Sādhanā, Vol. 11, Parts 1 & 2, October 1987, pp. 81–91. © Printed in India.

Byzantine-resilient distributed computing systems

L M PATNAIK[1] and S BALAJI[2]

[1]Department of Computer Science & Automation, Indian Institute of Science, Bangalore 560 012, India
[2]Control Systems Division, ISRO Satellite Centre, Vimanapura P.O., Bangalore 560 017, India

Abstract. This paper is aimed at reviewing the notion of Byzantine-resilient distributed computing systems, the relevant protocols and their possible applications as reported in the literature. The three agreement problems, namely, the consensus problem, the interactive consistency problem, and the generals problem have been discussed. Various agreement protocols for the Byzantine generals problem have been summarized in terms of their performance and level of fault-tolerance. The three classes of Byzantine agreement protocols discussed are the deterministic, randomized, and approximate agreement protocols. Finally, application of the Byzantine agreement protocols to clock synchronization is highlighted.

Keywords. Byzantine generals problem; agreement protocols; distributed computing; fault-tolerance.

1. Introduction

Very high reliability and uninterrupted operation of the computing system are vital in certain applications like on-board spacecraft systems and nuclear power plants. Malfunction of such critical systems causes severe penalties. This calls for the design of highly reliable and available computing systems. There are two approaches for achieving higher reliability, namely fault-avoidance and fault-tolerance. Fault-avoidance results from conservative design principles, such as, the use of high-reliability components, component burn-in, and careful signal-path routing with the goal of reducing the possibility of a failure. The fault-tolerance approach, on the other hand, accepts the inevitability of failures and overcomes the effects of defects through functional redundancy, thereby achieving a higher reliability than that is achievable by fault-avoidance techniques.

Fault-tolerance concepts have been extensively used during the past 15–20 years. The SIFT (software implemented fault-tolerance) computer is an example of a computer that is built using fault-tolerance concepts (Wensley *et al* 1978). The four

classes of faults studied in the literature are 'fail-stop', 'omission faults', 'timing faults' and 'Byzantine faults' (Cristian *et al* 1984; Ezhilchelvan & Shrivastava 1986). Because of the complexity of 'Byzantine faults', the Byzantine fault-tolerance concepts have gained the significant attention of researchers in the recent past. Developing techniques to tolerate 'Byzantine faults' in a distributed computing system (DCS) is gaining significance due to the inherent advantages of reliability, load balancing, high throughput, and modular expansion of such systems. This paper is an attempt at reviewing the work done in the area of 'Byzantine-resilient distributed computing systems'. The rest of the paper is organized as follows. Section 2 presents some of the conceptual preliminaries required to study the Byzantine-resilient DCS. In §3, we discuss the three classes of agreement problems. Some of the deterministic, randomized and approximate Byzantine agreement protocols reported in the literature with their merits and demerits are highlighted in §4. A few typical examples found in the literature concerning the use of Byzantine agreement protocols are highlighted in §5. Section 6 concludes the paper.

2. Conceptual preliminaries

In this section, we present some of the conceptual preliminaries required to study Byzantine-resilient DCS. We classify the faults into four types (Cristian *et al* 1984; Strong 1985; Ezhilchelvan & Shrivastava 1986). The simplest kind of fault is the 'fail-stop' type. In this type, a component (a processor or a link) may fail at any time, but once it fails, it immediately ceases to operate. A similar but more encompassing fault is the 'omission fault'. In this case, the component may fail to provide some specified function but otherwise it continues to operate normally. Both these classes of faults cannot alter and/or introduce spurious messages. Another class of faults is the 'timing faults'. In this case, a correct message or response comes from the faulty component earlier or later than the time specified for the arrival of the message. It may be noted that the 'omission faults' can be viewed as a special case of 'late timing faults' with an infinite delay. The 'early timing faults' and 'late timing faults' are also referred respectively as 'race faults' and 'performance faults' (Cristian *et al* 1984). Finally, the class of the most complex faults is the 'Byzantine faults' or the 'arbitrary faults'. A 'Byzantine fault' is an instance of arbitrary behaviour on the part of a device, a processor, or a program. It can exhibit malicious behaviour. It can send messages when it is not supposed to, make conflicting claims with other processes, act dead for a while and then revive itself.

