94

REACHING AGREEMENT IN THE PRESENCE OF FAULTS : "CONSENSUS IN DISTRIBUTED SYSTEMS"

Dissertation submitted to
Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND TECHNOLOGY

by
JYOTSNA BEHL

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWHARLAL NEHRU UNIVERSITY
NEW DELHI-110 067,
JANUARY 1992

CERTIFICATE

This is to certify that the thesis entitled **"REACHING AGREEMENT IN PRESENCE OF FAULTS : CONSENSUS IN DISTRIBUTED SYSTEMS"** being submitted by me to Jawaharlal Nehru University in partial fulfilment of the requirements for the award of the degree of Master of Technology, is a record or original work done by me under the supervision of Dr. P.C. Saxena, School of Computer and System Sciences, during the Monsoon semester, 1991.

The results reported in this thesis have not been submitted in part or full to any other University or Institution for the award of any degree etc.

Jyotsna Behl
(JYOTSNA BEHL)

[illegible]
Dr. R.G. Gupta 3\1\92
Dean,
SC&SS. J.N.U.,
New Delhi - 110 067.

[illegible]
Dr. P.C. Saxena
SC&SS. J.N.U.,
New Delhi - 110 067.

ACKNOWLEDGEMENTS

I wish to express my sincere and heartfelt gratitude to Dr. P.C. Saxena, School of Computer and Systems Sciences, Jawaharlal Nehru University, for the unfailing support he has provided through out. In all respects, Iam very grateful for the patience he has exhibited and for the time he has spent with me discussing the problem. It would have been impossible for me to come out successfully without his constant guidance.

I extend my thanks to Prof. R.G. Gupta, Dean, School of Computer and Sytem Sciences, JNU for providing me the oppurtunity to undertake this project. I would also like to thank the authorities of our school for providing me the necessary facilities to complete my project.

I acknowledge and thank each and everyone of those who, directly or indirectly, helped me in this work.

(JYOTSNA BEHL)

PREFACE

To understand the problem raised by distributed working is to understand how tomorrow's information system will work. Just as there has come to be a standard structure for a car, agreed to by all manufacturers as well understood by all users, there is no doubt that these future systems will consist of machines of a wide variety of types distributed around a ring, raging from individual work station to a centralized file store, with general or special-purpose processors in between. Thus if we want all the processes to be implemented in order to mobilise, co-ordinate and stop the many activities of a distributed system, we shall understand the problems faced and a need of standards.

Keeping the above in mind, the thesis centers around the need of standard algorithms dealing with different problems of distributed systems. In my thesis, I have considered only the problem of **"Reaching Agreement in the Presence of faults"**.

In this thesis, the algorithms have been investigated for both synchronous and asynchronous system. The algorithms considered include solutions for interactive consistency problem, solutions for Original and Weak Byzantine General problem, Asynchronous consensus and broadcast protocols, variants of Byzantine general problem by assuming starting values as reals, simple constant time consensus protocols in realistic failure models, simultaneous byzantine agreement, eventual byzantine agreement, etc.

The thesis is divided into 5 Chapters in the following fashion; Chapter1, has the introduction to the subject and an abstract of the thesis. Chapter2 has general notations and definitions, theorems, lemmas along with their proofs are dealt with in Chapter 3.

While going through the thesis, the reader may want to skip over Chapters 2 & 3 and go directly to Chapter 4, in which the problem is explained in detail. While reading Chapter 4, you may revert back to Chapters 2 & 3 for referring to notations, definitions, theorems, lemmas and proofs.

For sake of convienance the definitions are arranged alphabetically. Also in Chapter 3 you will find that the theorems appear in the same order as they do in the text of Chapter 4.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

The field of distributed applications is constantly growing. This increase in the use of computer science as a prefered tool in ever more diverse areas is essentially the result of developments in this discipline.

The control of distributed applications is based on understanding of what is known as "DISTRIBUTED SYSTEM". Compared to CENTRALIZED OPERATING SYSTEM, DISTRIBUTED SYSTEM differ essentially since the entities that form the latter cooperate in the achievement of common aim by exchanging messages, and thus there is no global state in the system that can be detected instantly by one of these entities.

The problems related to distributed system include :

(a) Mutual dependency of logical clocks

(b) Network routes

(c) Learning distributed information

(d) Determination of global states.

(e) Maximum delay in the transfer of messages

(f) Topological structure of the network

(g) Distributing a global synchronization constrain

(h) Reaching agreement in presence of faults

(i) Mutual exclusion

(j) and many more.

In this thesis, algorithms for reaching agreement in the presence of faults have been investigated for both synchronous and asynchronous system. The algorithms considered include solutions for interactive consistency problem, solutions for Original and Weak Byzantine General problem, Asynchronous consensus and broadcast protocols, variants of Byzantine general problem by assuming starting values as reals, simple constant time consensus protocols in realistic failure models, simultaneous byzantine agreement, eventual byzantine agreement, etc.

Fault-tolerate systems often require a means by which independent processors or processes can arrive at,an exact mutual agreement of some kind. It may be necessary, for example, for the processors of a redundant system to synchronize their internal clocks periodically. Or they may have to settle upon a value of a time-varying input sensor that gives each of them a slightly different reading. In the absence of faults, reaching a satisfactory mutual agreement, is usually an easy matter. In most cases it suffices simply to exchange values, (CLOCK TIME, in the case of clock synchronization) and compute some kind of average. In the presence of faulty processors, however, simple exchanges cannot be relied upon, a bad processor might report one value to a given processor and another value to some other processors, causing each to calculate a different "average."

One might imagine that the effect of faulty processors could be dealt with through the use of voting schemes involving more than one round of information exchange; such schemes might force faulty processors to reveal themselves as faulty or at least to behave consistently enough with respect to the nonfaulty processors to allow the latter to reach an exact agreement. As we will show, it is not always possible to devise schemes of this kind, even if it is known that the faulty processors are in a minority. Algorithms that allow exact agreement to be reached by the nonfaulty processors do exist, however, if they sufficiently outnumber the faulty ones.

The Byzantine generals problem involves obtaining agreement among a collection of processes, some of which may be faulty. The precise defination is given in the chapter 2.

Nonfaulty processes are assumed to correctly follows their alogrithm, but faulty processes may do anything. We assume that the absence of a message is detectable, which is equivalent to assuming that a faulty process sends every message that it is supposed to--although it need not send the correct message. The difficulty of the problem lies in the fact that a faulty process may send conflicting information to two different processes.

This problem was described in [2_1] in term of byzantine general metaphor, hence its name. Essentially the same problem appeared in [1_0], where it is called the interactive consistency problem. In [1_0] the problem was shown there to be solvable if and only if fewer than one-third of the processes are faulty, no solution works for three processes in the presence of a single fault.

In the work done by Lamport in [2_0] we consider a weaker version of the problem, in which condition (1) is replaced by :

(1) If all processes are nonfaulty, then every process i obtains the value v

A form of agreement problem quite well-known is the the transaction commit problem, which occurs in distributed database systems, all the data manager processes which have participated in the processing of a particular transaction to agree on whether to install the transaction's results in the database or to discard them. The latter action might be necessary, for example, if some data managers were unable to carry out the required transaction processing. Whatever decision is made, all data managers must make the same decision in order to preserve the consistency of the database.

The transaction commit problem for a database is an instance of this weaker problem, in which process 0 represent a transaction coordinator, and the other processes represent the database sites affected by the transaction [2_3].

Any solution to the original Byzantine Generals Problem is obviously a solution to the Weak Byzantine Generals (WBG) Problem so the WBG Problem solvable if fewer than one-third of the processes may be faulty.

The Byzantine Generals Problems arises in practice when trying to get the nonfualty processes to agree upon the value of some input quantity. The WBG problem arises when trying to get the nonfaulty processes simply to agree, regardless of what they agree upon. To eliminate the trivial possibility of having them agree upon a prearranged value, we can assume that each process chooses a Private value, and that these private valuesare used in reaching agreement,can then be formulated as WEAK INTERACTIVE CONSISTENCY PROBLEM, defined in the next chapter.

A consensus protocol enables a system of n asynchronous process, where some of the process are faulty, to reach to an agreement.

Next we discuss protocols that enable a fully interconnected system with reliable asynchronous transfering, to reach an agreement. In reliable asynchronous message system, a message can be have arbitrary delays.

The faulty processes considered, fall under two classes

1) Fail stop processes : A process may stop participating in the protocol that is, it may just Die.
2) Malicious Process : A Faulty process may send incorrect messages.

A fail stop process creates problem even though it doesn't send any false messages, because there is no way to find a difference between a dead process and a slow one. Whereas in case of a malicious process, the contradictory messages may cause trouble for the distributed system.

The system involving fail-stop processors, was investigated in [3_0]. Fischer etal. It showed the impossibility of a consensus protocol if only one failure may occur. However, in [3_0], the concept of an admissible solution is a protocol that always terminates within a finite number of steps. In Module 4 we are interested in a different kind of solution: we consider protocols , which may never terminate, but this would occur with probability 0, and the expected termination time is finite. There are two ways to introduce probabilities on the possible executions of a protocol. The other approach, and the one we adopt in this thesis, is to postulate some probabilistic behavior about the message system.

In the case of fail-stop processors, we describe a probabilistic protocol that can withstand up to $\lfloor (n-1)/2 \rfloor$ failures, where n is the number of processes. We also show that there is no consensus protocol that can withstand up to $\lfloor (n-1)/3 \rfloor$ processes may fail. Agreement among remote processes is one of the most fundamental problems in distributed computing and is at the centre of many algorithms for distributed data processing, distributed file management, and fault tolerent distributed applications.

One may also consider more Byzantine types of failure in which faulty processes might go completely haywire, perhaps even sending messages according to some malevolent plan. We would like to have an agreement protocol which is as reliable as possible in the presence of such faults. No completely asynchronous consensus protocol can tolerate even a single unannounced process death. We may not consider Byzantine failures,further-more we assume that the message system is reli-able and that it delivers all messages correctly and exactly once. Still with these assumptions, the stopping of a single process at an inopportune time can cause any distributed commit protocol to fail to reach agreement. Hence this crucial problem has no convincing solution or still greater restrictions on the kind of failures to be tolerated!.

As crucial to the proof that processing is completely asychronous we cannot make assumptions about the relative speeds of processes or about the delay time in delivering a message. We can also assume that processess don't have access synchronized clocks, so algorithms based on time-outs, for example, cannot be used. Finally we cannot postulate the ability to detect the death of a process, so it is impossible for one process to tell whether another has died or is just running very slowly.

The impossibility in the result applies to even a very weak form of the consensus problem. Let us assume that every process starts with an initial value in {0, 1}. A nonfaulty process decides on a value in {0,1} that is by entering an appropriate decision state. Also all nonfaulty processes which make a decison are to choose the same value. For the purpose of the impossibility proof, we want only that some process eventually make a decision. The solution in which , 0 is always chosen is ruled out by saying that both 0 and 1 are possible decision values, although perhaps for differrent initial configurations.

Process are also modelled as automata, one atomic step. Every message is finally sent as long as the destination process makes infinitely many attempts to receive, but messages can be delayed, arbitrarily long,and sent out of order.

In module five we assume a different kind of model.

We assume a model in which processes can be send messages containing arbitary real values and store arbitray real values as well.

We assume that each process starts with an arbitary real value. For any preassigned t > 0 (as small as desired), an approximate agreement algorithm must satisfy the following conditions:

(a) Agreement: All nonfaulty processes eventually halt with output values that are within of each other.

(b) Validity: The output by each nonfaulty process must be in the range of initial values of the nonfaulty processes.

Thus, in particular, if all nonfaulty processes should happen to start with the same initial value, the final values are all required to be same as the common initial value. This is consistent with the usual requirments by Byzantine agreement algorithms. However, should the nonfaulty processes start with different values, we do not require that the nonfaulty processes agree on a unique final value.

We consider both synchronous and asynchronous versions of the problem. Systems in which there is a finite bounded delay on the operations of the processes and on their intercommunication are said to be synchronous. In such systems, unannounced process deaths, as well as long delays, are considered to be faults. For synchronous system, we give a simple and rather efficient algorilthm for achieving approximate agreement. This algorithm works by successive approximation, with a provable convergence rate that depends on the ratio between the number of faulty processes and the total number of processes are allowed to terminate at different times.

For asynchronous systems, in which a very slow process cannot be distinguished from a dead process, exact agreement cannot be reached by an algorithm that is guaranteed to terminate [5_5, 5_9]. Exact agreement can, however be attained by algorithms that only teminate with probability 1 [5_1, 5_3]. An interesting constrast to the results in [5_5] and [5_9] is the second algorithm, which enables processes in an asychronous system to get as close to agreement as one chooses.

Our algorithm for the asynchronous case also works by successive approximation. In this case, however , the total number of processes required by the algorithm is more than five times the number of possible faulty processes. As in the synchronous case we acheive termination using a technique that ensures that all nonfaulty processes halt but permits different processes to terminate at different times. Our algorithm for obtaining approximate aggrement are of a very simple form. Namely, at each round, until termination is reached, each process sends its latest value to all processes (including itslef). On receipt of a collection V of Values, the process computes a certain function f(V) as its next value. The function f is a kind of averaging function. Here we use functions that are appropriate for handling t faulty processes.
Particularly for handling t faulty processes, We show that these funcations have particularly nice approximation behaviour. In

particular, we show that, for algorithms of a specific form, no apprioximation function can provide uniformly faster convergence than the functions used in [5_0]. [5_6] presented similar algorithms but used approximation functions that provided slower convergence than is achieved by the functions used in [5_0].

Randomization has proven to be an extremely useful tool in the design of protocols for distributed agreement. New randomized protocols for the consensus problem in synchronous and asynchronous fail-stop and failure-by-omission models are presented in the thesis. These protocols terminate within constant expected time, and unlike previous fast protocols, are very simple and need not rely on any preprocessing. Infact, we believe that these protocols will be the method of choice in practical implementations. The major novelty of algorithms developed in [6_0] is the notion of a weak form of a global coin, and a method of generating it.

The situation for deterministic alogrithms for consensus is well understood. A result of Dolev and Strong implies that in asynchronous fail-stop model, at least t + 1 rounds are needed, in the worst case, to achieve consensus;they also provided an algorithms that achieves this bound and transmits only a polynomial number of messages [6_10]. In an asynchronous model, Fischer et al. showed that no protocol exists for consensus in the fail-stop model that tolerates even a single fault [6_14].

Fortunately, randomization can overcome this inherent intractability. Ben-Or describes a protocol for asynchronous consensus that tolerates upto n/2 faults in the fail-stop model, and terminates with probability 1 [6_4]. Results of a similar nature were given by Bracha and Toueg [6_6]. However the expected number of rounds needed to reach agreement as maesured locally by every processor is exponential in the asynchronous case (can be shown to be O(t/ $\sqrt{n}$) in the synchronous one). Rabin introduced the important notion of a global coin flip [6_22],which is a coin flip whose outcome is visible to all processors. He describes a different protocol that employs such a coin,so that each processor can use the outcome of a common coin. The expected number of rounds to reach agreement is O(T (n)), where T(n) is the time required to flip the coin in a network of n processors. In order to implement his global coin, Rabin required some predealt information to be distributed by a trusted third party. Bracha, using a beautiful "boot-strapping"construction, showed that Rabin's result could be improved so that agreement is reached in O(T(log n)) expected number of rounds (i.e., the time to flip many independent coins in subnetwork whose size is logarithmic in n, in the size of original network) [6_5]. It has been shown how to use cryptographic techniques to implement such a coin-toss in T(n) = O(n) rounds, so that overall, Bracha's procedure can be run in O(log n) expected time as shown in [6_3] and by A.C. Yao, private communication . However, this scheme requires an assignment of processors to committees for which no explicit construction is known. In contrast, the protocol given by Benny Chor in [6_0] is completely constructive. Feldman and Micali [6_12] have also the nonconstructive part of Bracha's probabilistic assignment, by having the processors generate the assignment themselves.However, in the process, Feldman and Micali introduced a preprocessing phase that requires O(T)rounds.Their protocol is superior to deterministic protocols in an amortized sense,since additional agreements require only O(1) time.The best-known bound for a Byzantine fault model without predealt information or preprocessing is O(log n). Since the alogrithms given in [6_0], for omission faults run in constant expected time, current results leave a log n seperation between the Byzantine and omission fault models.

Finally in the last module of the thesis, I discuss two closely-related types of agreement that can be reached in a distributed system in the presence of undetected processor faults.

One type is called Simultaneous Byzantine Agreement (SBA) and the other Eventual Byzantine Agreement (EBA). Corresponding to these two types of agreement, are two distinct problems in coordination among multiple processors in a distributed system. One problems is synchronization: Processors may be required to perform some action at the same time, immediately after reaching agreement on that action [7_18]. The other is consistency as required, for example, in the atomic commitment of a distributed database transaction. The participants in the transaction commit protocol must agree on whether or not the transaction is to be committed. In this case, it is enough to know that the choice will eventually be the choice of all other parties to the agreement [7_10].

The difference between these two problems and the consequent differences in requirements for their solution is discussed in the thesis. Because SBA implies EBA within the model considered, EBA can always be reached as early as SBA. It is shown that EBA can often be reached earlier than SBA.

This thesis provides a complete survey report of work done in the field, starting from the year 1980 to 1990. It presents the protocols with detailed proofs whereever found necessary.

CHAPTER TWO

This chapter is compiled for reference with respect to "Glossary of Notations" and "Definitions". The Definitions are arranged alphabetically to permit easy access to any definition as in a dictionary.

GLOSSARY OF NOTATIONS

General Notations :

o<	:	is Greek letter Alpha
o-	:	is Greek letter Sigma
$	:	is Greek letter Beta
II	:	stands for big Pi
#	:	stands for small Pi
%	:	stands for Phi
&	:	stands for Delta
@	:	stands for Si
A	:	stands for Rho
⅄	:	stands for Lemda
n	:	the number of processes.
P	:	the set {0,.....,n-1} of processes.
P*	:	the set of finite sequence of processes. (elements of P including empty set)
II	:	the set of message paths from 0. This implies a sequence in P* begining with 0.

