INFORMATION AND COMPUTATION 75, 130–143 (1987)

Asynchronous Byzantine Agreement Protocols

GABRIEL BRACHA

13Bart Street, Tel-Aviv 69104, Israel

A consensus protocol enables a system of n asynchronous processes, some of them faulty, to reach agreement. Both the processes and the message system are capable of cooperating to prevent the correct processes from reaching decision. A protocol is t-resilient if in the presence of up to t faulty processes it reaches agreement with probability 1. Byzantine processes are faulty processes that can deviate arbitrarily from the protocol; Fail–Stop processes can just stop participating in it. In a recent paper, t-resilient randomized consensus protocols were presented for $t < n/5$. We improve this to $t < n/3$, thus matching the known lower bound on the number of correct processes necessary for consensus. The protocol uses a general technique in which the behavior of the Byzantine processes is restricted by the use of a broadcast protocol that filters some of the messages. The apparent behavior of the Byzantine processes, filtered by the broadcast protocol, is similar to that of Fail–Stop processes. Plugging the broadcast protocol as a communicating primitive into an agreement protocol for Fail–Stop processes gives the result. This technique, of using broadcast protocols to reduce the power of the faulty processes and then using them as communication primitives in algorithms designed for weaker failure models, was used succesfully in other contexts.

1. INTRODUCTION

An important advantage of distributed systems over centralized systems is their ability to tolerate failures. A common method to achieve that ability is to have several processes cooperatively perform the same task. Fundamental to such cooperation is the ability of processes to agree on a common datum or action. The *consensus* and *Byzantine Generals* (Lamport *et al.*, 1982; Pease *et al.*, 1980) problems provide basic paradigms of achieving agreement in distributed systems in the presence of faulty processes.

In the *consensus problem*, each process p starts with a local binary value, $value_p$, and the processes have to decide on a common value. A *consensus protocol* that solves the consensus problem terminates when each correct process makes an irreversible *decision* on some value. The protocol must satisfy the following properties:

Agreement: all correct processes decide on the same value.

Validity: if all correct processes start with the same value v, then all correct processes decide on v.

0890-5401/87 $3.00

In the *Byzantine Generals problem*, one process, the *transmitter*, broadcasts its local value, v, to the other processes. The processes have to agree on the value of the transmitter. A *Byzantine Generals protocol* must satisfy the following properties:

Agreement: all correct processes decide on the same value.

Validity: if the transmitter is correct then all correct processes decide v.

We consider the following model of a distributed system. The system consists of n processes that communicate by sending *messages* through a *message system*. We assume a reliable message system in which no messages are lost or generated. Each process can directly send messages to any other process, and can identify the sender of every message it receives. Up to t of the processes are *faulty* and may deviate from the protocol. A protocol is called *t-resilient* if it satisfies the agreement and validity requirements in the presence of up to t faulty processes.

Solutions to agreement problems depend strongly on the assumptions made about the system and the type of solution desired. There are several orthogonal parameters that characterize agreement problems. These parameters define the synchrony of the system, the behavior of faulty processes, and the way the protocol terminates (deterministic termination or probabilistic termination).

Two basic types of systems are considered: synchronous systems and asynchronous systems. In a *synchronous* system, processes run in lock step, and messages sent in one step are received in the next. Protocols in such systems can be viewed as a sequence of *rounds*. In each round, every process sends messages to other processes, receives messages sent to it in that round, and then changes its state according to these messages.

The other extreme is an *asynchronous* system in which there are no bounds on message delays or relative speeds of processes. In such a system, it is impossible to detect missing messages; there is no way to distinguish between a "slow" message and a message not sent. Protocols for asynchronous systems can also be viewed as a sequence of rounds. In each round, every process sends messages to all others, waits for only $n-t$ messages of that round, and changes state. The processes cannot wait for more than $n-t$ messages in a round since there is a possibility that all t faulty processes do not send any message in that round.