A 'process' is an isolated agent of a DCS, having only a partial view of the global state of the DCS. The 'processes' involve themselves in cooperatively performing a joint task under the influence of a distributed algorithm. A fault-tolerant DCS concerns itself with the problems related to performing a cooperative task with potentially non-cooperative processes (a fault-tolerant system continues to perform its intended task despite failures in the underlying hardware/software). A DCS that can tolerate 'Byzantine faults' is termed as 'Byzantine-resilient DCS'. The degree of fault-tolerance of a DCS is equal to the maximum number of faults that can be tolerated without affecting the overall system performance. The two classes of processes considered in the design of a fault-tolerant DCS are 'synchronous' and

'asynchronous' (Perry 1985). Systems in which there is a finite bounded delay on the operations of the processes and on their intercommunication time are said to be 'synchronous'. In such systems, 'unannounced process deaths', as well as long delays in the response from the processes, are considered to be faults. The processes execute the assigned tasks in lock step in 'synchronous' systems. In an 'asynchronous' system, finite differences in the process speeds and message delivery time are allowed. In such a system, a very slow process cannot be distinguished from a 'dead' process.

The interval of time during which each non-faulty process is able to exchange a message with all other non-faulty processes is called a 'phase'. An 'oral message' is one whose contents are completely under the control of the sender (Pease *et al* 1980; Lamport *et al* 1982), so a traitorous sender transmits any possible message. An 'authenticated message' contains a portion of the message encoded in such a way that any receiver can verify that the message is authentic and the receiver can identify the sender, but no process can forge the signature of another (Pease *et al* 1980; Lamport *et al* 1982; Dolev & Strong 1983). Thus no process can change the contents of a message. The definition of 'oral message' is embodied in the following:

(a) every message that is sent is delivered correctly;
(b) the receiver of a message knows who has sent it;
(c) the absence of a message can be detected.

3. The agreement problems

One of the most important issues in a fault-tolerant DCS is that of reaching an agreement. In most applications, it is necessary for all processes to agree on the value broadcast by some process. Reaching an agreement is complicated by the presence of potentially faulty processes among the 'participants'. The key point is not what the processes agree on but the fact that they must all come to the same conclusion. Though voting seems to be an obvious solution, since distinct reliable processes might receive conflicting votes from a faulty process, the processes might also reach conflicting conclusions about the outcome of the election and hence fail to reach an agreement. Moreover, voting demands substantial hardware thereby having a detrimental effect on the cost and reliability. Reaching agreement has been extensively studied in the literature (Pease *et al* 1980; Lamport *et al* 1982; Fischer 1983; Attiya *et al* 1984; Toueg 1984; Turpin & Coan 1984; Lamport 1984; Mahaney & Schneider 1985; Perry 1985). In particular, reaching agreement in the presence of 'Byzantine faults' has gained significant attention of the researchers.

Three closely related agreement problems that have been extensively studied in the literature (Fischer 1983) are (i) the consensus problem, (ii) the interactive consistency problem, and (iii) the generals problem. The consensus problem is for the non-faulty processes to agree on a bit y, called the consensus value. A protocol for the consensus problem has to ensure that each reliable process i eventually terminates with a bit y_i, and $y_i = y$ for all i. The bit y, in general, will depend on the initial bits x_i for all $1 \leq i \leq n$, where n is the number of processes participating. The protocol to solve this 'consensus problem' must satisfy the following conditions:

Agreement: All non-faulty processes agree on a common value.

Validity: If all non-faulty processes choose the same initial value, then all non-faulty processes agree on this value.

The next in the class of agreement problems is the 'interactive consistency' problem. The 'interactive consistency' problem is similar to the 'consensus problem' except that the goal of the protocol is for the non-faulty processes to agree on a vector Y, called the 'consensus vector'. The last in the class of agreement problems is the 'generals problem' (Lamport *et al* 1982). The 'generals problem' is defined as follows: Given is a collection of n distributed, potentially faulty processes able to communicate only by means of messages. Assume that a distinguished process called the 'general' or 'transmitter' is trying to send its initial bit x to all other processes. The protocol for the 'generals problem' has to ensure that all the processes in the collection agree on the value x. The desired protocol is said to solve the 'generals problem' if it satisfies the following constraints:

Agreement: All non-faulty processes agree on a common value.