IIi	:	the set of message paths from 0 to i, that is a sequence in II ending in i.
II(k)	:	set of message paths of length <= k in II.
IIi(k)	:	set of message paths of length <= k in IIi.
V	:	the set of all possible values v.
scenario	:	a mapping % : II --> V , specifying the value of the content of every message. if # belongs to II then %(#) gives content of message received at final destination of path II .
i-scenario:		a mapping %i : IIi --> V , the part of a scenario "seen" by Process i.
WBG algorithm B:		a collection of mappings Bi from i-scenarios to V, such that for any scenario % in which atleast n-m processes are nonfaulty (1) If all processes in P are nonfaulty in % then for all i belonging to P , Bi(%i) = %(0) . (2) If any i , j belonging to P are nonfaulty in % than Bi(%i) = BJ(%J)

<u>Notation for n=3.:</u>

&(i,j)	:	the signed, clockwise distance from i to j.
o-(#)	:	the signed angular distance travelled by the path #.
$\overline{r}$	:	r mod 3.
$II^{(r)}$	:	a scenario in which a faulty Process $\overline{r}$ relays the value F to $\overline{r - 1}$ and the value T to $\overline{r + 1}$.

<u>Notation used in proof of Theorem 2.1:</u>

P'	:	the set of process {0',1',2'}
⅄	:	a mapping assigning to each process in P a process in P', which assigns at most m proceses to each process in P'. Also its extension to a mapping from message paths in P to message paths in P'.
i"	:	an element in P that is assinged by | to i", where i = 0,1 or 2.
/\	:	a mapping that takes scenarios on P' into scenarios on P, defined by letting the value of /\ [%'] on the message path # equal the value of %' on the path | (#).

ADMISSIBLE RUN : We may add that a run is admissible if at most one process is faulty and that all messages sent to non-faulty processes are eventually received.

AUTHENTICATORS: An authenticator is redundant augment to a data item that can be created ideally, only by the originator of the data.

A SYNCHRONOUS APPROXIMATION ALGORITHM P : It is a system of n processes, n>=1. Each process p has a set of states, including a subset of states called initial states and subset called halting states. There is a value mapping that assigns a real number as the value of each state. For each real number r, there is exactly on initial state with value r. Each process acts deterministically according to a transition function and a message generation function. The transition function takes a nonhalting process state and a vector of messages received from all processes (one message per process state. The message generation function takes a nonhalting process state and a vector of messages received from all processes(one message per process) and produces a new process state. The message generation fuction takes a nonhalting state and produces a vector of messages to be sent to all processes (one per process).

<u>ASYNCHRONOUS FAILURES:</u> Except for a set of at most t sending processors, all messages sent by every processor are eventually dellivered.

<u>ATOMIC STEPS :</u> In an atomic step of the system, a process can attempt to

*** Receive a message

*** Perform local computation on the basis of whether or not a message was delivered to it and if so which one.

*** Send an arbitrary but finite set of messages to other processes.

The protocol prescribes the computation & the message sent as a function of the message recieved & the local state.

BYZANTINE GENERALS PROBLEM: given a collection of processes numbered from 0 to n - 1 which communicate by sending messages to one another, to find an alogrithm by which process 0 can transmit a value v to all the processes such that:

(1) If process 0 is nonfaulty, then any nonfaulty process i obtains value v.

(2) If processes i and j are nonfaulty, then they both obtain the same value.

Note that condition 2 follows from 1 if process 0 is nonfaulty.

CANDIDATE: A prdcessor p is said to be a candidate in round i of a history H if p does not fail before round i and if H and the silencing of p at round in of H are serial. Note that if p fails in round i` of H are serial.

Note that if p fails in round i of serial history H, then p is a candidate in round i of H. However p can be both correct in H and a candidate in run in of H.

CONFIGURATION : of the system consists of :

*** Internal state of each process

*** Contents of message buffer.

F^{i} denotes any configuration where all the correct processes have decided i.

CONSENSUS PROTOCOL : A consensus protocol enables an asynchronous system of n processes, with some faulty proceses, to reach an agreement.

CONSERVATIVE EXTENSION: If Hk is an initial sequence of a history in U, then the conservative extension of Hk is the unique history H' in U such that

(1) H'k = Hk, and

(2) no processor fails after round K.

CORRECT PROCESS : A correct process is a process which always follows the protocol until the protocol completion. However a fail-stop process may die during the execution of the protocol,i.e. it may stop participating in the protocol, also death of a process occurs without warning messages. In the model, it is obvious that such a death can not be detected by other processes. In particular, there is no way to differentiate between a slow process & a dead one.

CRITICAL HISTORY: An edge e in round k of history H is critical if there is a history J in U such that

1) J is not output equivalent to H.

2) J is identical to H through round k except for edge e, and

3) J is the conservative extension of Jk

In other words, and edge is critical if altering the state of its message and taking the conservative extension alters the output values of correct processors. Note that A must specify for any critical edge.

DECIDING RUN : A run is said to be a deciding run if some process reaches a decision state in that run.

DECISION STATE : is the state in which the output register has value 0 or 1.

DECISION VALUE : A configuration C has decision value v if some process p is in a decision state with yp = v.

EVENT : The step is completely determined by the pair e=(p, m), which is called an <u>event</u>.

EVENTUAL AGREEMENT: The processors are said to have reached agreement when the following two conditions hold:

(1) all correct processors have given the same value as output,

(2) if the origin is correct, then all correct processors have given the input value as output.

These two conditions define Byzantine agreement [7_21].

We call such a state eventual agreement, emphasizing the fact that nothing is assumed about the relative times at which the correct processors give their output values.

FINITENESS OF A WBG ALGORITHM B: A WBG alogrithm B is said to be finite if every scenario % there is an integer k such that for any scenario @ and all i belonging to P such that if the restrictions of %i and @i to $II^{(k)}$ are equal, then Bi(%) = Bi(@).

f-CANDIDATE : A processor p is said to be an f-candidate in round i of history H is p does not fail before round i, and if both H and the silencing of p in round i of H are f-serial.

f-SERIAL HISTORY : A history H is said to be f-serial if H is in U, H has no more than f faults, for each positive integer k<=f+1, the number of processors exhibiting faulty behaviour in Hk does not exceed k, and no processor fails in H after round f+1.

INITIAL CONFIGURATION : is said to be one in which each process starts at an initial state and empty message buffers.

INITIAL STATE : The system initially starts with all the processes in some **initial state**, with all the buffers empty, Yp undefined, and Xp having some value in {0, 1}. The value can be assigned to Yp form {0, 1} by the protocol. Once Yp is assigned a value v it cannot be changed, and P is said to have decided v.

INTERACTIVE CONSISTENCY : (Defination) Consider a set of N isolated processors, of which it is known that maximum M are faulty. Which processors are faulty is not known.
Assuming two-party messages system and the communicatin medium to be fail-safe and of negligible delay. Also the sender of a message is always identifible by the receiver. Let each processor P has some private value of information Vp.
An algorithm for m,n > 0, based on an exchange of messages that allows each nonfaulty processor P to compute a vector of values with an element for each of the N processors, such that:

(a) the nonfaulty processors compute exactly the same vector,

(b) the element of this vector corresponding to a given nonfaulty processsor is the private value of that processor.

Such an algorithm is said to achieve interactive consistency, since it allows the nonfaulty processors to come to a consistent view of the values held by all the processor, including the faulty ones.

Note that the algorithm need not reveal which processor are faulty, and that the element of the computed vector corresponding to faulty processors may be arbitrary, it matters only that the nonfaulty processsors compute exactly the same value for any given faulty processor.

The computed vector is called an interactive consistency (i.c.) vector.

Once interactive consistency has been achieved, each noonfaulty processor can apply an averaging or filtering function to the i.e. vector, according to the needs of the application. Since each nonfaulty processor applies this function to the same vector of values, an exact agreement is necessarily reached.

INTERACTIVE CONSISTENCY FOR M FAULTS: For each p belonging to P, let Fp be a mapping that takes a p-scenario o-p and a processor q as arguments and returns a value in V. Intuitively, Fp gives the value that p computes for the element of the interactive consistency vector correspoding to q on the basis of o-p. We say that {Fp|p belonging to P} assures interactive consistency for m faults if for choice of N subset of P, |n| >= n-m, and each scenario o- consistent with N,

(1) for all p,q belonging to N,
Fp(o-p,q)=o-(q),

(2) for all p,q belonging to N,
r belonging to P,
Fp(o-p,r)=Fq(o-q,r).

where o-p and o-q denote the restrictions of o- to strings beginning with p and q, respectively.

Instuitively, clause (1) requires that each non faulty processor p correctly computes the private value of each nonfaulty processor q, and clause (2) requires that each two nonfaulty processors compute exactly the same vector.

INTERNAL STATE : Internal state of a process constitutes

*** values in Xp and Yp

*** Program counter

*** Internal storage.

i-SCENARIO : An i-scenario is a mapping from IIi to V thus it describes the messages received by process i. For any scenario %,we let %i, be the i-scenario that is the restriction of % to IIi, so %i, is the part of % "seen" by process i.

K-LEVEL SCENARIO: Let p be the set of processors and v a set of value for k>=1, we define a k-level scenario as a mapping from the set of nonempty strings (possibly having repetitions) over p of length <= k+1, to v. For a given k-level scenario o- and string w = p1p2...pr, 2<+r<=k+1, o-(w) is interpreted as the value p2 tells p1 that p3 told p2 that p4 told p3...that pr told pr-1 is pr's private value. For a single-element string p, o-(p) simply designates p's private value vp. A k-level scenario thus summarizes the outcome of a k-round exchange of information.

K-RESILIENT CONSENSUS PROTOCOL : is defined as a protocol that satisfies the following properties, provided at any time maximum number of faulty porocesses is K.

(1) Bivalence: If all the processes are correct and both F^0 and F^1 configurations are reachable.

(2) Consistency : Here there is no reachable configuration where correct processes decide different values.

(3) Convergence : For any initial configuration,

$$\lim_{t \to \text{infinity}} \Pr[\text{a correct process has not decided within t steps}] = 0$$

MESSAGE SYSTEM: The different processes may communicate by sending messages to each other via the <u>message system</u>. A <u>message</u> consists of the pair (p, m) where p denotes the name of the destination process. "m" denotes the "message value" from a fixed universe M. The message system maintains a system buffer for each process for messages sent not yet recieved called <u>message buffer</u>. It also includes the following primitives for each process q.

<u>Send(p,m)</u>: Immediately place the message m in process p's buffer

<u>Recieve(m)</u>: Either :

(1) It take out some message from q's buffer and return it in m

or

(2) It return the null value even when the buffer is not empty. It is a device to model the arbitrarily long transmission delays spent in a message system.

The choice of (1) or (2) is made nondeterministic subject only to the condition that if receive(m) is performed infinitely many times, then every messages (p , m) in the message buffer is eventually delivered.

The message system is allowed to return null a finite number of times in response to receive(m) even if a message (p,m) is available in the buffer.

m-FAULT WBG ALGORITHM B: An m-fault WBG algorithm B consists of a set of mappings Bi from i scenario values received by destination into V, for all i belonging to P, such that for any scenario % in which at least n - m processes are non faulty:

(1) If all processes in P are nonfaulty in %, then for all i belonging to p such that: Bi(%i) = %(0).

(2) For any i, j belonging to P such that if i and j are nonfaulty in %, then Bi(%i) = BJ(%J).

MULTISET RELATED DEFINATIONS : Let N be the natural numbers, including 0, and let R be the real numbers. We view a finite multiset U of reals as a function U: R --> N that is nonzero on at most finitely many r belonging to R. Intuitively, the function U assigns a finite multiplicity to each value r belonging to R. The cardinality of a multiset U is given by

$$\sum_{(r \text{ belonging to } R)} U(r)$$

and is denoted by |U|. We say that a multiset is empty if its cardinatly is zero; otherwise it is nonempty.

The difference U-V of multisets U and V is the multiset W defined by

$$W(r)=\begin{cases} U(r) - V(r) & \text{if } U(r) - V(r) >= 0 \\ 0 & \text{otherwise.} \end{cases}$$

The intersection of multisets U and V is the multiset W defined by

W(r) = min (U(r), V(r)).

For the following definations, it is assumed that the term multiset always refers to finite multisets of real numbers.

If g is a function on multisets, then g^k denotes the k-fold iteration of g; thus $g^1 = g$, $g^2 = g \circ g$, etc.

The minimum min(U) of a nonempty multiset U is defined by

min(U)=min{r belongs to R:U(r) is non-0}

The maximum max(U) is defined similarly. If U is nonempty, let p(U) (the range of U) be interval [min(U), max(U)], and let &(U) (the diameter of U) be max(U) - min(U). The mean mean(U) of a nonempty multiset U is defined by

$$\text{mean}(U) = \sum_{(r \text{ belonging to } R)} \frac{r\,U(r)}{|U|}$$

If U is nonempty multiset, we define the multiset s(U) (intuitively, the multiset obtained by removing one occurence of the smallest value in U) to be the multiset W defined by

$$W(r) = \begin{cases} U(r) - 1 & \text{if } r = \min(U) \\ U(r) & \text{otherwise} \end{cases}$$

The multiset l(U) (remove one occurrence of the largest value in U) is defined similary. If |U| >=2, then define reduce(U) = s(l (U)), the result of removing the largest and smallest element is removed from each.

ORDERLY CRASH FAILURE: is a crash failures in which failing processors must respect the order specified by the protocol in sending messages to neighbour. (Recall that for each round a protocol produces an ordered set of labeled outedges that we identify with messages to be sent.)

If a processor fails to send a specified message, it must also fail to send any message specified to be sent after that message in the protocol odèring.

OUTPUT EQUIVALENCE: Here, we take the transitive closure of the relation that holds between H and J (or Hk and Jk) when some processor correct in both gives the same output value in both.

PARTIAL CORRECTNESS OF CONSENSUS PROTOCOL: A consensus protocol is partially correct if it satisfies two conditions.

1) No accessible configuration has more than one decision value.

2) For each v { 0, 1 }, some accessible configuration has decision value v.

PATTERN : A pattern (for a history) is a function from the set of faulty processors to integers that gives the round number at which each processor failed.

REACHABLE FROM : If all the processes performing atomic steps in t belong to a subset of processes S, then we write Cs |---- Ds, and say that Ds is **reachable from** Cs.

The configuration D is said to be **reachable** if it is reachable from some initial configuaration.

REGISTERS : Each process P has the following :

*** One bit **Input Register** Xp.

*** **Output Register** Yp with values in { b, 0, 1 }.

This is also called Decision

*** unbounded amount of **Internal storage**

RUN : is the sequence of steps associated with a schedule.

SCENARIO: is a mapping from the set P+ (positive closure of P) of the nonempty strings over P, to V. For a given p belonging to P define a p-scenario as a mapping from the subset of P+, consisting of strings beginning with p, to V.

JAWAHARLAL NEHRU UNIVERSITY LIBRARY

TH-3923

SCHEDULE : A schedule from C is a finite / infinite sequence of events that can be applied, in turn, starting from C. Thus a schedule is a sequence of atomic steps .

If the execution of a schedule t from a configuration C results in a configuration D, we write

$$C \overset{t}{|\text{----}} D$$

If there exists a schedule t such that $C \overset{t}{|\text{---}} D$, we can also write $C\ |\text{---}\ D$;

SCHEDULER : is an agent that will determine the next atomic step in the execution.

Probability measure on the space of all possible schedules is provided by Probabilistic assumptions on the behaviour of the scheduler.

SERIAL HISTORY: An history H is said to be serial if :

(1) H is in U,

(2) for each positive integer k, k <= t, the number of processors exhibiting, faulty behaviour in Hk does not exceed k, and

(3) no processor fails after round t.

"o- IS CONSISTENT WITH N" : for a given choice N a subset of P of nonfaulty processors and a given scenario o-, say that o- is consistent with N if for each q belonging to N, p belonging to P, and w a set of all string over P, o-(pqw) = o-(qw). In other words, o- is consistent with N if each processor in N always reports what it knowns or hears truth fully.

SILENCING: Given a processor p in a history H in U, the silencing of p at rounf k of H is the unique history H' (not necessarily in U) such that

(1) H'k = Hk except that p sends no messages in round k of H'.

(2) no processor (except possibly P) fails after round k.

(3) P sends no messages after round k.

SIMULTANEOUS AGREEMENT: We say that the agreement is simultaneous if

(3) all correct processors give their outputs at the same round.

STEP : A step takes one configuration to another and consists of a primitive step by a single process P. Let C be a configuration. The step occurs in two phases as given below :

(1) receive(p) is performed on the message buffer in C to obtain a value m from M U {null}.

(2) depending on p's internal state in C and on m, p enters a new internal state and sends a finite set of messages to other processes.

SUBCONFIGURATION : Let C be a configuration and S be a subset of processes. A subconfiguration Cs is the restriction of C to the members of S.

Fs^i to denote any subconfiguration where all the correct processes in S have decided i.

SUBPATTERN : We call one pattern a subpattern of another if the corresponding history for the first pattern has as faulty processors only a subset of that of the second and the first pattern is the corresponding restriction of the second.

If history H in U has a pattern of failures that is a subpattern of that of a serial history, then H is also a serial history.

T-COMPUTATIONS: A sequence of configurations (called rounds), C0, C1, C2..., is a T-computation provided there exist messages sent by each process at each round such that:

(a) C0 is an initial configuration;

(b) for every i, and every p belongs to T, the messages sent out by p after Ci are exactly those specified by p's message generation fuction, applied to p's state in Ci;

(c) for every i, and every p belongs to T, p's state in Ci and the message sent to p after Ci.

In a T-computation, processes in T are nonfaulty whereas processes not in T may be faulty.

T-computation of an asynchronous approximation algorithm is one in which the processes in T always follows the algorithm, all processes (faulty and nonfaulty) continue to take steps until they reach a halting state, and any process that fails to enter a halting state eventually receives all messages sent to it.