In synchronous systems, the execution of the protocol is determined by the initial values of all processes and by the behavior of faulty processes. In asynchronous systems, we need to postulate an additional agent, the *scheduler*, that will determine in each round for each process which $n-t$ messages it receives out of n potential messages. We assume that the scheduler is an *adversary* scheduler and that it tries to prevent agreement by selectively routing messages to processes.

There is a whole spectrum of failure types, varying in the degree faulty processes can deviate from the protocol. The failure type affects the complexity of the protocol and the number of faulty processes it tolerates. In this paper we consider the following failure types:

Fail–Stop. Faulty processes may omit messages at any time, but after the first such omission, they "die" and stop participating in the protocol.

Byzantine. Faulty processes may deviate arbitrarily from the protocol; in particular they can fail to send messages when they should and can send spurious and contradictory messages.

There are two types of protocols: deterministic and randomized. In *deterministic* protocols no random steps are taken; the execution depends on the initial values, the behavior of faulty processes, and the scheduler. Protocols that use random steps are called *randomized* protocols.

For deterministic protocols we impose the following termination requirement:

Termination (deterministic). All correct processes decide by round r, for some a priori known constant r.

In Fischer *et al.* (1985) it was shown that, in asynchronous systems, deterministic consensus protocols are impossible even in the simple case of only one Fail–Stop process.

For randomized protocols we only require that agreement is eventually reached. In other words,

Termination (probabilistic). The probability that a correct process is undecided after r rounds approaches zero as r approaches infinity.

Thus, though the number of rounds to reach agreement is not bounded, the probability that the protocol does not terminate is zero.

In Ben-Or (1983) a t-resilient consensus protocol for asynchronous systems is presented. This protocol tolerates $t<n/2$ Fail–Stop processes, or $t<n/5$ Byzantine processes.

In Bracha and Toueg (1985) a weaker model is treated where there are some probabilistic assumptions on the behavior of the scheduler.

In Rabin (1983) a t-resilent consensus protocol for asynchronous systems with $t<n/10$ Byzantine processes is presented. The expected number of rounds to reach agreement is only four. In Toueg (1984) this protocol is extended to tolerate up to $t<n/3$ faulty processes. However, the model of Rabin (1983) is stronger than our model. It assumes an initial stage in which a sequence of coin tosses is reliably distributed to all the processes. But distributed on-line generation of such a sequence of coin tosses is very costly, and therefore we cannot compare this protocol to protocols that run on our model.

In Ben-Or (1983) it was left as an open problem whether there are consensus protocols that can tolerate up to $n/5 \leqslant t < n/3$ Byzantine processes. In this paper we answer that question by presenting a randomized consensus protocol that tolerates up to $t < n/3$ Byzantine processes, thus matching the lower bound on the number of correct processes required for any consensus protocol (Bracha and Toueg, 1985).

The main contribution of this paper is in the methodology that it illustrates. Instead of dealing directly with the worst behavior of the Byzantine processes, we develop a general technique that reduces their effect on the system so they can do little more than Fail–Stop processes. The technique has two parts, a reliable broadcast primitive and a validation method. The broadcast primitive forces the faulty processes either to send nothing or to send the same message to all correct processes. The validation method forces the faulty processes to send only messages that could have been sent by correct processes. This validation method is then plugged as a communication primitive in an agreement protocol for Fail–Stop processes. This methodology of designing protocols that reduce the power of the faulty processes and then using those protocols as communication primitives yields efficient algorithms that are easy to understand and prove. Also, it is quite general; it was carried further in (Srikanth and Toueg, 1984; Toueg *et al.*, 1985), where it yielded simpler and more efficient synchronous Byzantine Generals protocols and clock synchronization algorithms.

Plugging the assumptions of Rabin's model into the protocol we immediately obtain the result of Toueg (1984) and an improvement on Rabin (1983).

We also prove that Byzantine Generals protocols are impossible in asynchronous systems. A weaker version of the Byzantine Generals problem is proposed and solved.