Validity: If the general does not fail, then all non-faulty processes agree on x.

The 'generals problems' is referred to as 'Byzantine generals problem' in the literature. The name 'Byzantine' refers to a military scenario that was initially used to describe the problem (Lamport *et al* 1982). The version of the Byzantine generals problem is 'synchronous' or 'asynchronous' depending on the underlying collection of processes operating synchronously or asynchronously.

Given a protocol for the 'consensus problem', the Byzantine generals problem may be solved by having each process choose the value broadcast by the general as its initial value. On the other hand, given a protocol for the Byzantine generals problem, the consensus problem may be solved allowing each process to execute a copy of the 'Byzantine agreement protocol'. In view of this, in this paper, we will concentrate only on the 'Byzantine agreement protocol'.

A 'Byzantine agreement' is said to be 'immediate Byzantine agreement' (IBA), if all non-faulty processes also agree during the phase at which they reach agreement. IBA is essential in cases where the processes are required to perform some synchronous action immediately after reaching agreement. Otherwise, we say that the agreement is 'eventual Byzantine agreement' (EBA) in the sense that each process decides on its value y but cannot synchronize its decision with that of the others until some later phase. One example where it is enough to ensure EBA is while guaranteeing the consistency of a distributed database. In a distributed database, one must operate on the most recently updated version of the database. All that is necessary to be ensured is that the version chosen by all other parties to the agreement is one and the same. It may be noted that IBA implies EBA. It is possible that EBA can often be reached earlier than IBA (Dolev *et al* 1982).

4. The Byzantine agreement protocols

The motivation behind the development of Byzantine agreement protocols (BAP) has been the realization, during the development of SIFT (Wensley *et al* 1978), that simple majority voting is not sufficient for obtaining 'interactive consistency' which arises in the synchronization of clocks, stabilization of inputs from sensors and

agreement on the results of diagnostic systems. It is shown by Pease *et al* (1980) that there exist protocols to guarantee 'interactive consistency'. The three classes of BAP studied in the literature are (i) deterministic BAP, (ii) randomized BAP, and (iii) approximate BAP. The deterministic BAP are capable of solving only the synchronous version of the Byzantine generals problem (BGP). Some of the randomized BAP that are reported in the literature are capable of solving both asynchronous and synchronous versions of the BGP. Approximate BAP help in situations wherein it is not possible to reach exact agreement. BAP are reported for both the cases of systems of processes that communicate with one another through (i) 'oral messages' and (ii) 'authenticated messages'. If a system can tolerate upto t failures and if $f < t$ is the actual number of failures in the system, then it is sometimes possible to stop the execution of the protocol in fewer phases than the BAP takes when t failures occur. Protocols with this property are referred to in the literature as 'early stopping protocols'. We will review the work done in the areas of 'synchronous' and 'asynchronous' systems of processes using both 'oral messages' and 'authenticated messages'.

4.1 *Deterministic Byzantine agreement protocols*

In this subsection, we consider algorithms for achieving Byzantine agreement among multiple processes. The context for this BAP is a network of unreliable processes that have a means of conducting several synchronized phases of information exchange, after which they must all agree on some set of information. Considerable work has been done in this area of 'deterministic' Byzantine agreement. Tables 1 and 2 summarize, respectively, some of the unauthenticated and authenticated BAP in terms of their performance (number of phases of

Table 1. Summary of unauthenticated BAP.

Serial Number	Reference	Performance: Number of phases	Performance: Number of messages	IBA/EBA	Remarks
1	Lamport *et al* (1982)	$t+1$	$O(n^{t+1})$	IBA	$n>3t$
2	Dolev & Reischuk (1982)	$t+1$	$\Omega(n+t^2)$	IBA	
3	Dolev *et al* (1982a)	$2t+3$	$O(nt+t^3)$	IBA	
4	Reischuk (1985)	$2f+3$	Polynomial in n and t	IBA	$n>20t$
5	Fischer & Lynch (1982)	$t+1$	—	IBA	
6	Dolev *et al* (1982b)	$t+1$	Polynomial in n and t	IBA	$n>2t^2+3t+4$
7	Dolev *et al* (1982b)	$\min(2f+5, 2t+3)$	$O(dnt^2\ V)$	EBA	n close to $3t+1$
8	Dolev *et al* (1982b)	$2t+3$	Polynomial in n and t	IBA	n close to $3t+1$
9	Dolev *et al* (1982b)	$\min(f+2, t+1)$	$O(dnt^2\ V)$	EBA	$n>2t^2+3t+4$

Table 2. Summary of authenticated BAP.