WEAK INTERACTIVE CONSISTENCY PROBLEM: Each process i chooses a private value wi. The process must be then communicate among themselves to allow each process to compute a public value, such that:

(1) If all processes are nonfaulty and all the wi have same value then every process computes this value as its public value.

(2) Any two nonfaulty processes compute the same public value.

This is equivalent to the WBG problem.

WEAKLY GLOBAL COIN: A coin is called weakly global if there exists an absolute constant c > 0, such that for all v BELONGING TO { 0, 1 }, the probability that at least min { $\lfloor n/2 \rfloor$ + t + 1, n } processors all see outcome v is at least c.

WITNESS EQUIVALENCE: For k round initial sequences, this is the transitive closure of the relation that holds between Hk and J when for some processor P correct in both. pHk = pJk. Histories H and J witness equivalent if their k round initial sequences are witness equivalent for every k.

In other words, witness equivalence (through round k) is the transitive closure of the relation that holds between two histories if there is a processor correct in both that cannot distinguish between the two (through round k).

CHAPTER THREE

THEOREMS, LEMMAS, AND THEIR PROOFS

THEOREM 1.1

If |V| >= 2 and n>=3m, there exists no {Fp|p belonging to P} that assures interactive consistency for m faults.

PROOF.

Suppose, to the contratry, that {Fp|p * p} assures interactive consistency for m faults. Since n<=3m, P can be partioned into three nonempty sets A,B, and C, each of which has no more than m members. Let v and v' be two distinct values in V. The general plan is to construct three scenarios o< , $, and o- such that o< is consistent with N=A U C, $ with N= B U C and o- with N = A U B. The members of C will all be given private value v in o< and v' in $. Moreover, o<, $ and o- will be constructed in such a way that no distinguish o< from o- (i.e. o<a = o-a), and no processor b belongs to B can distinguish $ from o- (i.e. $b = o-b). It will then follow that for the scenario o- processors in A and B will compte different values for the members of C.

We define the scenarios o< , $ and o- recursively as follows:

i) For each w an elemnt of positive closure of P, not ending in member of C, let o<(w) = $(w) = o-(w) = v.

ii) For each a belonging to A, b belonging to B, c belonging to C

Let o<(c) = o<(ac) = o<(bc) = o<(cc) = v ,

$ (c) = $ (ac) = $ (bc) = $ (cc) = v',

o-(c) = o-(ac) = o-(cc) = v, o-(bc)=v'.

iii) For each a belonging to A, b belonging to B, c belonging to C , p belonging to P, w is any string over P ending in c.

Let o<(paw) = o<(aw), $ (paw) = o<(aw),

o<(pbw) = $ (bw), $ (pbw) = $ (bw),

o<(pcw) = o<(cw), $ (pcw) = $ (cw),

o-(paw) = o-(aw),

o-(pbw) = o-(bw),

o-(acw) = o<(cw),

o-(bcw) = $ (cw).

It is easy to verify by inspection that o< , $ and o- are in fact consistent with N= A U C, B U C, A U B, respectively. Moreover, one can show by a simple induction proof on the length of w that o<(aw) = o-(aw), $ (bw) = o-(bw)

for all a belonging to A, b belonging to B, w is any string over P

It then follows from the definition of interactive consistency that for any a belonging to A, b belonging to B, c belonging to C

v = o<(c) = Fa(o<a,c) = Fa(o-a,C) = Fb($`b,c) = v',

giving a contradiction.

Q.E.D.

LEMMA 2.1 i.

For any finite WBG alogrithm B there is a nonnegative integer k such that for any scenarios % and @ and all i bewlonging to P:

If the restrictions of %i and @i to $IIi^{(k)}$ are equal, then Bi(%i) = Bi(@i) .

PROOF:

Define an r-level finite scenario to be a mapping from $II^{(r)}$ to V. For any fixed i, we define a tree structure on the set of all such finite scenarios by lettilng an r-level scenario % be an ancestor of an r'-level senario %' if r < r' and %i and * is the restriction of %'i to $IIi^{(r)}$. Consider the subtree consisting of r-level scenarios %, for all r, such that there exist (infinite) scenarios @ and A whose restrictions to $IIi^{(r)}$ equal %i, and for which Bi(@i) does not equal Bi(Ai). If this subtree were infinte then by Konig's lemma it would have an infinte path. Such an infinte path defines an infinte scenario % which contradicts the definition of finiteness. Hence, this subtree must be finite, which implies the existence of a ki such that for any scenarios % and @ : if the restrictions of % and @ to $IIi^{(ki)}$ are equal then Bi(%i)=Bi(@i).

To complete the proof, we let k equal sup{ ki : i belongs to P} .

LEEMA 2.2 :

For any path 0,p1.....pk belonging to II : o-(0,p1,.........pk) mod 3 = pk.

PROOF:

This is simple consequence of the observation that

&(r,s) + &(s,t) is equivalent to (&(r,t)mod 3).

For any integer r, we let $\overline{r}$ denote r mod 3, which equals 0, 1, 2.

We now choose two particular elements of V, which we denote T and F. The following lemma asserts the existence of a sequence of scenarios $\%^{(r)}$ for integral values of r (including negative integers) which will form the basis for a proof by contradiction.

Only two values,denoted T and F, appear in $\%^{(r)}$. In this scenario, Processes $\overline{r+1}$ and $\overline{r+2}$ are nonfaulty, so they relay values correctly. The faulty Process $\overline{r}$ acts correctly excepts when relaying messages # for which o-(#) = $\overline{r}$, in which case it sends the value T to process $\overline{r + 1}$ and the value F to Processor $\overline{r - 1}$ = $\overline{r + 2}$.

LEMMA 2.3:

For any values T and F in V, and any integer r there is a scenario $\%^{(r)}$ such that for i = 1, 2 :

1) Process $\overline{r + i}$ is nonfaulty in $\%^{(r)}$.

2) For any # belonging to $II_{\overline{r+i}}$:

$$\%^{(r)}(\#) = \begin{cases} F & \text{if } o\text{-}(\#) >= r + i . \\ T & \text{if } o\text{-}(\#) < r + i . \end{cases}$$

PROOF:

By Leema 2, condition 2 defines $\%^{(r)}_{\overline{r+i}}$ for i = 1, 2. Since there are no requirements on $\%^{(r)}_{\overline{r}}$, and Process $\overline{r}$ is allowed to be faulty , we need only show that Condition 2 is achievable when Processor $\overline{r + 1}$ and $\overline{r +2}$ correctly relay messages to one another. However, this follows easily from the observation that if # belongs to $II_{\overline{r+i}}$, then $o\text{-}(\#, \overline{r + i \pm 1}) = o\text{-}(\#) \pm 1$.

LEMMA 2.4 :

For any integer r : if $\%^{(r)}$ is as in Lemma 2.3, then $\%^{(r)}_{r+2} = \%^{(r+1)}_{r+2}$

LEMMA 2.5:

If there are at least two distinct elements in V, then there does not exist a 1-fault WBG algorithm for n=3.

PROOF:

Let B be such an algorithm, and let T and F be distinct elements of V.

Let $\%^{T}$ and $\%^{F}$ be the scenarios defined by $\%^{T}(\#) = T$ and $\%^{F}(\#) = F$ for all # belongs to II. It follows from condition 1 of the definition of a WBG algorithm that $B_i(\%^{T}_i) = T$ and $Bi(\%^{F}_i) = F$ for all i. For each integer r, let $\%^{(r)}$ be the scenario whose existence was proved in Lemma 3.

Let k be the nonnegative integer whose existence is guaranteed by Lemma 1, with $\%^{T}$ substitued for %. Since o-(#) is less than or equal to the length of #, for any # in $II^{(k)}_{k+1}$, we have o-(#) <= K

$< K + 1$, so $\%^{(k)}(\#) = T$. Hence the restrictions of the scenarios $\%^{T}_{k+1}$ and $\%^{(k)}_{k+1}$ to $II^{(k)}_{k+1}$ are equal, so we must have $B_{k+1}(\%^{(k)}_{k+1})$ equal to T. Similarly, choosing such a nonnegative integer k' for $\%^{F}$, since $-(o-(\#))$ is less than or equal to the length of #, for any # in $II^{(k')}_{-k'}$, we have $o-(\#) >= -k' = (-k' - 1) + 1$, so $\%^{(-k'-1)}_{-k'}(\#) = F$. Hence the restrictions of $\%^{F}_{-k'}$ and $\%^{(k')}_{-k'}$ are equal, so $B_{-k'}\ \%((-k'-1)/(-k')) = F$.

It follows from Lemma 2.4 that for any $r: B_{r+2}(\%^{(r)}_{r+2}) = B_{r+2}(\%^{(r+1)}_{r+2})$.

Since $r + 1$ and $r + 2$ are nonfaulty in $\%^{(r)}$, it follows from condition 2 of the definition of WBG algorithm that $B_{r+1}(\%^{(r)}_{r+1}) = B_{r+2}(\%^{(r)}_{r+2})$. Hence, for each $r: B_{r+1}(\%^{(r)}_{r+1}) = B_{r+2}(\%^{(r+1)}_{r+2})$.

A simple induction argument shows that $B_{k+1}(\%^{(k)}_{k+1}) = B_{-k'}(\%^{(-k'-1)}_{-k'})$.

However, we saw above that $B_{k+1}(\%^{(k)}_{k+1}) = T$ and $B_{-k'}(\%^{(-k'-1)}_{-k'}) = F$.

Since T and F are distinct elements this provides the required contradiction.

THEOREM 2.1:

If n>2 and V contains at least two distinct elements then there exists an m-fault WBG algorithm if and only n > 3m.

PROOF:

The "if" part follows from the existence of algorithm to solve the original Byzantine Generals Problems demonstrated in [2_1] and [1_0]. To prove the "only if" part we assume the existence of such an algorithm and derive a contradiction.

Assume B is an m-fault WBG algorithm with 3 <= n <= 3m. We will use it to construct a 1-fault WBG algorithm for three processes, thereby contradicting Lemma 5. We first partition the (n-element) set P into three nonempty, disjoint sets P0, P1, P2 each containing at most m elements. (We can do this becasue 3<=n<=3m). Let 0 be an element of P0. We define the mapping ⅄ :P->{0',1',2'} by letting ⅄(p) = i' if and only if p belongs to Pi. We extend ⅄ to a mapping from P* into {0', 1', 2'}* in the obvious way by letting ⅄(p0, ... , pk) = ⅄(p0), ... , ⅄(pk) .

We also let 0", 1", 2" be element in P such that 0" = 0 and 1" belongs to P1 and 2" belongs tp P2 . Hence, ⅄(i") = i' .

We construct a 1-fault WBG algorithm B' for the set P'={0',1',2'} as follows. For any scenario %' on P', we define the scenario $\wedge$[%'] on P by $\wedge$[%'](#) = %'($\downarrow$(#)) .

The WBG algorithm B' is defined by $B'_{i'}(\%'_{i'}) = B_{i''}(\wedge[\%'_{i''}])$.

Observe that if i' is nonfaulty in %' , then every process in Pi (including i") is nonfaulty in $\wedge$[%'] .

To show that B' is a 1-fault WBG algorithm, we must verify the following conditions.

1) If all process in P'are nonfaulty in %', then for all i' belonging to P' : $B'_{i'}(\%'_{i'}) = \%'(0')$.

2) For any ii', j' belonging to P' : if i' and j' are nonfaulty in %', then $B'_{i'}(\%'_{i'}) = B'_{j'}(\%'_{j'})$.

To prove these conditions we use the observation that if Process i' is nonfaulty in %' , then every process in Pi is nonfaulty in $\wedge$[%']. Hence if all processes in P' are nonfaulty in %' then all processes in P are nonfaulty in $\wedge$[%']. Using condition 1 for the m-fault WBG algorithm B, we see that

$$\begin{aligned} B'_{i'}(\%'_{i'}) &= B_{i''}(\wedge[\%']_{i''}) \\ &= \wedge[\%'](0'') \\ &= \%'(0'), \end{aligned}$$

which proves condition 1 for B'.

Next assume that the i' and j' are nonfaulty in %'. Since i" and j" are nonfaulty in $\wedge$[%'], condition 2 for B yields

$$B'_{i'}(\%'_{i'}) = B_{i''}(\wedge[\%']_{i''})$$

$$= B_{j''}(\wedge[\%']j'')$$

$$= B'(\%'_{j''\,j'})$$

This proves condition 2 for B'. We have thus constructed a 1-fault WBG algorithm for the three process 0', 1', 2', contradicting Lemma 2.5.

Q.E.D.

__THEOREM 2.2:__

If |v| < D, for all v * V, then the algorithm AG(k) satisfies the following properties.

1) If all processes are nonfaulty then vi = v for every i.

2) If Processes i and j are nonfaulty then | vi - vj | < 2D/k

__PROOF:__

Note that no limit is placed upon the number of faulty process. The proof of this theorem uses the following lemma i.e. lemma 2.6

To prove the first property we simply observe that if all process are nonfaulty then they correctly relay values, so all the $v_i^{(r)}$ equal v. To prove the second property we note that if Process i and j are nonfaulty, then they correctly relay the values of $v_i^{(r)}$ and $v_j^{(r)}$ to one another in round r+1. It therfore follows that for each $r \geq 1$:

$$v_i^{(r)} \leq v_j^{(r+1)} , \qquad v_j^{(r)} \leq v_i^{(r+1)} .$$

The second property then follows immediately from the above lemma substituting $v_i^{(r)}$ for s_r and $v_j^{(r)}$ for t_r .

LEMMA 2.6 :

Assume that $|v| < D$ for all v belonging to V. If s_r , t_r are elements of V such that:

$$s_r \leq t_{r+1} , \qquad t_r \leq s_{r+1}$$

for all r with $1 \leq r < k$ then

$$\left| \sum_{r=1}^{k} s_r - \sum_{r=1}^{k} t_r \right| < 2D.$$

PROOF:

It follows from the first inequality of the, hypothesis that:

$$\sum_{r=1}^{k} s_r <= s_k + \sum_{r=2}^{k} t_r$$

From this we deduce that

$$\sum_{r=1}^{k} s_r - \sum_{r=1}^{k} t_r <= s_k - t_1 <= 2D .$$

The symmetric argument, interchanging s and t yields

$$\sum_{r=1}^{k} t_r - \sum_{r=1}^{k} s_r <= 2D .$$

and combining the two inequalitites proves the lemma.

Q. E. D.

THEOREM 2.3:

If V is a bounded set of numbers then $AG^{(\infty)}$ is an infinte m-fault WBG algorithm for any m.

PROOF:

The proof is quite simple and rests upon the observation that if i and j are nonfaulty then

$$v_i^{(r)} <= v_j^{(r+1)} , \quad v_j^{(r)} <= v_i^{(r+1)} .$$

for all r>0 which in turn implies that sup{ $v_i^{(r)}$ } = sup{ $v_j^{(r)}$ }.

LEMMA 3.1:

Suppose that from some configuration C, the schedules o-1 and o-2 lead to configurations C1, C2, respectively. If the sets of processes taking steps in o-1 and o-2, respectively, are disjoint, then o-2 can be applied to C1 and o-1 can be applied to C2, and both lead to the same configuration C3. (See Figure 3.3.1.)

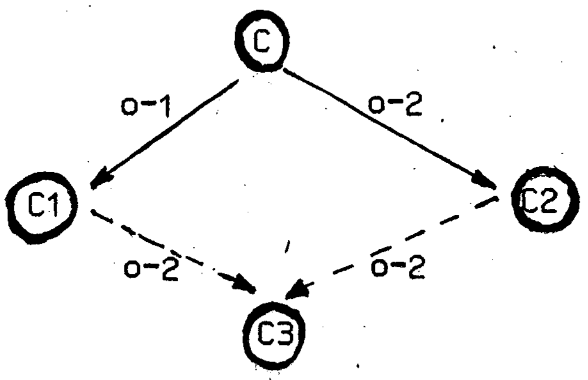

Figure 3.3.1

PROOF:

The result follows at once from the system definition, since o-1 and o-2, do not interact.

THEOREM 3.1:
No consensus protocol is totally correct inspite of one fault.

PROOF:
Assume to the contrary that P is a consensus protocol that is totally correct inspite of one fault. We prove a sequence of lemmas which eventually lead to a contradiction.

The basic idea is to show circumstances under which the protocol remain forever indecisive. This involves two steps.

First, we argue that there is some initial configuration in which the decision is not already predetermined. Second, we construct an admissible run that avoids ever taking a step that would commit the system to a particilar decision.

Let C be a configuration and let V be the set of decision values of configurations reachable from C. C is bivalent if |V| = 1, let us say 0-valent or 1-valent according to the corresponding decision value. By the total correctness of P, and the fact that there are always admissible runs, V is not a null set .

LEMMA 3.2:

P has a bivalent initial configuration.

PROOF:

Assume not. Then P must have both 0-valent and 1-valent initial configurations by the assumed partial correctness. Let us call two initial configurations adjacent if they differ only in the initial value Xp of a single process P. Any two initial configurations are joined by a chain of initial configurations, each adjacent to the next. Hence, there must exist a 0-valent initial cofiguration C0 adjacent to a 1-valent initial configuration C1. Let p be the process in whose initial value they differ.

Now consider some admissible deciding run from C0 in which process p takes no steps, and let σ be the associated schedule. Then σ can be applied to C1 also, and corresponding configurations in two runs are identical except for the internal state of process P. It is easily shown that both runs eventually reach the same decision value. If the value is 1, then C0 is bivalent; otherwise, C1 is bivalent. Either case contradicts the assumed nonexistence of a bivalent initial conffiguration.

LEMMA 3.3:

let C be a bivalent configuration of P, and let e = (p,m) be an event that is applicable to C. Let Q be the set of configurations reachable from C without applying e, and let D = e(Q) = {e(E) | E belongs to Q and e is applicable to E}. Then, D contains a bivalent configuration.