2. Reliable Broadcast

We first present a broadcast protocol that will be used as a primitive in the consensus protocol. In a single instance of the broadcast protocol, some designated process, p, sends messages containing its value to all other processes. The protocol is a *reliable broadcast protocol* if it satisfies the following properties:

1. if p is correct, then all correct processes agree on the value of its message;

2. if p is faulty then, either all correct processes agree on the same value or none of them accepts any value from p.

2.1. *The Broadcast Primitive*

The broadcast primitive is described in Fig. 1. There are three types of messages used in the protocol: *initial*, *echo*, and *ready*. An (*initial*, v) message means that p wishes to broadcast the value v. An (*echo*, v) message means that its sender knows that p sent v because it received either an (*initial*, v) message from p, or enough (*echo*, v) or (*ready*, v) messages confirming it. A (*ready*, v) message means that its sender knows that v is the only value sent by p, and that it is ready to accept v because it received enough (*echo*, v) or (*ready*, v) messages. When a processes receives enough (*ready*, v) messages it *Accepts* v as the value sent by p, knowing that all other correct processes are bound to accept v too.

The protocol is divided into steps corresponding to the message types. In each step a process waits until it receives enough messages that permit it to send the next message type (including those received at previous steps), then it sends a message to all the processes and moves to the next step. Thus, a correct process sends one message of each type (one message each step) to any other process. When process p wants to broadcast a value v, it calls *Broadcast*(v). The broadcast is succesful if all the correct processes *Accept* v. Thus, *Broadcast* and *Accept* provide us a pair of communication primitives.

2.2. *Correctness Proof*

In this section we show that, for $0 \leqslant t < n/3$, the protocol of Fig. 1 is a reliable broadcast protocol.

LEMMA 1. *If two correct processes s and t send* (*ready*, v) *and* (*ready*, u) *messages, respectively, then* $u = v$.

Protocol 1

```
Broadcast($v$)
step 0. (By process $p$)
        Send (initial, $v$) to all the processes.

step 1. Wait until the receipt of,
            one (initial, $v$) message
            or $(n+t)/2$ (echo, $v$) messages
            or $(t+1)$ (ready, $v$) messages
            for some $v$.
        Send (echo, $v$) to all the processes.

step 2. Wait until the receipt of,
            $(n+t)/2$ (echo, $v$) messages
            or $t+1$ (ready, $v$) messages
            (including messages received in step 1)
            for some $v$.
        Send (ready, $v$) to all the processes.

step 3. Wait until the receipt of,
            $2t+1$ (ready, $v$) messages
            (including messages received in step 1 and step 2) for some $v$.
        Accept $v$.
```

FIG. 1. The broadcast primitive.

Proof. Suppose not; let q be the first process that sends a (*ready*, v) message, and let r be the first process that sends a (*ready*, u) message. Process q must have received more than $(n+t)/2$ (*echo*, v) messages, and process r must have received more than $(n+t)/2$ (*echo*, v) messages. Therefore, some correct process must have sent both (*echo*, u) and (*echo*, v) messages. But correct processes send only one message of each type during a broadcast, and hence a contradiction. ∎

LEMMA 2. *If two correct processes, q and r, accept the values v and u, respectively, then $u = v$.*

Proof. If q accepts the value v then it must have received $(2t+1)$ (*ready*, v) messages, and therefore at least $(t+1)$ (*ready*, v) messages from correct processes. Similarly, r must have received at least $(t+1)$ (*ready*, u) messages from correct processes. By Lemma 1, $u = v$. ∎

LEMMA 3. *If a correct process q accepts the value v then every other correct process will eventually accept v.*

Proof. If q accepts the value v then q received $(2t+1)$ (*ready*, v) messages. At least $t+1$ of these messages were sent by correct processes. Therefore, every other process receives at least $(t+1)$ (*ready*, v) messages, and sends its own (*ready*, v) message. Note that by Lemma 1 it is impossible for a correct process to send a different *ready* message. Thus, at least $n-t$ processes send (*ready*, v) messages. Every correct process r eventually receives at least $(2t+1 \leqslant n-t)$ (*ready*, v) messages and accepts v. ∎