Serial Number	Reference	Performance: Number of phases	Performance: Number of messages	Performance: Number of signatures	IBA/EBA	Remarks
1	Lamport *et al* (1982)	$t+1$	$O(n^{t+1})$	—	IBA	$n>3t$
2	Dolev & Strong (1982)	$t+1$	$O(nt+t^2)$	$O(nt^2+t^3)$	IBA	—
3	Dolev & Reischuk (1982)	$t+1$	$\Omega(n+t^2)$	$\Omega(nt)$	IBA	—

information exchange and number of messages) and nature of BAP (IBA or EBA). In the tables, the symbols f, t and n denote the actual number of failures in the system, the upper bound on the number of potentially undetected faulty processes, and the total number of processes participating, respectively. The symbol d denotes the phase number at which the protocol terminates and V is the consensus vector.

4.1a *Two simple deterministic agreement protocols*: We present below Pascal-like procedures for two simple deterministic Byzantine agreement protocols developed by Lamport *et al* (1982). The first algorithm uses oral messages and the second uses digital signatures (to avoid forgery of messages) for reaching agreement. These algorithms are used with suitable modification by Lamport & Melliar-Smith (1984) for achieving fault-tolerant clock synchronization. The procedures are self-explanatory.

4.1b *Unauthenticated protocol*:

```
PROCEDURE Oral_Message_Protocol (m);
  BEGIN
    Send_Message_to_All; (*to all processes which have not acted as sender so
                          far*)
    FOR consensus := 1 TO n DO
      BEGIN
      FOR process := 1 TO n DO
          BEGIN
          Receive_Message (v_process);
          (*process receives the message from the sender
            and adds it to the set v_process*)
          (*v_process := default if not received in time*)
            m := m - 1;
          IF m > = 0 THEN Oral_Message_Protocol (m);
          (*process recursively sends messages to all other
            processes which have not yet acted as sender*)
          Find_Majority (v_process, v_consensus);
          (*return the majority of v_process in v_consensus*)
          END;
      END;
END;
```

4.1c *Authenticated protocol*:

```
PROCEDURE Signed_Message_Protocol;
  BEGIN
      Sign_the_Message; (*sender signs the message for authentication*)
      Send_to_All; (*and sends the signed message to every other process*)
      FOR process := 1 TO n-1 DO (*repeat the following for every other
                                   process*)
          BEGIN
          v_process := [ ] (*empty set*)
          FOR round := 1 TO m+1 DO (*m is the degree of fault-
          tolerance*)
            BEGIN
            Receive_Message (msg, modified);
            IF NOT modified THEN (*if the msg is not modified*)
              BEGIN
              v_process := v_process + msg;
              (*add the message msg to the set v_process*)
              Append_Sign (msg); (*process appends its signature
                to the message msg*)
              Relay_to_Others_Yet_to_Sign (msg);
              END;
          END;
          Compute_Agreed_upon_Value;
          (*decode the set v_process using a predetermined
            deterministic function to get the consistency vector*)
          END;
      END;
```

4.2 *Randomized Byzantine agreement protocols*

The randomizing BAP employ randomly chosen numbers in the execution of protocols to reach an agreement. Two possible notions of randomized BAP are discussed in the literature (Rabin 1983). One notion is concerned with the protocols which achieve Byzantine agreement with a small probability of error (in consensus value). Given $\varepsilon > 0$, we say that randomizing protocols, P_i, $1 \leq i \leq n$, are $1-\varepsilon$ reliable BAP in the presence of upto t faulty processes, if for every fixed or randomized behaviour of upto t faulty processes, the non-faulty processes reach Byzantine agreement with a probability of at least $1-\varepsilon$. We call this randomized BAP of type 1. The other notion of 'randomizing protocols' demands that for some constant c, the non-faulty processes achieve Byzantine agreement within an expected number c of phases and without any error. We call this randomized BAP of type 2.