PROOF:

Since e is applicable to C, then by defination of Q and the fact that message can be delayed arbitrarily, e is applicable to every E belonging to Q.

Now assume that D contains no bivalent configurations, so every configuration B belonging to D is univalent. We proceed to derive a contradiction.

Let Ei, be an i-valent configuration reachable from C, i = 0, 1 . (Ei exists since C is bivalent.) If Ei belongs to D, let Fi= e(Ei). Otherwise, e was applied in reaching Ei, and so there exists Fi belonging to D from which Ei is reachable. In either case, Fi is i-valent since Fi is not bivalent (since Fi belongs to D and D contains no bivalent configurations) and one of Ei and Fi is reachable from the other. Since Fi belongs to D, i = 0, 1, D contains both 0-valent aand 1-valent configurations.

Call two configurations neighbours if one results from the other in a single step. By an easy induction, there exist neighbours C0, C1 belongs to D such that Bi = e(Ci) is i-valent, i =0, 1. Without loss of generality, C1 = e'(C0) where e'=(p', m').

<u>Case 1:</u> If p' is not equal to p, then D1 = e'(D0) by Lemma 3.1 . This is impossible, since any successor of a 0-valent configuration is 0 - valent.

<u>Case 2:</u> If p' = p, then consider any finite deciding run from C0 in which P takes n steps.

Let o- be the corresponding schedule, and let A = o-(C0). By Lemma 3.1, o- is applicable to Di, and it leads to an i-valent cofiguration Ei = o-(Di), i =0, 1. Also by Lemma 1, e(A) = E0 and e(e'(A)) = E1. Hence, A is bivalent. But this is impossible since the run to A is deciding (by assumption), so A must be univalent.

In each case, we reached a contradiction, so D contains a bivalent configuration.

THEOREM 3.2:

There is a partially correct consensus protocol in which all nonfaulty processes always reach a decision, provided no processes die during its execution and a strict majority of the processes die during its execution and a strict majority of the processes are alive initially.

LEMMA 4.1:

With a k-resilent consensus protocol, for any reachable configuration C, and for any subset S of process that contains at least n-k correct processes, either Cs $\vdash \overset{0}{F}$s or Cs $\vdash \overset{1}{F}$s.

PROOF:

Let C be a reachable configuration; S be a subset of process that contains at least n-k correct processes, and $\bar{S}$ be the complement of S (i.e.the set of processes, that are not in S). Note that $|\bar{S}|$ <=k. Assume first that all the process in $\bar{S}$ are fail-stop. Suppose that, after reaching configuration C, all the processes die without sending warning messages. This results in a configuration C'. We have C's=Cs. From the consistency and the convergence properties of the k-resilent protocol, we must have C's $\vdash \overset{0}{F}$s or C's$\vdash \overset{1}{F}$s. Since C's = Cs, and since the death of process in $\bar{S}$ cannot be detected, we have Cs $\vdash \overset{0}{F}$s or Cs $\vdash \overset{1}{F}$s. This must also hold even if ther are correct processes in $\bar{S}$.

LEMMA 4.2 [4 3]:

For k>=1, any k-resilent consensus protocol has a bivalent initial configuration.

PROOF:

Suppose all the processs are correct. Initial configurations differ only by the processes input values. Two initial configuration differeing by the input a value of only one process are adjacent. Assume, for contradiction there is a k-resilent protocol such that any initial confilguration is either 0-valent or 1-valent. By the bivalence property of the protocol there must be one of each. Therefore there must be two adjacent initial configuration, I^0 and I^1 , that are 0-valent and 1-valent, respectively. These configurations differ only by the input value of some process p. Therefore, Is=Is, where S includes all the processes except p. From Lemma 4.1, either $Is^0 \vdash Fs^0$ or $Is^0 \vdash Fs^1$. If $Is^0 \vdash Fs^0$, then $Is^1 \vdash Fs^0$, and therefore we have $I1 \vdash F0$. But I1 is 1-valent a contradiction. A similar contradiction is obtained if we assume $Is^0 \vdash Fs^1$.

THEOREM 4.1:

There is no $\lceil n/2 \rceil$ -resilent consensus protocol for the fail stop case.

PROOF:

Assume there is such a protocol and consider a system in which all the processes are correct. Let C be any reachable configuration, S be any subset of processes of size $\lfloor n/2 \rfloor$ and $\bar{S}$ be the complement of S. We claim that C_S and $C_{\bar{S}}$ are either both 0-valent or both 1-valent.

From Lemma 4.1 , since the protocol is $\lceil n/2 \rceil$ - resilent and $|S|, |\bar{S}| >= n - \lceil n/2 \rceil$, we have $C_S \vdash F_S^i$ and $C_{\bar{S}} \vdash F_{\bar{S}}^j$ for some decision values i and j. Suppose that there exists two schedules σ-0 and σ-1 such that $C_S \xrightarrow{\sigma\text{-}0} F_S^0$ and $C_{\bar{S}} \xrightarrow{\sigma\text{-}1} F_{\bar{S}}^1$ (or viceversa). Then we can apply the schedule $\sigma = \sigma\text{-}0 \cdot \sigma\text{-}1$ to configuration C, and this result in a confilguration where processes in S decide 0 and processes in $\bar{S}$ decide 1 (or vice versa). This contradicts the consitency of the protocol and the claim is proved.

By Lemma 4.2, there is a bivalent initial configuration I. From the claim without loss of generality both I_S and $I_{\bar{S}}$ are 1-valent.

Let σ be a schedule such that $I \xrightarrow{\sigma} F^0$. We denote by I^t the configuration reached from I after the first t steps in σ.

Note that $I^o = I$ and $I^{|o-|} = F^o$. Clearly both $Is^{|o-|}$ and $\overline{Is}^{|o-|}$ are 0-valent. Let t be the smallest index such that both Is^t and $\overline{Is}^t$ are 0-valent. Note that t>0. From the initial claim, and the minimality of t, both Is^{t-1} and $\overline{Is}^{t-1}$ must be 1-valent.

Let p be the process that performs the atomic step s such that $I^{t-1} \overset{s}{|\!-\!-\!-} I^t$. Suppose p belongs to S, and therfore $Is^{t-1} \vdash Is^t$. Since Is^t is 0-valent we have $Is^t \vdash Fs^o$. Then we must have $Is^{t-1} \vdash Fs^o$. But Is^{t-1} is 1-valent and this is a contradiction. We obtain a similar contradiction if we assume that p belongs to $\overline{S}$.

THEOREM 4.2:

For any k, $0 <= k <= \lfloor (n-1)/2 \rfloor$, the protocol 4_1 is a k-resilient consensus protocol for the fail stop case.

PROOF:

We need the following few definition. Each execution of the protocol outerloop is called a phase. A process is in phase t if at the begining of this phase its variable phaseno has the value t. A message (witness for i) whose phaseno field is equal to t is called a t-message (t-message for i). A process p decide in phase t if it sets the decision variable <u>dp</u> while its phaseno variable is equal to t. The value of the variable <u>VAR</u> of process p, when p is at the begining of phase t, is denoted by $\underline{VARi}^{t}$.

We prove the therorem by showing the protocol's constitency, deadlock-freedom, convergence, and bivalence, in the presence of up to k faulty processors.

<u>Consistency:</u>

Let t be the smallest phase in which a process decides. We claim that, for any processes p and q we cannot have both $withness\_count(0)p^{t} > 0$ and $witness\_count(1)q^{t} > 0$. Suppose for some i, $witness\_count(i)p^{t} > 0$. Then process p, in phase t-1, must have received from some process r a (t-1)-witness for i. So r must have received in phase t-2 more than n/2 (t-2)-message with value i. Therefore if both $witness\_count(0)p^{t} > 0$ and $witness\_count(1)q^{t} > 0$, since there are at most n processors, there must be a least one processor that sent (t-2)-messages with both values. This is impossible in the protocol 4_1,and the claim is proved. From this claim and the description of the protocol, it is now easy to check that a process can never have both witness_count(0) and witness_count(1) greater than 0 in the same phase.

Let t be the smallest phase in which a process decides, let us say process p decides 0 in phase t. We prove that no other process q can decide 1.

Since p decides 0 in phase t, we have $witness\_count(0)_p^t > k$. From the claim, we cannot have $witness\_count(1)_q^t > k$. Therefore if q decides in phase t, it also decides 0.

We now show that all the t-messages sent are of the form (t, 0, cardinallity). Sice $withness\_count(0)_p^t > k$ process p recives more than k (t - 1)-witness for 0. Consider a process r that sends a t message. Process r must have received n-k (t-1)-messages, and one of them must be a (t-1)-witness for 0. Then process r increments witness_count(0) in phase t-1. From the initial claim, process r sets its value to 0 in phase t-1, and it sends (t, 0, cardinality) messages in phase t.

Consider a process q that decides in phase t+1. From the above remark, all the t-messages received by q have value 0, and therfore q must decide 0.

We now prove that all the (t+1) messages sent are of the form (t+1, 0, n-k). Consider a process r that sends (t+1)-messages. From the description of the protocol 4_1, we see that if r decides in phase t, the (t+1) messages it sends are of the form (t+1, 0, n-k). If r does not decide in phase t, it must have received n-k. Thefore it sends (t+1, 0, n-k). If r does not decide in phase t, it must have received n-k t-message in phase t, the (t+1)-messags it sends are of the form (t+1, 0, n-k). If r does not decide in phase t, it must have received n-k t-message in phase t. We already proved that all the t-messages have value 0. So, in phase t, process r sets its value to 0 and its cardinality to n-k. Therfore, it sends (t+1, 0, n-k) messages in phase t+1.

A process r that reaches phase t+2 must have received n-k (t+1)-messages. From the remark above all the (t+1) messages are witnesses for 0 and therfore r decides 0 in phase t+2.

Since any process that reaches phase t+2 decides 0, no process can ever be in a phase higher than t+2 and no process can decide 1.

<u>Deadlock-freedom:</u>

Since processes wait for each other's messages, the protocol might be exposed to deadlocks. We prove that the protocol is deadlock free.

Suppose for contradiction the protocol runs into a deadlock. Let D be the set of deadlocked processed. Each process q in D is deadlocked in phase tq. Let $t0 = \min_{(q \text{ belongs to } D)} t_q$, and p belongs to D be a process that is deadlocked in phase t0. Let S be a set of n-k correct processes. There are two possible cases.

1) No process in a phase t, t <= t0 - 2. By the minimality of t0, every process in S either decides in phase t0 - 1 or t0, or it reaches phase to without deciding in either case it send t0 - messages to all the processes. Therfore there will be at least n - k t0-messages in p's buffer and p cannot be deadlocked in phase t0; this is a contradiction.

2) Some process decides in phase t, t <= t0 - 2. Let t be the smallest phase in which a process decides. In the proof of the protocol consistency, we shwd that no process can ever be in a phase greater than t+2 decides. Note that p is deadlocked in phase t0 >= t+2. This is a contradiction, and the proof of deadlock freedom is complete.

<u>Convergence:</u>

Let S be a set of n-k correct processes. Suppose no process in S decides in a phase t, t<to. We prove that there is a fixed Theta such that, with probabililty greater than Theta, all the processes in S decide in phase t0 + 2.

Since there are no deadlocks, every process in S will reach phase t0. Note that for t = t0 , t0 + 1, and t0 + 2, from the assumption of fair scheduler there is a positive constant Rho such that with probability greater than p, every process in S receive in phase t. In other words, with probability greater than Theta = $(\text{Rho})^3$ for three consecutive phase all the processes in S exchange messages exclusively among themselves obviously to the rest of the system. It is clear from the protocol that, if this happens, then all the processes in S decide in phase t0 + 2.

Bivalence:

If all the processes start with the same input value, all the correct processes decide that value within two steps.

LEMMA 4.3:

With a k-resilent consensus protocol, for any reachable configuration C, and for any subset S of proesses that contain at least n-k correct processes, either $Cs \mid\!- Fs^{0}$ or $Cs \mid\!- Fs^{1}$ by some legal schedule.

PROOF:

The malicious processes can behave just like fail-stop processes and die. The proof follows from this observation and the proof of Lemma 4.1.

THEOREM 4.3:

There is no $\lceil n/3 \rceil$ resilent consensus protocol for the malicious case.

PROOF:

Suppose there is a $\lceil n/3 \rceil$ - resilent protocol. Let S and T be subsets of processes of size $\lfloor 2n/3 \rfloor$ such that $|T \cup S| = n$.

Note that |T intersection S| <= n/3. Let C be a legally reachable configuration. All the mallicious processes have followed the protocol so far. If they contaniue to follow the protocol then there is no way in which they differ from correct processes.

Therfore by Lemma 4.3, $Cs \vdash Fs^{i}$ and $Ct \vdash Ft^{i}$, for some decision values i and j.

We claim that Cs and Ct are either both 0-valent or both 1-Valent. Suppose not then without loss of generality there are legal schedules o-0 and o-1 such that $Cs \xrightarrow{o\text{-}0} Fs^{0}$ and $Ct \xrightarrow{o\text{-}1} Ft^{1}$. Suppose that all the processes in T intersection S are malicious. The following schedule is possible.

From C by schedule o-0 we first reach a configuration where all the correct processes in S decide 0. Then the malicious processes in S intersection T change there states and their buffers' content back to what they were in C, resulting in some configuration C'.

The only difference between Ct' and Ct is that in Ct' the buffers of the process in T may have additional messages (that were added during the execution of $\sigma 0$). Since $Ct \overset{\sigma 1}{\vdash\!\!-\!\!-\!\!-} Ft^{1}$, the processes in T can now follow the legal schedule $\sigma 1$ from configuration C', until all the correct processes in T decide 1. Then shcedule violates the consistency of the protocol and the claim is proven.

The rest of the proof follows closely the last part of the proof of Theorem 4.1. Let I be the bivalent initial configuration guranteed by Lemma 4.2. From the claim, without loss of generallity, both Is and It are 1-valent. Let σ be a legal schedule such that $I \overset{\sigma}{\vdash\!\!-\!\!-} F^{0}$. We denote by I^{t} the configuration reached from I after the first t step in σ. Clearly both $Is^{|\sigma|}$ and $It^{|\sigma|}$ are 0-valent. Let t be the smallest index such that both Is^{t} and It^{t} are 0-valent. Note that $t > 0$. From the initial claim and the minimality of t, both Is^{t-1} and It^{t-1} must be 1-valent.

Let p be the process that performs the atomic step s such that $I^{t-1} \overset{s}{\vdash\!\!-\!\!-} I^{t}$. Assume that p belongs to S. We have $Is^{t} \vdash Fs^{0}$, and therefore $Is^{t-1} \vdash\!\!- Fs^{0}$. However Is^{t-1} is 1-valent and this is a contradiction. We obtain a similar contradiction if we assume that p belongs to T.

THEOREM 4.4

For any k, $0 <= k <= \lfloor (n - 1)/3 \rfloor$, the protocol 4_2 is a k-resilent consensus protocol for the malicious case.

PROOF:

We show the protocol's dèadlock-freedom, consistency, convergence and bivalence, in the presence of upto k faulty processes. We use the same notation and definitions as in the proof of Theorem 4.2.

Deadlock-freedom:

We have to prove that it is always possible for a process to accept n-k messages. Consider a correct process p in phase t, where t is the smallest phase among correct processes in the system. At least n-k correct processes are in phase t or in a higher phase. Let q be such a process. Process q has already sent a (initial, q, v, t) message to all the other processes. Since there are at least n-k correct processes, p's buffer will receive at least n -k (echo, q, v, t) messages. Since n-k > (n+2)/2, then p at phase t, eventually accepts this message with value v from q. Therefore p accept n-k messages from correct processes and p proceeds to the next phase.

Consistency:

Consider any two processes p and q at some phase t. We claim that, if p and q accept a message from some processor, then these messages must must have the same value. Suppose not then at phase t, p accepts a message with value 0 from r and q accepts a message with value 1 from r. Then more than (n+k)/2 process sent (echo, r, 0, t) messages to p, and more than (n+k)/2 processes echoed (echo, r, 0, t) and (echo, r, 1, t). Since there are at most k malicious processes then at least one correct process has sent both (echo, r, 0, t) and (echo, r, 1, t). From the description of the protocol, correct processes cannot do that and hence a contadiction.

Let t be the smallest phase in which a correct process decides. Let us us say process p decides 0 in phase t. Process p must have accepted message wait value 0 from a set S of more than (n+2)/2 processes. By deadlock-freedom, any other correct process q will accept at phase t, messages from n-k processes. Therfore it must accept messages from more than $(n + k) / 2 - k = (n - k) / 2$ processes in S. By the claim the value of the messages accepted by q from processes in S must be 0. So q accepts more than $(n - k) / 2$ messages with value 0, and it changes its value to 0.

At phase t+1, all the correct processes will have 0. Note that it takes at least $(n - k) / 2$ messages wait value 1 to change the value of a correct process to 1.

Since there are only $k < n/3$ malicious processes and $k < (n-k)/2$, this can never happen. Therfore from phase t on, all the correct processes will have value 0 and they can not decide 1.

Convergence:

Let S be a set of correct processes that have not decided yet. Suppose no process in S decides in a phase t, t < t0. We prove that there is a fixed Theta such that, with probability greater than theta, all the processes in S decide in phase t0 + 1.

Since there are no deadlocks, every process in S reaches phase t0. From the assumptions on the system behaviour there is such that in phase t0 and t0 +1 the following happens with probability greater than Theta. At phase t0, every process in S accepts messages from the same set of n-k processes. At phase t0 + 1, every process in S accepts messages only from correct processes.

It is clear from the protocol that all the processes in S decide in phase t0 + 2.

Bivalence:

If all the processes start with the same input value within two phases all the correct processes decide that value.

THEOREM 4.5:

It is impossible to achieve asynchronous Byzantine Agreement with k >= n/3 .

PROOF:

Suppose it is possible; since K>= n/3, we can partition the processes to three disjoint sets, A, B and C, of size k 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 malicious, and they do not send any messages during the protocol. Since the transmitter is correct, the processes in A and B will agree on 0 within some time t.