LEMMA 4. *If a correct process p broadcasts v then all correct processes accept v.*

Proof. Suppose that p is correct and broadcasts v. Every correct process q receives an (*initial*, v) message and responds by sending (*echo*, v) messages. Every correct process q will receive $(n-t > (n+t)/2)$ (*echo*, v) messages from the correct processes, and possibly $t < (n+t)/2$ different messages from the faulty processes. Therefore, q will send a (*ready*, v) message. In step 3, every correct process q will receive $(n-t \geqslant 2t+1)$ (*ready*, v) messages, and possibly t different *ready* messages from the faulty processes. Therefore q will accept v. ∎

THEOREM 1. *The protocol of Fig.* 1 *is a reliable broadcast protocol.*

Proof. Let process p broadcast a message with the value v:

1. If p is correct then, by Lemma 4, all correct processes accept v.

2. If p is faulty and some correct process q accepts a value v, then, by Lemma 3, all correct processes accept v. Otherwise, no correct process accept any value. ∎

3. Correctness Enforcement

In the previous section, we restricted the behavior of the Byzantine processes by forcing them to send the same message to all processes or no message at all. However, we could not control the content of the message. In this section we present a scheme that forces the Byzantine processes to conform to the underlying protocol.

Consider the general outline of an asynchronous protocol in Fig. 2. N is the *protocol function* that determines the new value of the variable v according to the round number and S. Note that the set of $n-t$ messages from round k could have been received by process p while it was not yet in round k, these messages are stored by p till round k and only then they are used to generate the value v. We will show that for any protocol that can be put in this form the faulty processes can be forced to send only messages according to the protocol.

As a first step we use *Broadcast* and *Accept* instead of *Send* and *Receive*. This will cause several instances of the broadcast protocol to be active at the same time: broadcasts of different processes and, because of the asynchrony of the system, even broadcasts of the same process but from different rounds. To distinguish between messages sent in different broadcasts, all messages sent in the kth round by process p will be tagged with (p, k). A message of the form (p, k, v) is said to be a k-message with value v. A call to *Broadcast* (p, k, v) initiates the broadcast protocol of Fig. 1 with all messages tagged with (p, k). For each round k, each process p maintains a set of k-messages, $VALID_p^k$, defined as follows: $VALID_p^1 = \{(q, 1, v) \mid (q, 1, v)$ is accepted, and $v \in \{0, 1\}\}$.

For $k > 1$, $(q, k, v) \in VALID_p^k$, if (q, k, v) is accepted, and there exist $n-t$ messages $m_1, \ldots, m_{n-t}, \in VALID_p^{k-1}$ such that $v = N(k-1, \{m_1, \ldots, m_{n-t}\})$.

The processes update their *VALID* sets whenever they accept a message. A process p *validates* a k-message m if $m \in VALID_p^k$. Messages that are not validated are ignored in the protocol (although they are still stored for future validation). Intuitively, a k-message m is validated only if it could have been sent by a correct process in that round.

The basic round form is modified again to use *Broadcast* and *Validate* instead of *Send* and *Receive*, so that it has the following form shown in Fig. 3.

We now show that *Validate* has the same properties as *Accept*:

```
round($k$) by process $p$
Send ($p,k,v$) to all the processes
Wait until a set $S$ of $n-t$ messages from round $k$ have been received
$v := N(k,S)$
```

Fig. 2. A round of a general asynchronous protocol.

```
round($k$) by process $p$
  Broadcast($p$,$k$,$v$)
  Wait till a set $S$ of $n-t$ $k$-messages have been validated
  $v := N(k,S)$
```

FIG. 3. A modified round of a general asynchronous protocol.

LEMMA 5. *If two correct processes, p and q, validate* (r, k, v) *and* (r, k, u) *messages, respectively, then* $u = v$.

