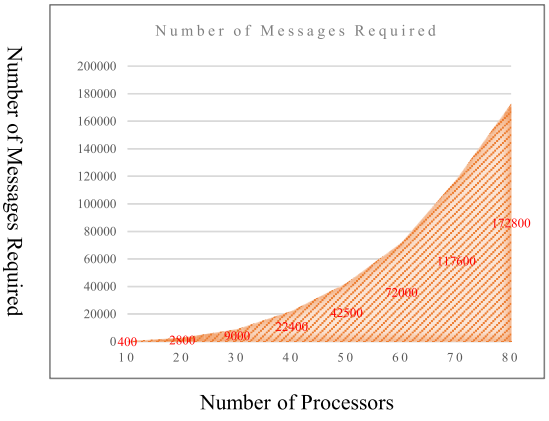


FIGURE 9. The number of messages required.

can be greater. For instance, given the number of processors $n = 10$, the number of allowable Byzantine processors by ECP_{AP} protocol is 3. Given the number of processors $n = 80$, the number of allowable Byzantine processors by ECP_{AP} protocol grows to 26.

The number of rounds of message exchange is shown in Fig. 8. As shown in this figure, with the increase of the number of processors, the number of rounds of message exchange also increases. This is because when the number of processes increases, the number of allowable Byzantine processors will also increase. To eliminate possible interference from Byzantine processors, there should be more rounds of message exchange. However, when the number of rounds of message exchange becomes greater, the number of messages required will grow relatively. Fig. 9 shows the number of messages required given varied numbers of processors in the network. The number of messages required is also equal to the storage cost. For instance, given the number of processors $n = 10$, the number of messages required is 400 units. Given the number of processors $n = 80$, the number of messages required grows to 172,800 units. As can be observed in Fig. 9, the number of messages required grows rapidly.

VI. CONCLUSION

In this paper, the traditional Consensus problem is extended to the Consensus (n, m) problem with alternative plans. The main contributions of this paper are summarized as follows: (1) The number of processors that are allowed to have an initial value is relaxed; (2) The proposed protocol makes all the correct processors in the network obtain the same set of good plans and the same set of bad plans. This is very important to the Consensus problem with alternative plans, because Correia et al. [22] have pointed out that when correct processors have possibly different good plans and possibly different bad plans (i.e. Variation 4 in Table 1), the Consensus problem with alternative plans cannot be solved; (3) ECP_{AP} does not take advantage of the reliable broadcast protocol to transmit messages. Thus, Byzantine senders can send inconsistent messages to different processors. Nevertheless, the proposed ECP_{AP} can still solve the Consensus (n, m) with alternative plans problem without using the reliable broadcast protocol.

ACKNOWLEDGMENT

This paper was presented in part at the Proceedings of the 2013 IEEE Wireless Communications and Networking Conference (WCNC'13) [24].

REFERENCES

[1] *Distributed Computing*. Accessed: Jul. 1, 2019. [Online]. Available: http://en.wikipedia.org/wiki/Distributed_computing
[2] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 452–465, Apr. 2010.
[3] W. Zhao, "Application-aware Byzantine fault tolerance," in *Proc. IEEE DASC*, vol. 14, Aug. 2014, pp. 45–50.
[4] J. Chen, W. Zhang, Y.-Y. Cao, and H. Chu, "Observer-based consensus control against actuator faults for linear parameter-varying multiagent systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 7, pp. 1336–1347, Jul. 2017.
[5] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient Byzantine fault tolerance," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2807–2819, Oct. 2016.
[6] S. Iqbal, M. L. M. Kiah, B. Dhaghighi, M. Hussain, S. Khan, M. K. Khan, and K. R. Choo, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," *J. Netw. Comput. Appl.*, vol. 74, pp. 98–120, Oct. 2016.
[7] C.-X. Shi, G.-H. Yang, and X.-J. Li, "Data-based fault-tolerant consensus control for uncertain multiagent systems via weighted edge dynamics," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published. doi: 10.1109/TSMC.2017.2743261.
[8] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Hoboken, NJ, USA: Wiley, 2013.
[9] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.
[10] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980.
[11] *Byzantine Fault Tolerance*. Accessed: Jul. 1, 2019. [Online]. Available: http://en.wikipedia.org/wiki/Byzantine_fault_tolerance
[12] V. Pathak and L. Iftode, "Byzantine fault tolerant public key authentication in peer-to-peer systems," *Comput. Netw.*, vol. 50, no. 4, pp. 579–596, 2006.
[13] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling Byzantine fault-tolerant replication to wide area networks," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 1, pp. 80–93, Jan./Mar. 2010.
[14] V. King, J. Saia, V. Sanwalani, and E. Vee, "Towards secure and scalable computation in peer-to-peer networks," in *Proc. IEEE FOCS*, vol. 6, Oct. 2006, pp. 87–98.
[15] V. King, J. Saia, V. Sanwalani, and E. Vee, "Scalable leader election," in *Proc. SODA*, 2006, pp. 990–999.