(2) Only mthe tranmitter is malicious. It sends 0-messages to processes in A and B, and 1-messages to processes in C. Also, messages from C are delayed for a period longer than t before they are received. The processes in A and B have the same view of the system as in scenario 1, and therefore can agree on 0 at time t.

In a similar fashion we can construct scenario 3 with the following properties:

(3) Only the transmitter is malicious. It sends 1-messages to A and C, and 0-messages to B. Messages from B are delayed for a period longer than t'. At time t' the processes in A and C agree on 1.

Now we can combine scenarios 2 and 3 to yield a contradiction:

4) The processes in A are malicious, the processes in B and C are correct. The processes in A send messages to processs in B as in scenario 2, and to processes in C as in scenario 3.

All messages between in B and processes in C are delayed for a period longer than max(t, t'). In this scenario at time max(t, t'), the processes in B will agree on 0 and the processes in C will agree on 1, a contradiction.

THEOREM 4.6:

The protocol 4.3 acheives Asynchronous Byzantine Agreement for $k = 1, \ldots, \lfloor n-1)/3 \rfloor$ malicious processes.

PROOF:

We have to show that if some correct process p decides some value then all the correct then they alll decide on the transmitter's value.

First we claim that no two correct processes p and q can send ready messages with different values. Suppose this is possible, then p received more than (n + k) / 2 (echo, 1) messages, or a (ready, 1) message from a correct process. Similarly, q received more than (n + k) / 2 (echo, 0) messages, or a (ready, 0) message from a correct process. In either case, some two correct processes, and s and t, received more than (n + k) / 2 (echo, 0) messages and more than (n + k) / 2 (echo, 1) messages, respectively. Therefore, some correct process r must have sent both (echo, 1) and (echo, 0) messages. But this is impossible for a correct process. Since decision require 2k + 1 ready messages with the same value, it is also clear that no two correct process can decide different values.

Suppose p decides i, then p received 2k+1 (ready, i) messages. At least k + 1 of them were sent by correct processes. Therfore, every correct process will also receive at least k + 1 (ready ,i) messages, and will send its (ready, i) message. Thus, at least n-k process will send(ready, i) messages. Therefore, every correct process will receive at least 2k+1 (ready, i) messages and will decide i.

It is clear that if the transmitter is correct, then all the correct processes will decide on its value.

LEEMA 5.1:

Suppose that V and W are nonempty multisets. Then

1) |V intersection W| - |s(v) intersection s(w)| <=1;

2) |V intersection W| - |l(v) intersection l(w)| <=1.

PROOF:

We prove the first inequality; the argument for the second is symmetric.

If V and W have the same minimum, then the same element is removed from each, and hence at most one element is removed from their intersection. If the minima of V and W are not the same, then same, then either the minmum of V is not in W, or the minimum of W is not in V. In either case, at most one element is removed from the intersection.

LEEMA 5.2:

Suppose that j is a nonnegative integer and that V and W are multisets such that |V| >= 2j and |W| >= 2j. Then

$$|V \text{ intersection } W| - |\text{ reduce}^{j}(V) \text{ intersection reduce}^{j}(W)| <= 2j$$

PROOF:

Follows from repeated application of Leema 5.1.

LEEMA 5.3:

Suppose that j is a nonnegative integer and that U and V are nonempty multisets such that |V - U| <=j and |V| > 2j. Then

$A(\text{reduce}^{j}(V))$ is a subset of A(U).

PROOF:

Suppose $A(\text{reduce}^{j}(V))$ is not a subset of A(U). Then either $\min(\text{reduce}^{j}(V)) < \min(U)$ or $\max(\text{reduce}^{j}(V)) > \max(U)$.

If $\min(\mathrm{reduce}^{j}(V)) < \min(U)$, then

$$\sum_{r<\min(u)} V(r) \geq j + 1 .$$

Hence $|V - U| \geq j+1$, which contradicts a hypothesis.

The case $\max(\mathrm{reduce}^{j}(V)) > \max(U)$ is symmetric.

LEMMA 5.4:

Suppose $K > 0$ and $t \geq 0$ are integers. Suppose that U and V are nonempty multisets such that $| V - U | \leq t$ and $|V| > 2t$. Then $f_{k,t}(V)$ belongs to $A(U)$.

PROOF:

Follows easily from Lemma 5.3 (with $j = t$).

LEMMA 5.5:

Suppose V, W and U are multisets, and $K > 0$, $t \geq 0$, and $m > 2t$ are integers, with $| V | = | W | = m$, $| V - U | \leq t$, $| W - U | \leq t$, and $| W - V | = | V - W | \leq k$. Then

$$\left| f_{k,t}(V) - f_{k,t}(W) \right| \leq \frac{\&(U)}{c(m - 2t, k)}$$

PROOF:

Let M = reduce^t (V) and N = reduce^t (W). Since V and W each contain exactly m elements, M and N each contain exactly m - 2t elements, and hence select^k (M) and select^k (N) each contain exactly c = c (m - 2t, k) element. Let m0 <= m1 <= ..... <= mo-1 be the element of select^k (M), and let n0 <= n1 <= .... <= n_{c-1} be the elements of select^k (N). Notice that there are at least k_i + 1 elements in M that are less than or equal to m1, and at most ki elements in M that are strictly less than mi; similarly for N.

We begin by showing that max(mi, ni) <= min(mi+1, ni+1) for 0 <= i <= c - 2. It suffices to show that mi <= ni + 1; a symmetric argument demonstrates that ni <= mi + 1.

We proceed by contradiction : Suppose that mi > ni + 1. As noted above, there are at least k(i+1) + 1 elements in N less than or equal to ni + 1. By the supposition these elements are strictly less than mi. Therefore, there are at least k (i + 1) + 1 - ki (= k + 1) elements in N that are not in M; thus | N - M | >= k +1. Now by hypothesis, | W - V | <= k, so |W intersection V| >= m -k. Then lemma 5.2 shows | N intersection M | >= m - k - 2t, and hence | N - M | <= (m - 2t) - (m - k - 2t) = k. This is a contradiction and we conclude that mi <= ni + 1.

Now we use the inequality shown above to obtain the desired result.

$$\left| f_{k,t}(V) - f_{k,t}(W) \right| = \left| \text{mean}(\text{select}_k (\text{mean}(\text{select}_k (N)) \right|$$

$$= \frac{1}{c} \left| \left(\sum_{i=0}^{c-1} m_i \right) - \left(\sum_{i=0}^{c-1} n_i \right) \right|$$

$$= \frac{1}{c} \left| \sum_{i=0}^{c-1} (m_i - n_i) \right|$$

$$<= \frac{1}{c} \sum_{i=0}^{c-1} \left| m_i - n_i \right| \quad \text{(by the triangle inequality)}$$

$$= \frac{1}{c} \sum_{i=0}^{c-1} (\max(m_i , n_i) - \min(m_i , n_i)).$$

By the inequality demonstrated above, for 0 <= i <= c - 2,

(max(mi, ni) - min(mi, ni))

<= (min(mi + 1) - min(mi, ni) - min(mi, ni))

so weget,

$$\left| f_{k,t}(V) - f_{k,t}(W) \right|$$

$$<= \frac{1}{c} [\max(m_{c-1}, n_{c-1}) - \min(m_{c-1}, n_{c-1})].$$

$$+ \frac{1}{c} \sum_{i=0}^{c-1} [\max(m_{i+1}, n_{i+1}) - \min(m_i, n_i)]$$

Collecting terms then shows that

$$\left| f_{k,t}(V) - f_{k,t}(W) \right|$$

$$<= \frac{1}{c} [\max(m_{c-1}, n_{c-1}) - \min(m_0, n_0)].$$

Now, A(M) is a subset of A(U) and A(N) is a subset of A(N) by Lemma 5.3 (with j = t), so max (m_{c-1} , n_{c-1}) <= max (U) and min (m_0 , n_0) >= min (U). Hence,

$$\left| f_{k,t}(V) - f_{k,t}(W) \right|$$

$$<= \frac{1}{c} [\max(U) - \min(U)].$$

$$= \frac{1}{c} \&(U) .$$

as desired.

THEOREM 5.1:

If $n >= 3t + 1$, then there exists a t-correct synchronous approximation algorithm with n processes.

PROOF:

Given in section 4.5.2 of chapter 4.

LEMMA 5.6:

Suppose $n, t > 0$ are such that $n >= 3t + 1$. Let T be a set of processes, with $| T | >= n - t$. Let h be a positive integer. Let U and U' be the multisets of values of processes in T immediately before and after round h, respectively, in a particular T-computation of S0. Then

1) $$\&(U') = \frac{\&(U)}{c(n - 2t, t)}$$

2) A(U') is a subset of A(U)

PROOF:

Let p and q be arbitrary processes in T. Let V and W be the multisets of values (including default values) received by p and q, respectively, at round h. Then $| V | = | W | = n$. Since there are at most t faulty processes, $| V - U | <= t$ and $| W - U | <= t$. Moreover, since V and W contain identical enteries for all the processes in T, we know that $| V - W | = | W - V | <= t$.

1) The multisets V, W, and U satisfy the hypotheses of Lemma 5.5 (with m = n and k = t). Thus,

$$\left| f_{t,t}(V) - f_{t,t}(W) \right| <= \frac{A(\ U\ )}{c(n\ -\ 2t,\ t)}$$

2) The multiset V and U satisfy the hypotheses of Lemma 5.4 . Thus ft.t(V) belongs to A(U). Since p and q were choosen arbitrarily, the result follows.

<u>**LEEMA 5.7:**</u>

Assume that n >= 3t + 1. Let T be a set of processes, with |T| >= n - t. Let h be a positive integer. Let U and U' be the multisets of values off processes in T, immediately before and after round h, respectively, in a particular T-computation of S. Then A(U') is a subset of A(U).

<u>**PROOF:**</u>

Let p be an arbitary process in T. Let v and v' be the values held by p immediately before and after round h, respectively. It suffices, since p is arbitary, to show that v' is an element of A(U). If p has terminated prior to the start of round h, then v' = v belongs to A(U). If p has not halted prior to the start of round h, then let V bet the multiset of values received by p in round h. Then V and U satisfy the hypotheses of Leema 5.4, and since v'=ft,t(V), it follows that v'belongs to A(U).

THEOREM 5.2:

If $n >= 5t + 1$, then there exists a t-correct asynchronous approximation algorithm with n processes.

PROOF:

Given in section 4.5.3 of chapter 4.

LEMMA 5.8:

Suppose n, t>0 are such that n >= 5t + 1. Let T be a set of processes, with | T | > n - t. Let h be a positive integer, Let U and U' be the multisets of values of processes in T, immediately before and after round h, respectively, in a particular T-computation of A0. Then

1)
$$\&(U') = \frac{\&(U)}{c(n - 3t, 2t)}$$

2) A(U') is a subset of A(U)

PROOF:

Let p and q be arbitrary processes in T. Let V and W be the multisets of values received by p and q, respectively, at round h. Then | V | = | W | = n - t. Since there are at most t faulty processes, | V - U | <= t and | W - U | <= t. Moreover, since V and W both contain identical entries for all the processes in T from which both p and q heard, we know that | V intersection W | >= -3t. Hence, | V-W | = | W - V | = | V | - | V intersection W | <= 2t.

1) The multisets V, W, and U satisfy the hypotheses of Lemma 5.5 (with m = n and k = 2t). Thus,

$$\left| f_{2t,t}(V) - f_{2t,t}(W) \right| <= \frac{A(U)}{c(n - 3t, 2t)}$$

2) The multiset V and U satisfy the hypotheses of Lemma 5.4 . Thus f2t,t(V) belongs to A(U). Since p and q were choosen arbitrarily, the result follows.

LEEMA 5.9:

Assume that n >= 5t + 1. Let T be a set of processes, with |T| >= n - t. Let h be a positive integer. Let U and U' be the multisets of values of processes in T immediately before and after round h, respectively, in a particular T-computation of A. Then A(U') is a subset of A(U).

THEOREM 6.1:

The function COIN_TOSS produces a weakly global coin in the dynamic-broadcast message-oblivious fault model,where the constant probability for either common outcome is at least (1 - $) /2e, provided t <= $n (where $ is any constant less than 1).

PROOF

Refer to [6_0].

THEOREM 6.2:

The function COIN_TOSS produces a weakly global coin in the dynamic-reception message-oblivious fault model, when $t <= n(1/4 - e)$ for some constant $e > 0$. If $t = n (1/4 - e)$, the probability for either common outcome is at least o< / 2e, where o< = 8e / (4e + 5).

PROOF

Refer to [6_0].

THEOREM 6.3:

The function COIN TOSS2 produces a weakly global coin in the dynamic reception message-oblivious fault model, when $t < n/2$. The probability for either common outcome is at least $(1 - (t / n)) /2e$.

THEOREM 6.4:

Under the assumption (*) (given in section 4.6.3), if all processors hold the same encryption and decryption key, then for polynomially many repeated calls of the function COIN_TOSS , each call produces a weakly global coin in the message-dependent fault models. This procedure is correct, provided $t <= n$, where is any constant less than 1 for the static and dynamic - broadcast case, and $t < n/2$ for the dynamic - reception case. The probabilities of each outcome are as in Theorems 6.1, 6.2, and 6.3 respectively.

PROOF

Refer to [6_0].

THEOREM 6.5:

Under the assumption(*), but without assuming common, predistributed encryption and decryption keys, polynomially many repeated calls of the function COIN_TOSS_4, each produce a weakly global coin in the message-dependent dynamic brodcast and dynamic reception fault models provided that $t < n/2$.

THEOREM 6.6:

The function ASYNCHRONOUS_COIN_TOSS_1 produces a weakly global coin in the asynchrnonous, message-oblivious fault model, provided $t < ((3 - \sqrt{5}) /2)n$.

PROOF

Refer to [6_0].

THEOREM 6.7:

Under the assumption (*), if all processors hold the same encryption and decryption key, then polynomially many repeated calls of the function ASYNCHRNOUS_COIN_TOSS_2 produce a weakly global coin in the asynchnous message dependent model provided $t < ((3 - \sqrt{5}) / 2)n$.

PROOF

Refer to [6_0].

THEOREM 6.8:

Under the assumption (*) but without assuming common, predistributed encryption and decryption keys, repeated calls of a modified function ASYNCHRNOUS_COIN_TOSS_2 preceded by a four round encryption key distribution phase round a weakly global coin in the message dependent asynchronos fault model that $t < ((3 - \sqrt{5}) / 2)n$.

LEMMA 6.9:

During each epoch, both of the values 0 and 1 are never sent in any execution of round 2 (step 10).

PROOF

It can be proved by a simple counting argument.

THEOREM 6.10:

The algorithm has the following three parts

Validity : If value v si distributed as input to all processors decide v during each epoch 1.

Agreement : Let e be the first epoch in which a processor decides. If processor P decides v in each epoch e, then by the end of epoch e+1 all processors decides v.

Termination : (a) In any epoch e, if the epoch is not bivalent at the point when the fastest processor begins executing step 18, then there is at least one value that, if it is adopted by $\lfloor n/2 \rfloor + t + 1$ processors executing the assignment in step 18, will cause each processors to decide by the end of epoch e +1, and otherwise

(b) in any epoch, e, if there is a value that is adopted by $\lfloor n/2 \rfloor + t + 1$ processors executing the assignment in step 18 then epoch e + 1 is not bivalent at the point that the majority value of COIN_TOSS in epoch e is uniquely determined.

__THEOREM 6.11:__

Using the agreement algorithm with coin toss as a subroutine, agreement is reached in constant expected number of rounds, provided the number of fault t satisfies

(a) $t < n/2$ for the all varients of the synchronous model;

(b) $t < ((3 - \mathrm{sqrt}(5))n$ for all varients of the asynchronous mode.

THEOREM 7.2.1:

If agreement algorithm A guarantee SBA for each history with at most t orderly crash faults then A require at least min(n - 1, t + 1) rounds to reach SBA in any serial history.

PROOF:

Given in section 4.7.1

LEMMA 7.2.2:

Let H and J be histories in U. If A uses k rounds to reach SBA in J and Hk is witness equivalent to Jk, then A uses K rounds to rech SBA in H, and H and J are output equivalent.

PROOF:

The proof for this lemma is straightforward, but long. Refer to [7_0] for the proof.

LEMMA 7.2.3:

If e is a significant outedge of a candidate p in round k <= t of a serial history H, then there is a serial history J such that Jt is witness equivalent to Ht and Jk is identical to Hk except that the state of the message at e is altered (from correct to absent or vice versa).

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.2.4:

If H and J are serial hostories then Ht is witness equivalent to Jt.

PROOF:

Refer to [7_0] for the detailed proof.

COROLLARY 7.2.5:

Algorithm A require at least min(n - 1, t + 1) rounds to reach SBA when there are actually no faults.

THEOREM 7.3.1:

Let A be an agreement algorithm that reaches EBA in histories of U(A, t). Then there is a history in U(A, t) with only f faults in which A requires at least min(min - 1, t + 1, f + 2) rounds to reach EBA.

PROOF:

Given in section 4.7.2

LEMMA 7.3.2:

Let H be an f-serial history. Then there is no critilical edge in round f from a processor that is an f-candidate in round f of H.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.3.2:

If A reaches EBA for all histories with at most t faults and if A reaches EBA within min(t, f + 1) rounds for all histories with at most 1 faults, then all f-serial histories are output equivalent.

PROOF:

Refer to [7_0] for the detailed proof.