Proof. If $p(q)$ validates (r, k, v) (resp. (r, k, u)) then it must accept it. By Theorem 1, $u = v$. ▌

LEMMA 6. *If a correct process p validates a k-message m, then every other correct process q validates m, i.e., if p and q are correct then* $VALID_p^k = VALID_q^k$.

Proof. The proof is by induction on k. If $k = 1$, then by Theorem 1, if m is accepted by p then m will be accepted by q, and we are done. Let us assume that the statement of the lemma holds for some $k \geqslant 1$. Let $m = (r, k+1, v) \in VALID_p^{k+1}$. Therefore, there are $n-t$ messages $m_1, ..., m_{n-t} \in VALID_p^k$ such that $v = N(k, \{m_1, ..., m_{n-t}\})$. By our induction hypothesis, $m_1, ..., m_{n-t} \in VALID_q^k$. By Theorem 1, q will eventually accept m. Therefore, by definition, $m \in VALID_q^{k+1}$. ▌

LEMMA 7. *If a correct process p broadcasts a k-message m, then every correct process q eventually validates m.*

Proof. The proof is by induction on k. If $k = 1$, then, by Theorem 1, we are done. Suppose that the statement of the lemma holds for round k. Since p is correct, it can send $m = (p, k+1, v)$ only if there exist $n-t$ messages, $m_1, ..., m_{n-t} \in VALID_p^k$, such that $v = N(k, \{m_1, ..., m_{n-t}\})$. By Lemma 6, for every correct process q and for each m_i, $1 \leqslant i \leqslant n-t$, eventually $m_i \in VALID_q^k$. Also, since p is correct, by Theorem 1, every other correct process q will accept m. Therefore, eventually $m \in VALID_q^{k+1}$ for every correct process. ▌

So far we have shown that the *Broadcast* and *Validate* primitives provide us a reliable broadcast protocol. The additional power of Validate will be explicitly exploited in the consensus protocol in the next section.

4. THE CONSENSUS PROTOCOL

In this section we show how to construct an $n/3$-resilient consensus protocol using the primitives described in the previous sections. The protocol is basically the consensus protocol of Ben-Or (1983) and Bracha and Toueg (1985), into which we plugged our stronger communication primitives.

The protocol, described in Fig. 4, is conducted in phases that are executed by all processes; a process can proceed to phase $i+1$ only after it completed phase i. Phase i consists of rounds $3i+1$, $3i+2$, and $3i+3$, that are instances of the generic round of Fig. 3. The messages contain either a simple value, v, or a tagged value, (d, v), indicating that the process is ready to decide v at that phase. For notational convenience, the protocol in Fig. 4 does not terminate once a decision is made. However, this can be easily accomplished.

5. Correctness Proof

In this section we prove that the protocol in Fig. 4 is a t-resilient consensus protocol, for $t < n/3$. Since the protocol requires processes to wait for each other, we must show that it does not deadlock.

Lemma 8. *If a correct process p is at round i, then p will eventually progress to round $i+1$.*

Proof. Suppose not; then some correct processes are forever blocked. Let r be the first round in which some correct process p is forever blocked. By choice of r, all correct processes have already broadcast messages at that round. By Lemma 7, all these messages are eventually validated. Therefore, p is not blocked at round r, a contradiction. ▌

Lemma 9. *If at the beginning of round $3r+1$ all correct processes have the same value v, then they all decide v at round $3r+3$.*

Proof. All $n-t$ correct processes have v as their value at the beginning of round $3r+1$. Therefore, every correct process will validate at least $n-2t$ messages with value v at round $3r+1$. Since $n-2t > (n-t)/2$ for $t < n/3$, each correct process retains v as its value at round $3r+1$. By Lemma 8 all correct processes will proceed to rounds $3r+2$ and $3r+3$. At round $3r+2$,