[16] M. J. Fischer and N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Inform. Process. Lett.*, vol. 14, no. 4, pp. 183–186, 1982.
[17] F. Bonnet, X. Défago, T. D. Nguyen, and M. Potop-Butucaru, "Tight bound on mobile Byzantine agreement," *Theor. Comput. Sci.*, vol. 609, pp. 361–373, Jan. 2016.
[18] C.-F. Cheng and K.-T. Tsai, "From immediate agreement to eventual agreement: Early stopping agreement protocol for dynamic networks with malicious faulty processors," *J. Supercomput.*, vol. 62, no. 2, pp. 874–894, 2012.
[19] C. Deplorte-Gallet, H. Fauconnier, and H. Tran-The, "Byzantine agreement with homonyms in synchronous systems," *Theor. Comput. Sci.*, vol. 496, pp. 34–49, Jul. 2015.
[20] O. Babaoglu and R. Drummond, "Streets of Byzantium: Network architectures for fast reliable broadcasts," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 6, pp. 546–554, Jun. 1985.
[21] H.-S. Siu, Y.-H. Chin, and W.-P. Yang, "Byzantine agreement in the presence of mixed faults on processors and links," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 4, pp. 335–345, Apr. 1998.
[22] M. Correia, A. N. Bessani, and P. Veríssimo, "On Byzantine generals with alternative plans," *J. Parallel Distrib. Comput.*, vol. 68, pp. 1291–1296, Sep. 2008.
[23] M. Correia, N. F. Neves, and P. Veríssimo, "From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures," *Comput. J.*, vol. 49, no. 1, pp. 82–96, Jan. 2006.
[24] C.-F. Cheng, "Obtaining consistent good/bad plan set in the presence of faults," in *Proc. WCNC*, vol. 13, Apr. 2013, pp. 286–290.
[25] C.-F. Cheng and K.-T. Tsai, "A recursive *Byzantine*-resilient protocol," *J. Netw. Comput. Appl.*, vol. 48, pp. 87–98, Feb. 2015.
[26] A. MostÃlfaoui, H. Moumen, and M. Raynal, "Randomized k-set agreement in crash-prone and Byzantine asynchronous systems," *Theor. Comput. Sci.*, vol. 709, pp. 80–97, Jan. 2018.
[27] C. Li, Y. Wang, and M. Hurfin, "Clock synchronization in mobile ad hoc networks based on an iterative approximate Byzantine consensus protocol," in *Proc. AINA*, vol. 14, May 2014, pp. 210–217.
[28] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Proc. CRYPTO*, vol. 1, 2001, pp. 524–541.
[29] S. Jafar, A. Krings, and T. Gautier, "Flexible rollback recovery in dynamic heterogeneous grid computing," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 1, pp. 32–44, Jan./Mar. 2009.
[30] H.-Y. Tzeng and K.-Y. Siu, "On the message and time complexity of protocols for reliable broadcasts/multicasts in networks with omission failures," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 7, pp. 1296–1308, Sep. 1995.
[31] A. Vempaty, L. Tong, and P. K. Varshney, "Distributed inference with Byzantine data: State-of-the-art review on data falsification attacks," *IEEE Signal Process. Mag.*, vol. 30, no. 5, pp. 65–75, Sep. 2013.
[32] A. W. Krings and T. Feyer, "The Byzantine agreement problem: Optimal early stopping," in *Proc. HICSS*, vol. 8, Jan. 1999, pp. 1–12.
[33] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, "Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement," *Inf. Comput.*, vol. 97, no. 2, pp. 205–233, 1992.

CHIEN-FU CHENG received the Ph.D. degree in computer science from National Chiao-Tung University, Hsinchu, Taiwan, in 2008.

He is currently a Full Professor with the Department of Computer Science and Information Engineering, Graduate Institute of Networking and Multimedia, Tamkang University, New Taipei, Taiwan. He has published many papers in several prestigious conferences and journals, such as the IEEE LCN, the IEEE MASS, the IEEE VTC, the IEEE WCNC, the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE INTERNET OF THINGS JOURNAL, the IEEE SENSORS JOURNAL, and the IEEE COMMUNICATIONS LETTERS. His current research interests include wireless communication and mobile computing, wireless ad hoc and sensor networks, distributed computing, and fault-tolerant computing.

Dr. Cheng has served as a TPC member for more than 50 communication and computer conferences, such as the IEEE GLOBECOM, the IEEE ICC, the IEEE INFOCOM, and the IEEE WCNC. He is currently an Associate Editor of IEEE ACCESS.

KUO-TANG TSAI received the B.S. degree in computer science and information engineering and the M.S. degree from the Graduate Institute of Networking and Communication, Tamkang University, New Taipei, Taiwan, in 2009 and 2011, respectively.

He is currently with AsusTek Computer Inc., Taipei, Taiwan. His research domains include distributed computing and fault-tolerant computing.