THEOREM 7.4.1

Execution of EAGREE by n > max(4t, 2(t + t - 1)^2)) processors results in EBA within min(f+2.t+1)rounds, where f, the actual number of faults does not exceed t.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.2 :

Suppose no correct processor is stopped at round i - 1 and let p, q, and r be correct processors. Then, at the end of round i, no correct processor has the name of a correct processor in X, every correct processor has pqs=rqs, and all correct processors share the same value for qs.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.3 :

If a correct processor is convicted at round i <= i + 1, then the value it has for s must have become persistent by round i -1.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.4 :

If a value becomes persistent before round t + 1, then it remains persistent throughout the execution of the algorithm and is given as output by each correct processor. If a value become persistent before rount t, then all correct processors are convicted at most two rounds later.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.5 :

If the origin is correct, then all correct processors will output its value.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.6 :

If for some i, 2 <= i <= t, a fault p separates a witness set from all other correct processors at round i, and if some correct processor is not convinced by round min(i + 2, t = 1), then there are correct processors that donot have p in their set X by the end of round i, but by roundI+1 each correct processor will have p in X and pPs and ps to the default value 0.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.7 :

If there is a correct processor is not convinced by round i + 2 with 1 <= i < t - 1, then there is a set { p1 | 1 <= j <= i } of i distinct faulty processors such that, for each j, each correct processor has pj in X and value pjs defaulted to 0 by the end of round j + 1 (and in each succeeding round).

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.8 :

If there are only f<t faults, then all correct processors are convinced by round f + 2.

PROOF:

Refer to [7_0] for the detailed proof.

LEMMA 7.4.9 :

All correct processors have the same value stored in s by round t + 1.

PROOF:

Refer to [7_0] for the detailed proof.

CHAPTER FOUR

ALGORITHMS FOR REACHING AGREEMENTS

MODULE ONE

The problem addressed here concerns a set of isolated processors, some subset of which may be faulty, that communicate only by means of two party messages. Each nonfaulty processor has a private value of information that must be communicated to each other nonfaulty processor. Nonfaulty processors always communicate honestly, whereas faulty processor may lie. The problem is to devise an algorithm in which processors communicate their own value and relay values received from others that allows each nonfaulty processor to infer a value for each other processor. The value infered for a non faulty processor must be that processor's private value and the value inferred for a faulty one must be consistent with the corresponding value inferred by each other nonfaulty procesor. Our results are formulated using the notion of INTERACTIVE CONSISTENCY, as defined in chapter 2 [1_0]. This problem is essentially the same as the Byzantine general metaphor in [2_1] or Byzantine general problem in [2_0]. (refer to chapter 2 for definations).

It is shown that the problem is solvable for, and only for n >= 3m + 1, where m is the number of faulty processors and n is the total number. It is also shown that if faulty processors can refuse to pass on information but cannot falsely relay information, the problem is solvable for arbitrary n >= m >=0. This weaker assumption can be approximated in practice using cryptographic methods.

In the following section we give algorithms devised to guarantee interactive consistency for and only for n, m such that n >=3m+1.

In section 4.1.1, we consider the single fault case that is m=1. We show that a minimum of four processors are required for this case.

Following this, in section 4.1.2, we consider a general algorithm for n >= 3m + 1. It is also proved that these algorithms assure interactive consistency (i.c.).

In section 4.1.3 a proof of impossibility for n $\leqslant$ 3m +1 is given.

In section 4.1.4, it is shown that interactive consistency can be assured for arbitrary n >= m >= 0 if it is assumed that faulty processors do not pass on information obtained from other processors but cannot false report this information. This case can be compared with the fail-stop case in Module Four. This can be implemented using authenticators and an algorithm using authenticators is the last algorithm presented.

SECTION 4.1.1 : THE SINGLE-FAULT CASE

Here we consider a procedure for obtaining interactive consistency in the simple case of m=1, n=4.

ALGORITHM:

The procedure consists of an exchange of messages, followed by the computation of the interactive consistency vector on the basis of the results of the exchange.

Two rounds of information exchange are required. In the first round the processors exchange their private value. In the second round they exachange the results obtained in the first round. The faulty processor (if there is one) may "lie,", or refuse to send messages. If a nonfaulty processor p fails to receive a message it expects from some other processor, p simply chooses a value at random and act as if that value had been sent.

The exachange having been completed, each nonfaulty processor p records its private value vp for the element of the interactive consistency corresponding to p itself. The element corresponding to every other plrocessor q is obtained by examining the three received reports of q's value (one of these was obtained directly from q in the first round, and the others from the remaining two processors in the second round). If at least two of the three reports agree, the majority value is used. Otherwise a default value such as "NIL" is used.

PROOF showing that this procedure assures interactive consistency, is given below.

First note that if q is nonfaulty, p will receive Vq both from q and and from the other nonfaulty processor(s). Thus p will record Vp for q as desired. Now suppose q is faulty. We must show only that p and the other two nonfaulty processors record the same value for q. If every nonfaulty processor records NIL, we are done. Otherwise, some nonfaultdy processor, say p, records a non-NIL value v, having received v from at least two other processors. Now if p received v from both of the other nonfaulty processors,each other nonfaulty processor must receive v from every plrocessor other tdhan p (and possibly from p as well);every nonfaulty processor will thus record v. Otherwise,p must have received v from all processors other than some other non faulty processor p'. In this case p' received v from all processors other than q (so p' records v), and other nonfaulty processors received v from all processors other than p. All nonfaulty processors therefore record v as required.

SECTION 4.1.2 : AN ALGORITHM FOR N >= 3M + 1

The procedure given in the last section requires two rounds of information exchange, the first cosisting off the form "my private value is" and the second consisting of communications of the form "processor x told me his private value is....". In the general case of m faults, m+1 rounds are required. In order to descibe the algorithm, it will be convenient to characterize this exchaange of messages in a more formal way.

ALGORITHM:

Let p be the set of processors, v a set of value for k>=1 and o- is a k-level scenario for string w=p1p2 ..... pr, 2<=r<=k+1. Note that for a given subset of nonfaulty processors, only certain mapping are possible scenarios,in particular, since nonfaulty processors are always truthful in relaying information, a scenario must satisfy

o-(pqw)= o-(qw)

for each nonfaulty processor q, arbitrary processor p, and string w.

The message a processor p receives in a scenario o- are given by the restriction o-p of o- to the strings beginning with p. The procedure we present now for arbitrary m>=0, n>=3m +1, is described in terms of p's computation, fora given o-p,of the element of the interactive-consistency vector corresponding to each processor q.

The computation is as follow:

(1) If for some subset Q of p of size > (n+m)/2 and some value v, o-p(pwq)=v for each string w over Q of length <=m, p records v.

(2) Otherwise, the algorithm for m-1, n-1 is recursively applied with p replaced by P - {q}, and o-p by the mapping $\overline{\text{o}}$-p defined by $\overline{\text{o}}$-p(pw)=o-p(pwq)

for each string w of length <=m over P-{q}. If at least {floor operator of (n+m)/2} of the n-1 elements in the vector obtained in the recursive call agree, p records the common value, otherwise p records NIL.

Note that o-p corresponds to the m_level subscenario of o- in which q is excluded and in which each processor's private value is the value it obtains directly from q in o-. Note also that the algorithm essentially reduces to the one given in the last section in the case m=1.n=4.

<u>PROOF</u> that the algorithm given above does indeed assure interactive consistency proceeds by induction on m:

<u>Basis</u> <u>m=0</u>. In this case no processor is faulty, and the algorithm always terminates in step (1)with p recording Vq for q.

__Induction Step m>0.__ First note that if q is nonfaulty, o-p(pwq) = Vq for each string w (including the empty string) of length <=m over the set of nonfaulty processors. This set has n-m members (which, since n>3m, is>(n+m/2) and so satisfies the requirements,for Q in step(1) off the algorithm. Any other set satisfying these requirements,moreover,must contain a nonfaulty processor (since it must be of size > (n+m)/2. and n >= 3m+1) and must therefore also yield Vq as the common value. The algorithm thus terminates at step(1), and p records Vq and q as required.

Now suppose that q is faulty. We must show that the value p records for q agrees with the value each other nonfaulty processor p' records for q.

First consider the case in which both p and p' exit the procedure at step(1), each having found an appropiate set Q. Since each such set has more than (n+m)/2 members, and since p has only n members in all, the two sets must have more than 2((n+m)/2)-n=m common members. Since atleast one of these must be nonfaulty, the two sets must give rise to the same value v, as required.

Next suppose that p' exits at step(1), having found an appropriate set Q and common value v,and that p executes step(2). We claim that in the vector off n-1 elements that p compuites in the recursive call, the elements corresponding to members of

$\bar{Q}$ = Q - {q} are equal to v. Since $\bar{Q}$ has at least {floor operator of (n=m)/2} members, it will then follow that p records v in accordance with step(2). To see that the elements corresponding to members of $\bar{Q}$ are indeed equal to v,recall that the mapping $\bar{o}$-p that p uses to compute the vector in the recusive call is the restriction, to strings beginning with p,of the m-level scenario o-p defined by

$$\bar{o}\text{-}p(w) = o\text{-}(wq)$$

for each string w of length <=m over P - {q}. By the induction hypothesis, this vector is identical to the one p' would have computed using the restriction o-p of o- had p' made the recursive call. Moreover, the value p' would have computed for the element of this vector corresponding to a given q' in Q must be v, since Q and v satisfy step (1) of the algorithm. (Note that Q is of size >=[(n+m)/2}>=[(n-1)]+(m-1)]/2, and that o-p(p'wq') = o-p(p'wq'q)= v for each string w of length <=m-1 over $\bar{Q}$.) The case in which p exits at step (1) and p' exits at step(2) is handled similarly.

In the one remaining case, both p and p' exit at step (2). In this case both recurse and must, by the induction hypothesis, compute exactly the same vector and hence the same value for q.

Q.E.D.

SECTION 4.1.3 : PROOF OF IMPOSSIBILITY FOR N < 3M +1

The procedure of the last section guarantees interactive consistency only if n>=3m+1. In this section it is shown that the 3m+1 bound is tight. We will prove not only that it is impossible to assure interactive consistency for n<3m + rounds of information exchange, but also that it is impossible, even allowing an infinite number of round exchange (i.e. using scenario mapping from all nonempty strings over P to V).

Just to gain some intuitive feeling as to why 3m processors not sufficient,cosider the case of three processors A, B, C of which one say C, is faulty. By prevaricating in just the right way, c can thwart A's and B's efforts to obtain consistency. In particular, C's messages to A can be such as to suggest to A that C's private value is say ,1, and that B is faulty. Similarly, C's messages to B cab be such a to suggest to B that C'sprivate value is 2 and that a is faulty. If c plays its cards just right. A will not be able to tell whether B or C is faulty, and B will not be able to tell whether A or C is at fault.A will thus have no choice but to record 1 for C's value. while B must record 2, defeating interactive consistency.

The precise statement of the impossibility result and its proof is given as a THEOREM 1.1 in chapter 3 . This is using the formal definations of (i) scenario
(ii) o- consistency with N
(iii)interactive consistency for m faults
given in chapter 2.

SECTION 4.1.4 : AN ALGORITHM USING AUTHENTICATORS

The negative result of the last section depends strongly on the assumption that a faulty processor may refuse to pass on values it has recieved from other processors or may pass on fabricated values. This section addresses the situation in which the latter possibilty is precluded. We will assume, in other words, that a faulty processor may "lie" about its own value and may refuse to relay values it has received, buyt may not relay altered values without betraying itself as faulty.

In practice, this assumption can be satisfied to an arbitrarily high degree of probability using authenticators. A processor p constructs an authenticator for a data item d by calculating Ap[d], where Ap is some mapping knoawn only to p. It must be highly improbable that a processor q other than p can generate the authenticatior Ap[d] for a given d. At the same time, it must be easy for q to check, for a given p,v, & d, that v=Ap[d]. The problem of devising mappings with these properties is a cryptographic one. Methods for their constructions are discussed in [1_2] and [1_3]. For many application in which faults are due to random errors rather than to malicious intelligence, any mappings that "suitably randomize" the data suffice.

A scenario σ is carried out in the following way. Let v=σ(p) designate p's private value. p communicates this value to r by sending r the message consisting of the triple <p,a,v>, where a=Ap[v]. When r receives the message, it checks that a = Ap[v]. If so, r takes v as the value of o-(rp). Otherwise r lets o-(rp) = NIL. More generally, if receives exactly one message of the form (p1, a1(p2,a2,a2...(pk,ak,v)...)), where ak = Ak[v] and for 1 <= i <= k-1, ai = Ai[(pi+1,ai+1...(pk,ak,v)], then o-(rp1...pk) = v. Otherwise o-(rp1...pk) = NIL.

A scenario o- constructed in this way is consistent with a given choice N of nonfaulty processors, if for all processors p is an element of N, q belongs to set P and strings w, w' over P.

(i) o-(qpw) = o-(pw)

(ii) o-(w'pw) is either o-(pw) or NIL .

Condition (i) ensures that nonfaulty processors are always truthful. Condition (ii) guarantees that a processor cannot relay an altered value of information recieved from a nonfaulty processor. That is it may fail to relay and act like a dead process but it will never tell a lie that is it will never act like a malacious process.

Next, we consider an algorithm, using (m+1)-level authenticated scenarios, that guarantees interactive consistency for any n>=m. As before, the procedure is described in terms of the value a nonfaulty processor p records for a given processor q on the basis of o-p:

ALGORITHM:

Let Spq be the set of all non-NIL values O-p(pwq), where w ranges over strings of distinct element with length <=m over P - {p,q}. If Spq has exactly one element v, p records v for q; otherwise, p records NIL.

PROOF: that the above algorithm assures interactive constituency.

Consider first the case in which q is nonfaulty. In this case o-p(pwd) is either o-(q) or NIL for each appropriate w by condition (ii) Since in particular, o-(pq) = o-(q) by (i) Spq = {o-(q)}.p thus records o-(q) for q as required.

If q is faulty, it suffice to show only that for each two nonfaulty processors p and p', Spq = Sp'q. So suppose v belongs to Spq, i.e., v= o-p(pwq) for some string w having no repetitions, with length <= m over P - {p,q}. If p' occurs in w (say w=w1p'w2), then o-(pwq) = o-(p'w2q) by (ii); hence v= o-(pwq) belongs to Sp'q. If p' does not occur in w aned w is of length <m , then pw is of length <=m; so v = o-(pwq) = o-(p'pwq) belongs to Sp'q.

Finally, if p' does not occur in w and w is of length m, w must be of the form w1rw2 where r is nonfaulty giving that v = o-(pwq) = o-(rw2q) (by (ii)) = o-(p'rw2q) (by (i)) belongs to Sp'q. In each case v belongs to Sp'q. A symmetrical argument shows that if v belongs to Sp'q, v belongs to Spq. Hence Sp'q = Spq as required. Q.E.D.

MODULE TWO

In the previous module we considered Interactive consistency problem which is essentially the same as Byzantine general problem.

The Byzantine Generals Problem requires process to reach agreement upon a value even though some of them may fail. In this modeule the problem is weakend by allowing them to agree upon an incorrect value if a failure occurs.

The transaction commit problem for a distributed database is a special case of the weaker problem. It is shown that, like the original Bynzantine Generals Problem the weak version can be solved only if fewer than one theird of the processes may fail.

Unlike the original problem an approximate solution exists that can tolerate arbitraly many failures.

In section 4.2.1 it is first shown that no solution to the WBG. problem exists if 1/3 or more of the processes are faulty. Hence the WBG problem discussed by L. Lamport in [2_0] is solvable in precisely those situations in which the original Byzantine General problem [2_1, 1_0] is.

In section 4.2.2, we show that if condition 2 of the WBG problem is replaced by a weaker condition requiring only approximate equality, then the problem is solvable with any number of faulty processes. More precisely, if the set of possible values is a bounded set of numbers, then for any $\epsilon > 0$ there is an algorithm which garantees that the values chosen by any two nonfaulty processes differ by less than ϵ. It was shown in [2_1] that no such approximate solution exists for the original Byzantine Generals Problem.

In section 4.2.3, an algorithm that works with any number of faulty processes is given. This algorithm requires the processes to send an infinite number of messages before choosing their values and hence this "solution" is of no practical interest, since it cannot be implemented. Its interest lies in fact that the original Byzantine Generals Problem does not possess such a "solution". Hence, the WBG Problem is in some sense strictly weaker than the Byzantine Generals Problem.

SECTION 4.2.1 : IMPOSSIBILITY RESULT

A proof is given to show that no solution to the WBG problem exists if one-third or more of the processes are faulty.

Let

P donate the set {0, ... , n - 1}, of processes

P* the set of all finite sequences of elements of p (including the null sequences).

II denote the set of all finite sequences of the form 0,# with
# belonging to P* -i.e., all elements of P* whose first element is 0.

0, pi.....pk is a path of length k traveled by a message that starts at process 0 and is relayed via processes pi.....pk-i to process pk.

IIi denote the subset of II consisting of all sequences ending in i - i.e. all message paths leading from process 0 to process i.

A Scenario % is amapping from II into a set of values V. If we think of an element # of II as a message path, then %(#) is the contents of the message received at its final destination We say that process i is nonfaulty in a scenario % if for every message path #,i belonging to II and every j belonging to P: %(#,i,j) = %(#,i). That is, i is nonfaulty in % if process i correctly relays all messages. If all processes are nonfaulty in %,then %(#) = %(0) for all # belonging to II, which means that every destination process of a path receives values send by process 0.

A solution to the WBG problem consists of an algorithm by which the processes send messages to one another based upon the contents of messages already received. Initially, the only information is the value v, which is known only to process 0. Therefore, all information travels along path in II. To send the maximum amount of information to one another, Process 0 would send the value v to all processes, and then processess would send one another the contents of every message they receive. Thus, if %(0) equals v, then a scenario % describes the maximum amount of information that the processes could send to one another. A nonfaulty process can always ignore information that it receives, and a faulty process can do anything - including guess any information that was withheld from it. Hence any algorithm for choosing values based upon the entire scenario %. Such an algorithm is called m-fault WBG Algorithm B and is defined in chapter 2.

Next it is shown that no m-fault WBG alogrithm exists if 3<n<3m. (The problem becomes trivinal if n <= 2)

If the value of Bi(%i) depended upon the entire infinite i-scenario %i, then the alogrithm B would require an Infinite amount of message passing and would not be a real soluion to the WBG problem. We Consider the defination for "finiteness of a WBG algorithm B", given in chapter 2,where II(k) is defined to be the set of message of length at most ki and IIi(k) = II(k) intersection IIi.