Protocol 2

```
Phase($i$): ( by process $p$ )
1.  Broadcast($p$,$3i+1$,$value_p$). Wait until validate $n-t$ $3i+1$-messages.
    $value_p$ := majority value of the $n-t$ validated messages.
2.  Broadcast($p$. $3i+2$,$value_p$). Wait until validate $n-t$ $3i+2$-messages.
    (i) If more than $n/2$ of the messages have the same value $v$, then $value_p = (d,v)$.
    (ii) Otherwise, $value_p := value_p$.
3.  Broadcast($p$,$3i+3$,$value_p$). Wait until validate $n-t$ $3i+3$-messages.
    (i) If validated more than $2t$ messages with value $(d,v)$ then $decision_p := value_p := v$
    (ii) If validated more than $t$ messages with value $(d,v)$ then $value_p := v$.
    (iii) Otherwise, $value_p := coin\_toss$ (0 or 1 with probability ½).
    Go to round 1 of phase $i+1$
```

Fig. 4. The consensus protocol.

in order to validate a message with value $u \neq v$, a correct process must also validate at least $(n-t)/2$ messages with value u from round $3r+1$. Since $(n-t)/2 > t$, this is clearly impossible. Therefore, the only possible value validated in round $3r+2$ is v, and all correct processes change their value to (d, v). At round $3r+3$, all correct processes validate at least $2t+1$ messages with value (d, v), and they decide v. ▌

LEMMA 10. *Let p and q be correct processes. If at phase r, p validates a message with value (d, v), and q validates a message with value (d, u) message, then $u = v$.*

Proof. Suppose not; then some correct processes p and q have validated messages in round $3r+3$ with values (d, v) and (d, u), respectively. If p validates a message with value (d, v) it must also validate more than $n/2$ messages with value v that were broadcast at round $3r+2$. Similarly, q must have validated more than $n/2$ messages with value u that were broadcast at round $3r+2$. Thus, for some process s, p validated an $(s, 3r+2, v)$ message while q validated an $(s, 3r+2, u)$ message. By Lemma 5, $u = v$. ▌

THEOREM 2. *The protocol described in Figure 4 is a t-resilient consensus protocol, for $t < n/3$.*

Proof. We show that the protocol satisfies the validity, agreement, and termination properties:

Validity. If all the processes start with value v then, by Lemma 9, they all decide v.

Agreement. Suppose, without loss of generality, that a correct process p decides 1 at round $3r+3$. Process p must have validated at least $2t+1$ messages with value $(d, 1)$. By Lemma 6, every other correct process validates at least $t+1$ such messages. Thus at step (ii) of round $3r+3$, every correct process q sets $v_q = 1$. At the beginning of phase $r+1$ every correct process has value 1. By Lemma 9, at the end of phase $r+1$, every correct process decides 1.

Termination. Consider phase r of the protocol. Some processes are forced to set their value at round $3r+3$ at step (ii), while the remainder choose their values at step (iii) by a coin toss. Consider p, the first correct process that has completed the validation in round $3r+3$. There are two cases:

1. Process p has validated a message with value (d, v). With probability $\rho \geqslant 2^{-(n-t)}$, all the correct processes that toss a coin at phase r will toss v. By Lemma 10, the rest of the correct processes are forced (case (ii) of round $3r+3$ in Fig. 4) to set their value to v.

2. Process p has not validated a message with value (d, v). If some other correct process q validates more than t messages with value (d, v), then one of these messages would have been among the $n-t$ messages that p validated, and hence a contradiction. Therefore, no correct process validates more than t messages with value (d, v) for any value v, no process has its value forced, and all correct processes toss coins. Again, with probability $\rho \geqslant 2^{-(n-t)}$, all correct processes will set their values to 1.

In either case, with probability greater then $2^{-(n-t)}$, all the processes have the same value at the beginning of phase $r+1$, and by Lemma 9 they all decide in phase $r+1$. The probability of not terminating is $\lim_{r \to \infty} (1-\rho)^r = 0$. ▌

In Bracha and Toueg (1985) it was proved that for $t \geqslant n/3$, t-resilient randomized asynchronous consensus protocols are impossible. Thus, the protocol is optimal in the number of faulty processes it can tolerate.