Randomizing protocols have been studied in the literature for both 'synchronous' and 'asynchronous' versions of the Byzantine generals problem. Unlike the deterministic BAP the randomizing BAP circumvent the impossibility of Byzantine agreement for an 'asynchronous' system of processes. The notion of 'randomizing protocols' for the solution of BGP was first introduced by Rabin (1983).

Randomized BAP of types 1 and 2 are discussed by Rabin (1983). Perry (1985) considers randomization as a means of achieving early stopping of the execution of the agreement protocols. Table 3 gives a summary of some of the randomized BAP reported in the literature in terms of their performance.

4.3 *Approximate Byzantine agreement protocols*

Clock synchronization and stabilization of inputs from sensors in a process control system are two examples where approximate agreement of messages is desired (Wensley *et al* 1978). Dolev *et al* (1983) consider a variant of the traditional Byzantine generals problem, in which processes start with arbitrary real values rather than with boolean values, and in which approximate rather than exact agreement is the desired goal. Dolev *et al* (1983) present algorithms to reach approximate agreement in both 'asynchronous' and 'synchronous' systems, under the assumption of a computation model in which processes can send messages containing arbitrary real values which the processes can as well store. For any preassigned $\varepsilon > 0$, as small as desired, an approximate agreement algorithm must satisfy the following two conditions:

Agreement: All non-faulty processes eventually halt with output values that are within ε of one another.

Validity: The value output by each non-faulty process must be in the range of the initial values of the non-faulty processes.

Dolev *et al* (1983) assume the lower bounds on the number of processes for reaching approximate agreement to be $3t$ in the 'synchronous' case and $5t$ in the 'asynchronous' case. Two agreement protocols to achieve approximate agreement are presented by Mahaney & Schneider (1985), which exhibit graceful degradation when as many as 2/3 of the processes are faulty.

Table 3. Summary of randomized BAP.

Serial Number	Reference	Authenticated/ unauthenticated	Performance		Remarks
			Number of phases	Number of messages	
1	Rabin (1983)	Authenticated (synchronous & asynchronous)	4	—	$t<n/10$
2	Perry (1984)	Unauthenticated (synchronous)	3	—	$n>3t$
		(asynchronous)	3	—	$n>6t$
3	Chor & Coan (1984)	Unauthenticated (synchronous)	$O(t/\log n)$	$O(n^2t/\log n)$	$n>3t$
4	Feldman & Micali (1985)	Authenticated (synchronous)	$O(\log n)$	—	$t<n/3$
5	Broder & Dolev (1984)	Authenticated (synchronous)	$3t+3$	—	$t<n/2$

5. Applications of Byzantine agreement protocols

Though considerable amount of work has been done in the area of Byzantine agreement, there has been a controversy among the research community with regard to the applicability of Byzantine agreement protocols, mainly because of high message overhead of these protocols. Some of the applications of Byzantine agreement protocols reported in the literature are (i) clock synchronization (Wensley *et al* 1978; Pease *et al* 1980; Lamport & Melliar-Smith 1984, 1985; Mahaney & Schneider 1985; Shin & Ramanathan 1987) (ii) fault-tolerant computer for nuclear power plant operations (Lala 1986; Lala *et al* 1986) (iii) real-time computing environment (Smith 1986) and (iv) distributed database management (Garcia-Molina *et al* 1986). In this section, we consider the specific case of clock synchronization and discuss the application of Byzantine agreement protocols to this significant problem of a DCS in the presence of malicious faults.

For clarity of discussion, we begin with definitions of a few necessary terms. A 'clock' is a device that periodically makes a transition between two successive clock states. The clock states can be conceived as being numbered consecutively. The time (T) that is directly observable in a particular clock is called its 'clock time'. The 'real time (t)' is the time that is measured in the Newtonian time frame which is not directly observable. The clock can be defined as a mapping C from real time t to a clock time T such that $C(t) = T$. The inverse clock function is defined as $r(T) = C^{-1}(T) = t$. The concept of clock synchronization is defined as follows (Johnson & Butler 1984):