A finite WBG alogrithm is one in which for every scenario, there is a k such that each process can choose its value after k rounds of message passing. This is a natural definition, since it insures that every process is eventually able to choose a value. However , it does not immediately rule out the possibility that the required number of rounds k can become arbitrarily large. In LEMMA 2.1 it is shown that this is not the case, and that a single value of k can be chosen for all scenarios.

To prove the nonexistence of an m-fault algorithm when n <= 3m, we first prove the noneexistence of a 1 - fault algorithm for n = 3. For this we assume that P = {0,1,2} .

We define the signed distance function & on P by:

$$\&(0,1) = \&(1,2) = \&(2,0) = 1,$$

$$\&(i,j) = -\&(j,i).$$

For any path # = 0,p1.....pk we define o-(#) to equal

$$\sum_{i=1}^{k} \&(p_{i-1} , p_i)$$

If we think of the processes 0,1 and 2 being arranged clockwise in a circle then &(i,j) is the clockwise angular distance from i to j (where a distance of 3 represents a full circle), and o-(#) is the singed angular distance traavelled by the path #.

Consider Lemma 2.2 and 2.3 given in chapter 2. Note that the two conditions of Lemma 2.3 define the values of all messages in the scenario $\%^{(r)}$ except for the ones that Process r sends to itself.

Lemma 2.4 is a simple corollary of Lemma 2.3 .

The main result is proved in form of Theorem 2.1 .

SECTION 4.2.2 : APPROXIMATE SOLUTION

The approximate solution of the WBG problem that works in the presence of any number of faulty processes, is described next. By taking the limit of a sequence of such solutions, we obtain an exact solution using an infinite number of messages, whiich is given in the following section. In order to make the concept of an approximate solution meaningful, we assume that the set V of possible values is a set of real numbers.

For each integer k>0, we define an algorthm AG(k) that requires k rounds of message passing. Rather than describing it in terms of formal scenarios we will simply talk about processes sending messages to one anotehr. Nonfaulty processes are constrained to follow the algorithm while faulty ones may do anything. We assume that a faulty process sends every message that it is supposed to although possibly with an incorrect value. However value it sends is assumed to be some element of V. It should be obvious how this description can be translated into a definition of mappings on i-scenarios.

ALGORITHM AG$^{(K)}$:

The following k rounds of message passing are executed to compute the value $v_i^{(r)}$ for i belonging to P and $1 <= r<=k$.

(1) ROUND 1:

(a) Process 0 ends the value v to every Process i

(b) each Process i sets $vi^{(1)}$ equal to the value it receives from Process 0.

(2) ROUND r: ($1 < r <= k$)

(a) Each Process j sends the value $vj^{(r-1)}$ to every Process i.

(b) Each Process i sets $vi^{(r)}$ equal to the maximum of the n values it recieves.

(3) Each Process i then sets vi equal to the average of the k values $vi^{(r)}$.

Theorem 2,2 proves that the above algorithm is an approximate solution to the WBG problem.

SECTION 4.2.3 : INFINITE SOLUTION

To construct an infinite message solution to WBG problem, we let each Process i take as its value of v_i the limit as k goes to infinity of the value obtained by the algorithm $AG^{(k)}$. If the set V is unbounded then this limit could be infinte in which case some arbitary preassigned value is used. This gives us the following.

ALGORITHM $AG^{(\infty)}$:

Compute the value $v_i^{(r)}$ as described in Algorithm $AG^{(k)}$, for all i belonging to P and r >= 1. For each i, define v_i to equal $\sup\{v_i^{(r)} : r>=1\}$, where ~ is interpreted to be some arbitary fixed element of V.

We now show that $AG^{(\infty)}$ is a "solution" to the WBG problem that can tolerate any number of faults. Since it requires choosing a value based upon an infinte sequence of messages, it cannot be regarded as a solution in any practical sense.

Theorem 2.3 shows that $AG^{(\infty)}$ is an infinite solution to WBG algorithm.

MODULE THREE

After going through modules one and two, we know that solutions are known for the "Byzantine Generals" problem [2_1, 2_0, 1_0], which is with reference to synchronous system.

In this module we consider an asynchronous system and problem of reaching agreement here, is called consensus problem.

Refer to chapter 2 for the defination of consensus problem.

The consensus problem involves an asynchrnous system of processes some of which may be unrealliable. The problem is for the relliable processes to agree on binary value. In this module it is shown that every protocol for this problem has the possibility of nontermination even with only one faulty porocess.

For the main result of Fischers's work refer to section 4.3.1.

In section 4.3.2, an algorithm is presented, which solves the consensus problem for N processes, provided majority of the processes are nomn faulty and no process dies during the execution of the algorithm.

SECTION 4.3.1 : MAIN RESULTS OF FISCHER'S WORK

A consensus protocol P is totally correct in spite of one fault if it is partially correct, and every admissible run is a deciding run.

The main theorem given by Fischer, Lynch and Paterson shows that every partially correct protocol for the consensus problem has some admissible run that is not a deciding run. This theorem is stated and proved in chapter 2 as theorem 3.1 .

This theorem uses two lemmas, Lemma 3.2 and 3.3 which are again stated and proved in chapter 3.

SECTION 4.3.2 : ALGORITHM FOR INITIALLY DEAD PROCESS

Here in this section, we exhibit a protocol that solves the consensus problem for N processes, based on certain conditions.

This protocol was given by Fischer based on the following conditions :

a. Majority of processes are non faulty.
b. no process dies during the execution of the protocol.
c. no process knows in advance which of the other processes are initially dead and which are not.

This protocol works in 2 stages :

<u>Stage 1</u>

Here the protocol constructs a directed graph in the following way :

1. Processes construct a Directed graph G with a node corresponding to each process.
2. Every process broadcasts a message containing process number.
3. It then listens for message from L-1 other processes.

$$L = \lceil (N+1)/2 \rceil$$

4. G has an edge from i to j if and if only j recieves a message from i. Thus G has an indegree given by :

$$\text{Indegree } (G) = L - 1$$

<u>Stage 2.</u>

In this stage the processes construct G+ that is the transitive closure of G.

Each process K will know about :

1. all of the edges (j,k) incident on K in G+.
2. initial values of all such j.

This stage is carried out in the following stages :

1. Each process broadcasts to all other processes the following :
   - its process number.
   - the initial value.
   - names of L-1 processes it needs from the first stage onwards.

2. It then waits until it has received a stage 2 message from every ancestor in G that it knows about.
3. Waiting continues till such time as all currently known about processes have been heard from.
4. Using the information obtained each process computes all the edges of G+ incident on each of its ancestors.
   - each process knows all of its ancestors.
   - edges of G incident on them.
5. Determine the ancestors belonging to an initial cliques of G+.(i.e. a clique with no incoming edges).
6. It can be shown that there can be only one initial clique, i.e. cardinality >= L.

   Every process that completes 2nd stage knows exactly the set of processes comprising it.

Finally each process makes a decision based on :

1. initial values of processes in the initial cliques.
2. any agreed upon rule.

Since all processes knows the initial values of all members of the initial clique, they all reach the same decision.

We now arrive at theorem 3.2 given in chapter 3.

MODULE FOUR

In the previous module we presented a very primtive consensus protocol. This module is an extension of the previous module in the sense that it gives protocols that enable a system of n asynchrnous processes, some of which are faulty, to reach agreement.

Here we consider two kinds of faulty processes : fail stop processes that can only die and malicious processes that can also send false messages. The class of asynchrnous systems with fair shcedulers is defined and consensus protocols that terminate with probability 1 for these systems are investigated. With fail stop process it is shown that $\lceil (n+1)/2 \rceil$ correct processes are necessary and sufficient to reach agreement. In the malicious case it is shown that $\lceil (2n+1)/3 \rceil$ correct processes are necessary and sufficent to reach agreement. This is contrasted with an earlier result [3_0] stating that there is no consensus protocol for the fail stop case that always terminate within a bounded number of steps, even if only one process can fail.

The possibility of reliable broadcast (Bynzantine Agreement) in asynchrnous systems is also investigated. Asynchrnous Bynzantine Agreement is defined and it is shown that $\lceil (2n+1)/3 \rceil$ correct processes are necessary and sufficient to acheive it.

The solutions in this module are different from solution given in module three because here we consider protocols which may never terminate, but this would occur with probability 0, and the expected termination time is finite. This is done by postulating some probabilistic behavior about the message system. This is done making probabilistic assumptions on the behaviour of a scheduler (defined in chapter 2). Here class of fair schedulers is considered (defined in chapter 2).

In section 4.3.1 we consider the fail-stop case. In this section we first find the maximum number of faulty proceses which any consensus protocol can manage. In other words a lower bound on the number of correct processes is derived. Next a fair scheduler is defined and finally a $\lfloor (n-1)/2 \rfloor$ resilient consensus protocol is derived.

In section 4.3.2 we consider the malacious case. Here we first discuss the model to be considered. Then we find the maximum number of faulty proceses which any consensus protocol can manage. That is, the lower bound on the number of correct processes is derived and finally a $\lfloor (n-1)/3 \rfloor$ resilient consensus protocol is derived.

In section 4.3.3 we consider Asynchronous Byzantine Agreement. Here we first discuss the problem. Then we find the lower bound on the number of correct processes and finally a protocol that acheives Asynchronous Byzantine agreement for $k= 1$ to $\lfloor (n-1)/3 \rfloor$ malacious processes is derived.

SECTION 4.4.1 : FAIL-STOP CASE

LOWER BOUND ON NUMBER OF CORRECT PROCESSES

We could have undetectable deaths during the execution of the protocols, this implies that, at any stage of the protocol, processes will have to act depending on partial information about the state of the system. This is formalized by the lemma 4.1 given in chapter 3.

Refer Theorem 4.1 give in chapter 3. It proves a very important result.

FAIR SCHEDULERS

We may view Protocols for asynchronous systems as consisting of rounds. While in round t, a process sends messages to every other process, and waits until it recieves n - k messages sent by different unique processes in round t. After this the process changes its state, and starts round i+1. We notice that the new state is a function of the old state and the messages are recieved in round t.

Hence the processes cannot wait for more than n - k messages as there is always the possibility in which all k faulty processes do not send any messages in round t. We define R(q, p, t) to be the event that p recieves a message from q in round t. The progress of this system depends on the joint probability distribution of the R(q, p, 1) events, which is determined by the schedular.

We can say that a schedular is fair provided the following conditions prevail:

1). For any processes, p and q, and round t, there is a positiv constant e such that a Pr[R(q, p, t)]>t

2). For any distinct processes r, p, and q, and round t, the event R(q, r, t) and R(q,, p, t) are independent.

These conditions in particular, guarantee that, for any round k, there is a constant probability p that all processes recieve n-k messages from the same set of correct processes.

$\lfloor (n-1)/2 \rfloor$ RESILIENT CONSENSUS PROTOCOL

Here we describe a k-resilient consensus protocol for a system with a fair schedular and k = 1,2.....$\lfloor (n - 1)/2 \rfloor$. The protocol consists of rounds as seen earlier .

The state of a process and the messages exchanged consist of a phase number, a binary value, and a cordinality.

In each phase, a process does following step :

(1) A process sends a message with its state to all the processes.

(2) Then the process waits for messages.

(3) When a process receives n - k messages, with same phase number, it considers the sets of messages with value 0 and value 1, respectively.

(4) If there is a message with value i and cardinality > n/2 then will be called a witness for i.

(5) If a process recieves a witness for i, it changes its
        value to i
    else value = value of the largest message set.

(6) A process changes its cardinality to the size of the message set with value i.

(7) The process starts a new phase.

A process decides i if it receives more than k witnesses for value i. Since there are enough witnesses for that value in the message system so force the rest of the processes to reach the same decision.

Refer to theorem 4.2 to see that the following algorithm is a k-resilient consensus protocol for the fail stop case, for any k, $0 <= k <= \lfloor (n - 1) / 2 \rfloor$.

The algorithm is given on the next page.

```
process p:k-consensus

        value:integer init(ip)
        cardinality:integer init(1)
        phaseno:integer init(0)
        witness_count:array[0..1] of integer init(0)
        message_count:array[0..1] of integer init(0)
        msg:record of
              phaseno:integer
              value:integer
              cardinality:integer

        while (witness_count(0)<=k and witness_count(1)<=k)
             message_count:=witness_count:=0

        for all q, 1<=n, send(q, (phaseno,value,cardinality))

             while (message_count(0)+message_count(1)<n-k)
               receive(msg)
               case
                 (msg.phaseno=phaseno):
                  begin
                      message_count(msg.value):=message_count(msg.value)
                      if msg.cardinality>n/2
                      then
                 witness_count(msg.value):=witness_count(msg.value)+1
                  end
                 (msg.phaseno>phaseno):
                     send(p.msg)
               end
               if there is i such that witness_count(i)>0
               then value:=i
               else if message_count(1)>message_count(0)
                    then value:=1
                    else value:=0
                   cardinality:=message_count(value)
                   phaseno:=phaseno+1
             end

    let i be such that witness_count(i)>k

    dp:=i

    for all q, 1<=q<=n,
        begin
          send(q,(phaseno,value,n-k))
          send(q,(phaseno+1, value,n-k))
        end
```

SECTION 4.4.2 : MALICIOUS STOP CASE

MODEL

Here we describe a model in which we investigate a stronger failure behavior of the processes.

A malicious process is one which :

*** can send false and contradictory messages (even according to some malicious design),

*** can fail to send messages

*** can change its internal state to any other state.

The message system must be so designed that it must provide a way for correct processes to verify the identity of the sender of each message. Because if this was not done then one malicious process can impersonate the whole system, leading the correct processes to conflicting decisions.

The rest of the model described earlier in section II with the with the following additional definitions.

A schedule is said to be legal if all its steps are according to the protocol.

A configuration C is legally reachable if it is reachable by a legal schedule.

Henceforth, we reserve the notation |--- to denote only transitions by legal schedules.

LOWER BOUND ON THE NUMBER OF CORRECT PROCESSES

Refer to Lemma 4.3 and theorem 4.3 for lower bound on the number of correct precesses for the malicious case.

$\lfloor (n-1)/3 \rfloor$ RESILIENT CONSENSUS PROTOCOL

Here in this section we present a k-resilient consensus protocol for a system with a fair schedular and $k=1,2....,\lfloor (n-1)/3 \rfloor$ malicious processes.

The state of a phase number, and a binary value. As seen earlier the protocol consists of phases in which processes send to each other their states. To overcome misleading messages from the malicious processes: we use the technique of initial and echo.

Protocol Steps :

(1) A process in each phase first sends its state to all the processes, and waits until it accepts messages from n-k processes by following steps:

(a) In this a process sends to all the other processes an initial message with its name and its state.

(b) After receiving the initial message, every process echoes it back to all the processes.

(c) Process p, at phase t, accepts a message with value i from process q if it receives more than (n+k)/2 messages of the form (echo, q,i,t)

(2) It changes its value to the majority of the values of the accepted messages.

(3) A process decides i if it accepts more than (n+k)/2 messages with value i.

We prove that, once a process decides i, thereafter all the other correct processes will have value i.

The protocol seen in Figure 2, shows that processes do not exit the protocol after they decide. This feature was done for notational convenience only, and it can be avoided in the following manner:

When process p decides i, it sends to all the processes the message(initial,p,i) and echoes of the form (echo,q,i) for all q's. The last messages are special so that whenever a process receives them, it sends them back to itself. Once a correct process has decided i, all the correct processes will have value i. Hence ,this procedure will have the same effect as the actual participation of p in the protocol.

Refer to theorem 4.4 to see that the following algorithm is a k-resilient consensus protocol for the malacious case, for any k, $0 <= k <= \lfloor (n - 1) / 3 \rfloor$

```
process p:k-consensus

        value:integer init($i_p$)

        phaseno:integer init(0)
        message_count:array[0..1] of interger init(0)
        echo_count:array[1..n:0..1] of integer init(0)
        msg:record of
             type:(initial, echo)
             from:integer
             value:integer
             phaseno:integer

    while(true)
      message_count:=0
      echo_count:=0
      for all q,1<=n,send(q(initial,p,value.phaseno))

      while(message_count(0)+message_count(i)<n-k)
      receive(msg)
      if it is the first message received from the sender
        with these values of msg.type, msg.from and msg.phaseno then
           case
             (msg.type=initial):
               for all q,1<=q<=n,
              send(q,msg.from,msg.value,msg.phaseno))
           begin
             echo_count(msg.from,msg.value):=echo_count(msg.from,
             msg.value)+1
             if echo_count(msg.from,msg.value):=(n+k)/2+1
             then message_count(msg.value):=message_count(msg.value)+1
           end
        (msg.type = echo and msg.phaseno>phaseno):
           send(p.msg)
          end
       end

       if message_count(1)>message_count(0)
       then value := 1
       else value := 0

       if there is such that message_count(0)
       then value := 1
       else value := 0

       if there is such that message_count(i)>(n+k)/2
           then d := $i_p$

       phaseno:=phaseno + 1

     end
```

SECTION 4.4.3 : ASYNCHRONOUS BYZANTINE AGREEMENT

We now come across a major problem of ensuring reliable broadcasts in distributed systems ,commonly known as Byzantine Agreement[4_6], Unanimity[4_2], or Interactive Consistency[4_7].

All earlier studies of Byzantine Agreement deal with a synchronous system of n processes, where upto k processes can be malicious. Some specially designated process is a transmitter that sends a value to all the rest of the processes. A Byzantine Agreement is achieved if the following holds:

1) All correct processes agree on the same value.

2) If the transmitter is correct, all the correct processes agree on its value.

We can view the whole system implicitly as in one of the following states :

a) "before broadcast,"

b) "executing the agreement protocol,"

c) "after broadcast."

Thus, queries about the transmitted value can be handled in a consistent manner by any correct process.