6. Performance

THEOREM 3. (i) *If $t = c \cdot n$ for some constant c, then the expected number of phases to reach agreement is exponential in n.*

(ii) *If $t = c\sqrt{n}$ for some constant c, then the expected number of phases to reach agreement is a constant that does not depend on n (although it is exponential in c).*

Proof. The proof can be found in (Ben-Or, 1983). ▌

Theorem 3 provides us with a measure of the expected number of asynchronous steps taken by each process. We can also run the protocol in a synchronous system, but if we want to deduce the synchronous running time of the protocol from Theorem 3 we have to be more careful. Consider an instance of the broadcast protocol that is run in an synchronous system: If it is initiated by a correct process then it takes exactly three time-steps, but if it is initiated by a faulty process than it can take an indefinite amount of time, or it does not terminate at all. However, a round of the consensus protocol requires the termination of $n-t$ broadcasts which is guaranteed within three time-steps by the correct processes. Thus the correct processes set a pace of three time-steps per round, and the expected synchronous running time is constant (9) factor of the expected number of phases as specified by Theorem 3.

7. Rabin's Model

Rabin proposed in (Rabin, 1983) an $n/10$-resilient consensus protocol whose expected number of rounds is 4. In (Toueg, 1984) this protocol was extended to tolerate up to $t<n/3$ Byzantine processes. These protocols assume a stronger model of a distributed system than our own and therefore are not comparable to our protocol. This model assumes an initial phase in which a *trusted dealer* computes a sequences of coin tosses and distributes it to all the processes. Each coin toss value is shared among the processes by the "secret sharing" method, which guarantees that a collaboration of at least $t+1$ processes is needed to find the value of the coin. Thus, the rth coin toss will be available to processes only in the rth phase. A coin toss such that, after the toss, all the processes are guaranteed to have the same value of the coin is known as a *global* coin toss.

We can easily modify Protocol 2 to accommodate Rabin's model and immediately obtain the result of Toueg (1984). Instead of having each process toss its own coin in the third round of each phase, the processes access the global coin toss by exchanging portions of it. The resulting protocol is a consensus protocol that tolerates up to $t<n/3$ Byzantine processes and whose expected number of phases to termination is two.

8. Asynchronous Byzantine General Protocol

In this section we investigate the Byzantine Generals problem in asynchronous systems with Byzantine processes. In synchronous systems the Byzantine Generals problem is easily reduced to the consensus problem. First, the transmitter sends its value to all other processes. Then, the processes run a consensus protocol on the values they received from the transmitter. In asynchronous systems, such reduction is impossible. The processes do not know if the transmitter sent them any message at all, and therefore they cannot decide whether to continue waiting for a message from the transmitter or to regard the value of the transmitter as some default value. Thus, the termination properties for Byzantine Generals protocols are not achievable in an asynchronous system.

THEOREM 4. *Byzantine Generals protocols are impossible in asynchronous systems even in the presence of a single fault.*

Proof. Suppose that Byzantine Generals protocols are possible in an asynchronous system. Consider the following scenarios:

1. The transmitter is faulty, and all other processes are correct. The transmitter does not send any messages during the protocol. By the ter-

mination property of Byzantine Generals protocols, the processes agree on some value, let us say 0, at some time T_0.

2. The transmitter and the rest of the processes are correct. The transmitter sends 1-messages according to the protocol, however, none of these messages arrives befor time T_0. Until time T_0, the correct processes have the same view of the system as in the first scenario. Therefore they can go through the same execution of the protocol and decide on 0 at T_0, thus violating the validity requirement. ∎

We can still obtain a reasonable notion of reliable broadcast if, in the case of a faulty transmitter, we allow the broadcast not to terminate. The termination requirement is weakened as follows.