Two clocks r_i and r_j are synchronized within δ of each other at time T if

$$|r_i(T) - r_j(T)| < \delta.$$

Since the clocks can drift with respect to one another, it is necessary to synchronize the clocks periodically. A clock synchronization algorithm periodically resynchronizes the clocks in the system. Such an algorithm requires that each processor exchanges clock values with every other processor and processes these values to maintain sychronism. A fault-tolerant system needs a clock synchronization algorithm that works despite faulty behaviour by some processors and/or clocks. Construction of clock synchronization algorithms becomes difficult when the DCS is assumed to contain potentially malicious processors and clocks. Since the problem of clock synchronization is similar to that of agreement, a Byzantine agreement protocol with suitable modifications can solve the clock synchronization problem in the presence of malicious faults. Lamport & Melliar-Smith (1984, 1985) propose two algorithms known as 'interactive consistency' algorithms that are derived from Byzantine agreement protocols presented by Lamport *et al* (1982). The Pascal-like procedures presented in §4.1 hold good for the clock synchronization problem with the messages exchanged being the clock values. The first algorithm requires at least $3m + 1$ non-faulty clocks to handle upto m faulty clocks. The second algorithm uses digital signatures for authentication and requires at least $m + 1$ non-faulty clocks to tolerate upto m faulty clocks. Mahaney & Schneider (1985) present an 'inexact agreement algorithm' and discuss the applicability of this approximate Byzantine agreement algorithm to the clock synchronization problem. The Byzantine agreement protocol presented by Pease *et al* (1980) is made use of in the design of SIFT (Wensley *et al* 1978) computer for devising a clock synchroniza-

tion algorithm in the presence of Byzantine faults. All these algorithms assume a fully connected clocking system; that is, each and every clock in the network of clocks receives clock values from every other clock in the network. This assumption makes the design of the fault-tolerant clock synchronization mechanism complex because of the large number of interconnections required among the clocks. In a recent paper, Shin & Ramanathan (1987) propose a method that uses only 20–30% of the total number of interconnections required by the other methods that have been discussed above. The network of clocks/processors is assumed to exhibit the following properties: (i) the network of processors executes a number of jobs in parallel, (ii) each of these jobs is decomposed into a set of cooperating tasks that communicate closely with one another during the course of execution of the job, (iii) each job is assigned to a group of processors which are tightly synchronized, (iv) each group of processors is decomposed into redundant clusters, (v) each of the clusters in a group executes the same task for redundancy purposes. In view of these properties of the network, it is necessary to have intragroup and intergroup clock synchronization. The proposed method for clock synchronization makes use of the phase-locked algorithm (Krishna *et al* 1985) at two different levels. The phase-locked algorithm requires a fully connected network of $3m+1$ clocks to tolerate upto m malicious faults. By using the phase-locked algorithm, each clock synchronizes itself with respect to (i) all the clocks in its own group, and (ii) one clock from each of the other groups. This method greatly reduces the number of interconnections required for clock synchronization and seems to be promising in the context of a large network of processors.

6. Conclusions

The problem of obtaining 'interactive consistency' is one of the fundamental issues in the design of fault-tolerant distributed computing systems. It is realized by researchers that simple majority voting does not solve this problem, especially when the distributed computing system comprises potentially malicious components. When the problem of achieving extremely high reliability of the distributed computing system is faced, the Byzantine agreement protocols provide a solution to this problem. However, the solution seems to be inherently expensive. It is felt that the ongoing research in the areas of randomized Byzantine agreement protocols and the early stopping protocols will provide less expensive Byzantine agreement protocols.

The encouragement extended by Mr P S Goel, Indian Space Research Organisation Satellite Centre, Bangalore, during the course of this work, is gratefully acknowledged.

References

Attiya C, Dolev D, Gil J 1984 *Proc. ACM Symp. Principles of Distributed Computing* (New York: ACM Press)

Broder A Z, Dolev D 1984 *Proc. IEEE Symp. Foundations of Computer Science* (Silver Spring, MD: IEEE Comput. Soc. Press)

Chor B, Coan B A 1984 *Proc. IEEE Symp. Reliability in Distributed Software and Database Systems* (Silver Spring, MD: IEEE Comput. Soc. Press)

Cristian F, Aghili H, Strong R 1984 Atomic Broadcast from Simple Message Diffusion to Byzantine Agreement, IBM Res. Rep. RJ4540(48668)

Dolev D, Fischer M J, Fowler R, Lynch N A, Strong H R 1982a *Inf. Control* 52: 257–274

Dolev D, Lynch N A, Pinter S S, Stark E W, Weihl W E 1983 *Proc. IEEE Symp. Reliablity in Distributed Software and Database Systems* (Silver Spring, MD; IEEE Comput. Soc. Press)

Dolev D, Reischuk R 1982 Bounds on Information Exchange for Byzantine Agreement, IBM Res. Rep. RJ3587(42133)

Dolev D, Ruediger R, Strong H R 1982b *Proc. IEEE Symp. Foundations of Computer Science* (Silver Spring, MD: IEEE Comput. Soc. Press)

Dolev D, Strong H R 1982 *Proc. IEEE Symp. Reliability in Distributed Software and Database Systems* (Silver Spring, MD: IEEE Comput. Soc. Press)

Dolev D, Strong H R 1983 *SIAM J. Comput.* 12: 656–666

Ezhilchelvan P D, Shrivastava S K 1986 *Proc. IEEE Symp. Reliability in Distributed Software and Database Systems* (Silver Spring, MD: IEEE Comput. Soc. Press)

Feldman P, Micali S 1985 *Proc. IEEE Symp. Foundations of Computer Science* (Silver Spring, MD: IEEE Comput. Soc. Press)

Fischer M J 1983 *Proc. Int. Conf. Foundations of Computing Theory* (New York: ACM Press)

Fischer M J, Lynch N A 1982 *Inf. Process. Lett.* 14: 183–186

Garcia-Molina H, Pittelli F, Davidson S 1986 *ACM Trans. Database Syst.* 11: 27–47

Johnson S C, Butler R W 1984 *AIAA/IEEE 6th Digital Avionics Systems Conference* (New York: IEEE Press)

Krishna C M, Shin K G, Butler R W 1985 *IEEE Trans. Comput.* C-34: 752–756

Lala J H 1986 *Fault-tolerant Comput. Symp.-16* (Silver Spring, MD: IEEE Comput. Soc. Press)

Lala J H, Alger L S, Gauthier R J, Dzwonezyk M J 1986 *AIAA/IEEE 7th Digital Avionics Systems Conference* (New York: IEEE Press)

Lamport L 1984 *ACM Trans. Program. Lang. Syst.* 6: 254–280

Lamport L, Melliar-Smith P M 1984 *Proc. ACM Symp. Principles of Distributed Computing* (New York: ACM Press)

Lamport L, Melliar-Smith P M 1985 *J. ACM* 32: 52–78

Lamport L, Shostak R, Pease M 1982 *ACM Trans. Program. Lang. Syst.* 4: 382–401

Mahaney S R, Schneider F B 1985 *Proc. ACM Symp. Principles of Distributed Computing* (New York: ACM Press)

Pease M, Shostak R, Lamport L 1980 *J. ACM* 27: 228–234

Perry K J 1984 *Proc. IEEE Symp. Reliability in Distributed Software and Database Systems* (Silver Spring, MD: IEEE Comput. Soc. Press).

Perry K J 1985 Early Stopping Protocols for Fault-Tolerant Distributed Agreement, TR 85–662, Dept. of Computer Science, Cornell University

Rabin M O 1983 *Proc. IEEE Symp. Foundations of Computer Science* (Silver Spring, MD: IEEE Comput. Soc. Press)

Reischuk R 1985 *Inf. Control* 64: 23–42

Shin K G, Ramanathan P 1987 *IEEE Trans. Comput.* C-36: 2–12

Smith T B *Fault-tolerant Comput. Symp.-16* (Silver Spring, MD: IEEE Comput. Soc. Press)

Strong R 1985 *IEEE COMPCON* (Silver Spring, MD: IEEE Comput. Soc. Press)

Toueg S 1984 *Proc. ACM Symp. Principles of Distributed Computing* (New York: ACM Press)

Turpin R, Coan B A 1984 *Inf. Process. Lett.* 18: 73–76

Wensley J H, Lamport L, Goldberg J, Green M W, Levitt K N, Melliar-Smith P M, Shostak R E, Weinstock C B 1978 *Proc. IEEE* 66: 1240–1255