Weak termination: if the transmitter is correct then all correct processes eventually decide; if the transmitter is faulty, either all correct processes eventually decide, or none of them ever decides.

THEOREM 5. *In asynchronous systems, Byzantine Generals protocols with weak termination are possible if and only if $t < n/3$.*

Proof. If $t < n/3$, then the broadcast primitive of Fig. 1 is a weak termination Byzantine Generals protocol.

Suppose that there is a t-resilient Byzantine Generals protocol for $t \geqslant n/3$. We can partition the processes into three disjoint sets: A, B, and C, each of size t or less. Let the transmitter be in A and consider the following scenarios:

1. The processes in A and B are correct, and the transmitter sends 0-messages. The processes in C are faulty, and they do not send any messages during the protocol. Since the transmitter is correct and there are at most t faulty processes, the processes in A and B must agree on 0 within some time T_0.

2. Only the transmitter is faulty. It sends 0-messages to processes in A and B, and 1-messages to processes in C. The messages from C are delayed and not received until after T_0. The processes in A and B have the same view of the system as in scenario 1 and therefore must agree on 0 at time T_0.

In a similar fashion we can construct a third scenario with the following properties:

3. Only the transmitter is faulty. It sends 1-messages to A and C, and 0-messages to B. Messages from B are delayed and not received until after T_1. At time T_1 the processes in A and C agree on 1.

Combining scenarios 2 and 3 yields scenario 4 and a contradiction.

4. The processes in A are faulty, the processes in B and C are correct. The processes in A send messages to processes in B as in scenario 2, and to processes in C as in scenario 3. All messages between processes in B and processes in C are delayed and not received until after $\max(T_0, T_1)$. In this scenario, at time $\max(T_0, T_1)$, the processes in B will agree on 0 and the processes in C will agree on 1, a contradiction. ▌

RECEIVED September 19, 1984; ACCEPTED April 1987

REFERENCES

BEN-OR M. (1983), Another advantage of free choice: Completely asynchronous agreement protocols, *in* "Proceedings 2nd ACM Symposium on Principles of Distributed Computing, Montreal, Canada, August 1983," pp. 27-30.

BRACHA, G., AND TOUEG, S. (1985), Resilient consensus protocols, *J. Assoc. Comput. Mach.* **32**, No. 2, 824–840.

BRACHA, G. (1984), An $n/3$ resilient consensus protocol, *in* "Proceedings, 3rd Symposium on Principles of Distributed Computing, pp. 157–164.

FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. (1985), Impossibility of distributed consensus with one faulty process, *J. Assoc. Comput. Mach.* **32**, No. 2, 374–382.

LAMPORT, L., SHOSTAK, R., AND PEASE, M. (1982), The Byzantine Generals problem, ACM Transactions on Programming Languages and Systems, vol. 4, no. 3, July 1982, pp. 382–401.

PEASE, M., SHOSTAK, R., AND LAMPORT, L. (1980), Reaching agreement in the presence of faults, *J. Assoc. Comput. Mach.* **27**, No. 2, 228–234.

RABIN, M. (1983), Randomized Byzantine Generals, *in* "Proceedings, 24th Symposium on Foundations of Computer Science, Tuscon, Arizona, Nov. 1983," pp. 403–409.

SRIKANTH, T. K., AND TOUEG, S. (1984), "Byzantine Agreement Made Simple: Simulating Authenthication without Signatures," Tech. Rep. 84–623, Department of Computer Science, Cornell University, Ithaca, New York, July.

TOUEG, S. (1984), Randomized asynchronous Byzantine agreement, *in* "Proceedings, 3rd Symposium on Principles of Distributed Computing, Vancouver, Canada, August 1984," pp. 163–178.

TOUEG, S., PERRY, K. J., AND SRIKANTH, T. K. (1985), A simple and efficient Byzantine Generals algorithm with early stopping, *in* "Proceedings, 4th Symposium on Principles of Distributed Computing, Canada, August 1985.