The differnce in this view comes when we consider asynchronous systems, . Some correct processes can proceed with the protocol and reach agreement while others may not yet be aware the the protocol has begun.

It may be insufficient to start up the process on the protocol even if a process receives a message from the transmitter or from other processes . We definately require some threshold activity to start up a process, a threshold that guarantees that all the other is necessary to start up a process, a threshold that guarantees that all the other correct processes will also start the protocol and will agree on the same value.

The following two conditions illustrate the necessity of such a scheme.

1) The transmitter is malicious. At time t0 it sends to k processes 0- messages, to a different set of k processes 1-messages, and none to the rest. All these messages are received at time t1. After that, the transmitter stops participating in the protocol. If we regard this as a sufficient condition to start up a Byzantine Agreement protocol, then the system can proceed and agree, let us say on 1, at time t2.

2) The transmitter is correct and sends 0-messages to all the processes. At time the same k correct processes as in condition 1 receive these 0-message. Also, k malicious processes receive 0-messages, but they treat them as if they were 1 messages. Any other messages from the transmitter will be received only at a time later than t2. Consider the system during the interval [t1,t2]. The processes view of the system is the same as in scenario 1, and therefore they can simulate it and agree on 1 at time t2, thus violating requirement 2 of the Byzantine Agreement.

et us now study the two ways to overcome this phenomenon. We an restrain the behaviour of a malicious transmitter (it will e enough to force it to send 2k+1 messages with the same alue). Another way, the one we adopt, is to regard certain iews of the system as insufficient to start the protocol. rocesses may not start, unless presented with a view that uarantees starting up and agreement of all the correct processes.

or an asynchronous Byzantine Agreement to be achieved the ollowing must hold :

) If the transmitter is correct, all the correct processes ecide on its value.

) If the transmitter is malicious, then either no correct rocess will decide they will all decide on the same value.

OWER BOUND ON NUMBER OF CORRECT PROCESSES

efer to theorem 4.5 for lower bound on the number of correct recesses for the asynchronous Byzantine agreement.

SYNCHRONOUS BYZANTINE AGREEMENT PROTOCOL

here are three types of messages in the protocol:initial, echo, nd ready. The protocol starts with:

(1) The transmitter sends the initial messages

(2) it then processes report to each other the value they recived via (echo,v) messages.

(3) If more than (n+k)/2 (echo.v) messages are received by a process, it announces it with (ready,v) messages.

(4) If a process receives 2k+1 ready messages of the same value, it decides that value.

Refer to theorem 4.6 to see that the following algorithm achieves Asynchronous Byzantine Agreement, for k=1 to $\lfloor (n-1)/3 \rfloor$ malacious precesses.

The algorithm is given on the next page.

```
      msg_count:array of [types:0..1] of integer

      msg:record of type:(initial,echo,ready)

      value:integer


      while(there is no i such that
              msg_count(initial,i)>=1 or
              msg_count(echo,1)>(n+k)/2 or
              msg_count(ready,1)>=k+1)
      receive(msg)


      if it is the first message received from the sender
      with these values of mesg.type,msg.from
      then msg_count(msg.type,msg.value)=msg_count(msg.type,value)+1
      end


     for all q, send(echo,i)


     while(there is no i such that
         msg_count(echo.1)>(n+k)/2 or
         msg_count(ready.1)>=k+1
     receive(msg)


     if it is the first message recieved from the sender
     with these values of msg.type, msg.from
     then msg_count(msg.type,msg.value)=msg_count(msg.type,msg.value)+
     end

    for all q, send(ready,i)


    while(there is no i such that
           msg_count(ready,i)>=2k+1)
    receive(msg)


    if it is the first message received rom the sender
              with these values of msg.type,msg.from
    then
     msg_count(msg.type,msg.value)=msg_count(msg.type,msg.value)+1
    end

decide i
```

MODULE FIVE

This paper considers a variant of the Byzantine Generals problem is considered, in which processes start with arbitory real values rather Boolean values or values from some bounded range, and in which approximate, rather than exact, agreement is the desired goal. Algorithms are presented to reach approximate agreement in asynchrnous as well as synchronous systems. The asynchronous agreement algorithm is an interesting contrast to a result of Fischer et al, who show that exact agreement with guaranteed termination is not attainable in an asynchrnous system with as few as one faulty process. This is what we considered in module three. The algorithm work by successive approximation with a provable convergence rate that depends on the ration between the number of faulty processes and the toal number of processes. Lower bounds on the convergence rate for algorithms of this form are proved and the algorithms presented are shown to be optimal.

In Section 4.5.1, we prove some combinatorial properties of the approximation functions on which the algorithms depend. Then, in section 4.5.2, synchronous model is introduced and the synchronous approximate agreement algorithm is presented. In Section 4.5.3, asynchronous problem is discussed and asynchronous approximate agreement algorithm is presented. In section 4.5.4, resilience properties of the algorithms are discussed.

SECTION 4.5.1 : PROPERTIES OF APPROXIMATION FUNCTIONS

Here, we state and prove the relevant properties of the approximation functions.

Refer to the definations related to multisets. Lemma 5.1 shows that the number of common elements in two nonempty sets is reduced by at most 1 when the smallest (or the largest) element is removed from each.

Lemma 5.2 extends the results of the lemma 5.1 to removing the j largest and j smallest elements.

Lemma 5.3 is fundamental to the correctness of the algorithms. It states that if V and U are multisets such that V contains at most j values not in U, then every value in $\text{reduce}^{j}(V)$ is in the range of U. For example, if the multiset of values held by nonfaulty processes at some point in the algorithm is U, and the multiset of values received by some process is V, then at most t of the values in V are not in U, where t is the maximum number of faulty processes. The lemma then states that $\text{reduce}^{t}(V)$ is a multiset whose range is contained in the range of the values of the nonfaulty processes. This property is essential in showing that the validity condition is satisfied.

THE APPROXIMATION FUNCTIONS:

Suppose U is a nonempty multiset. Let $m = |U|$, and let $u_0 <= u_1 <= \ldots <= u_{m-1}$ be the elements of U in nondecreasing order. If $k > 0$, then define $\text{select}_k(U)$ to be the multiset consisting of the elements $u_0, u_k, u_{2k}, \ldots,$ and u_{jk}, where $j = \lfloor (m-1)/k \rfloor$. Thus, $\text{select}_k(U)$ chooses the smallest element of U and every kth element therafter.

An important role is played by the constants

$$c(m, k) = \left\lfloor \frac{m-1}{k} \right\rfloor + 1$$

where c(m,k) is the number of elements in $\text{select}_k(U)$ when U has m elements. The constant c(n-2t, t) appears as the convergence t of faulty processes, and (2) a constant k, the choice of which depends on t and on whether the algorithm is synchronous or asyncronous. For $k > 0$ and $t >= 0$ define the function fk, t by

$$fk,t(V) = \text{mean}(\text{select}_k(\text{reduce}^t(V))),$$

for all multisets V with $|V| > 2t$. The approximation function for the synchronous protcol with no more than t faulty processes is ft,k. The approximation functilon for the asychronous protocol with no more than t faulty processes is f2t.k.

To show why these functions are appropriate, consider Lemma 5.4 and 5.5.

LEMMA 5.4 is used in verifying the validity condition.

LEEMA 5.5 is applied to determine the rate of convergence of the approximation rounds. The multisets V and W are the multisets of values received by two nonfaulty processes in a given round and U is the multiset of avalues held by nonfaulty processes at the begining of that round. Nonfaulty processes use the appropriate approximation function to choose their values for the next round; the lemma tells us how quickly those values converge.

SECTION 4.5.2 : THE SYNCHRONOUS PROBLEM

Refer to the defination of a synchronous approximate algorithm P given in chapter 2.

We assume that the system acts synchromously, using a reliable communication medium. Each process is able to send messages to all process (including itself), and the sender of each message is identifiable by the receiver.

A configuration consists of a state for each process. An initial configuration consists of an initial state for each process. Let T be any subset of the processes. Refer to chapter 2 for the defination of T-computation.

Assume a fixed small value ϵ- , a fixed number number of process n, and fixed maximum number of faulty processes t.

A synchronous approximation algorithm is said to be t-correct provided that for every subset T of processes with | T | >= n -1, and every T-computation, the following is true:

Every p belongs to T eventually enters a halting state and the following two conditions hold for the values of those halting states:

(a) Agreement:

If two processes in T enter halting states with values r and s, respectively, then | r - s | <= ϵ-.

(b) Validity:

If a process in T enters a halting state with value r, then there exist process in T having x and y as initial values, such that x <= r <= y.

Theorem 5.1 is proved next.

Note that the following strategy would suffice to prove Theorem 5.1. The process could run n executions of a general (unlimited value set) Byzantine Generals algorithm, such as the one in [5_4], in order to obtain common estimates for the initial values of all the process. After this algorithm completes, all processes in T will have the same multiset V of values for all the processes. Then each process halts with value f(v), where f is a predetermined averaging function that is the same for all processes. This algorithm actually achieves exact real - valued agreement, with the required validity condition. However,solution presented below is simpler and more elegant and, moreover, extends directly to the asynchronous case, for which exact agreement is impossible. The algorithm has two additional advantages over using a Byzantine Generals algorithm: It is more resilient than typical Byzantine Generals algorithms, and it can, in some cases, terminate in fewer than t + 1 rounds.

We now present the synchronous approximation algorithm S. First, we describe a nonterminating algorithm, So, and then we discuss how termination is achieved. We assume that n >= 3t + 1.

SYNCHRONOUS APPROXIMATION ALGORITHM SO

At each round, each nonfaulty process p performs the following steps:

1) Process p broadcasts its current value to all processes, including itself.
2) Process p collects all the values sent to it at that round into a multiset V. If p does not receive exactly one correct value from some particular other process (which means, in the sunchronous model, that the other processes faulty), then p simply picks some arbitary default value to represent that process in the multiset. the multiset V, therfore, always contains exactly n values.
3) Process p applies the function ft,t to the multiset V to obtain its new value.

Lemma 5.6 states how the diameter and range of the nofaulty processes' values are affected by each round of algorithm S0.

Part 1 of Lemma 5.6 shows that, at each round, the diameter of the multiset of values held by nonfaulty processes decreases by a factor of c(n - 2t, t), which is at least 2 because n >= 3t +1. Thus, the diameter of the multiset of values held by nonfaulty processes eventually decreases to ϵ- or less. In addition, repeated application of part 2 of Leema 5.6 shows that, at each round h >= 1, the values held by nonfaulty processes immediately before round h are all in the range of the initial values of nonfaulty processes.

It is now easy to see why the function ft,t is appropriate for the synchronous algorithm. Since a correct process can receive at most t values in a round from faulty processes, t-fold application of reduce is sufficient to ensure that extreme values from faulty process are discarded. Thus, the second subscript of f is t. Also, if p and q are correct processes that receive multisets V and W, respectively, in a round, then t is the maximum number of values that can in V - W. Application of select t to the reduced multisets is therefore sufficient to obtain convergence, and the first subscript of f is also t.

Algorilthm S0 is not a correct synchronous approximation algorithm, for, as stated, it never teminates. We modify S0 to obtain a terminating algorithm S, as follows.

TERMINATING ALGORITHM S

At the first round, each nonfaulty process uses the range of all the values it has received at that round to compute a round number at which it is sure that the values of any two nonfaulty processes will be at most ϵ- part. Each process can do this because it knows the value of ϵ-, the guranteed rate of convergence, and, furthermore, it knows that the range of values it receives on the first round that must be executed (including the first round) is given by $[\log_c (A(V) / \epsilon-)]$, where v is the multiset of values received in the first round, and $c = c(n-2t,t)$.

In general, different processes might compute different round different numbers. Any process that reaches its computed round simply halts and sends its value out with a special halting tag. When any process, say p, receives a value with a halting tag, it knows it has to use the enclosed value not only for the designated round, but also for the future rounds (until p itself decides to halt, on the basis of p's own computed round number). Although nonfaulty processes might compute different round numbers, it is clear that the smallest such estimate is correct. Thus, at the time the first nonfaulty process halts, the range is already sufficiently small. At subsequent rounds, the range of values of nofaulty processes is never increased, although we can no longer guarantee that it decreases. Lemma 5.7 makes these ideas more precise.

SYNCHRONOUS APPROXIMATION ALGORITHM S

```
Round 1 (First Approximation Round):
        Input v;
                V <--- SynchExchange(v);
                v <--- ft,t(V);
                H <--- [log_c ( &(V) / E- ) ], where c = c(n-2t,t)

Round h (2 <= h <= H )  (Approximation Rounds):
                V <--- SynchExchange(v);
                v <--- ft,t(V).

Round  H + 1  (Termination Round):
                Broadcast(<v,halted>);
                Output v.

{ End of Main Algorithm }

        Subroutine SynchExchange(v);
                Broadcast(v);
                Collect n responses;
                    Fill in values for halted processes.
                    Fill in default values, if necessary.
                Return the multiset of responses.
        { End of subroutine }

{ End of Algorithm S }
```

To show that S is a correct synchronous approximation algorithm, we must show that all processes terminate, and that the agreement and validity conditions are satisfied. It is clear that all processes terminate. Consider the agreement property. At the first round at which some nonfaulty process halts, it is already the case that the values off all nonfaulty processes are within ϵ- of each other. By Leema 5.7, this diameter never increases at subsequent rounds, so the final values of all the nonfaulty processes are also within ϵ- of each other. The validity property also follows from repeated application of Leema 7. This completes the proof of Theorem 5.1. Q.E.D.

As a final note, observe that algorithm S can be modified so that a process need not always wait for its computed round to arrive before halting : It can halt after it receives halting tags from at least t+1 other processes.

SECTION 4.5.3 : THE ASYNCHRONOUS PROBLEM

In this section we reformulate the problem in an asynchronous model adapted from the one in [5_9]. In an asynchrnous approximation algorithm, we assume that processes have states as before, but now the operation of the processes is described by a transistion fuction that in one step tries to receive a message, gets back either "null" or an actual message, and on the basis of the message, changes state and sends out a finite number of other messages. Nonfaulty processes always follows the algorithm. Faulty processes on the other hand, are constrained so that their steps at least follow the standard form - in each step they try to receive a message, as nonfaulty processes do. However, they can change state arbitrariy (not necessarily according to the given algorithm) and send out any finite set of messages, (not necessarily the ones specified by the algorithm).

Refer to defination of T-computation for asynchronous approximation algorithm given in chapter 2.

An asynchronous approximation algorithm is said to be t-correct provided that for every subset T of processes with | T | >= n - t and every T-computation, every process in T eventually halts, and the same agreement and validity conditions hold as for the sunchronous case.

It seems simplest here to insist on the standard form being followed by all processes. The requirement that faulty processes keep taking steps until they enter halting states at any time they wish. Similarly, the requirement that faulty processes continue trying to receive messages is not a restriction, since they are free to do whtever they like with the messages received. Finally, the requirement that faulty processes only send finitely many messages at each step is need4ed so that faulty processes are unable to flood the message system, preventing messages from other processes from getting through.

We assume that processes take steps at completely arbitary rates, so that there is no way (in finite time) of distinguishing a faulty process from one that is simply slow in responding. Also, we assume that the message system takes arbitary lengths of time to deliver messages and delivers messges and delivers them in arbitary order.

Theorem 5.2 is proved in the following text.

We describe the asynchronous approximation algorithm. As in the synchronous case, first we describe a nonterminating algorithm A0, in which processes compute better and better approximations, and we then modify A0 to produce a terminating algorithm A. Assume that n >= 5t + 1.

ASYNCHRONOUS APPROXIMATION ALGORITHM A0

At round h, each nonfaulty process p performs the following steps:

1) Process p labels its current value with the current round number h, and then broadcasts this labeled value to all processes, including itself.
2) Process p waits to receive exactly n-t round h values and collects these values into a multiset V. Since there can at most t faulty processes, process p will eventually receive at least n - t round h values. Note that, in contrast to the synchronous case, process p does not choose any default values.
3) Process p applies the function f2t,t to the multiset V to contain its new value.

By analogy with Lemma 5.6, we have Lemma 5.8 which states the convergence properties of the above algorithm.

Part 1 of Leema 5.8 shows that, at each round, the diameter of the multiset of values of nonfaulty process decreases by a factor of c(n-3t,2t), which is at least 2 becauise n>=5t+1. thus, the diameter of the multiset of values held by nonfaulty processes eventually decreases to C- or less. In adition, repeated application of part 2 of Lemma 5.8 shows that, at each round h>=1, the valuaes by nonfaulty process immediately before round h are all in the range of the intial values of nonfaulty processes.

We can now see why f2t,t is the appropriate approximation function for the synchronous algorithm. The second subscript is t because, as in the synchronous case, that is the maximum number of values a correct process can receive in a round that are not values of correct processes. The first subscript is 2t becasiue if the correct processes p and q receive multisets V and W, respectively, in a round then 2t is the maximum number of values that can be in V-W(t faulty values, plus t nonfaulty values received by p but not by q).

The only remaining problem is termination. We cannot use the same technique that we used in the synchronous algorithm, because a process cannot wait until it hears from all other processes, and thus it cannot obtain an estimate of the range of the initial values of the nonfaulty processes. We solve this problem by adding an initialization round at the begining of the algorithm. In this initialization round(round 0), each nonfaulty process p performs the following steps:

<u>**Initialization Round for Asynchrnous Approximation Algorithm A**</u>

1) Process p labels its intial value with the round number 0 and then broadcasts this labeled value to all processes, including itself.
2) Process p waits to receive exactly n - t round 0 values and collects these values into a multiset Vp.
3) Process p chooses an arbitary element of $A(\text{reduce}^{2t}(Vp))$ (say $\text{mean}(\text{reduce}^{2t}(Vp))$) as its initial value for use in round 1. Let xp be this chosen value.

Suppose that p and q are arbitary nonfaulty processes. Then, since $| Vp | > 4t$ and $| Vp - Vq | <= 2t$, it follows that Vp and Vq satisfy the hypotheses for the multisets V and U, respectively, in Leema 5.3 (with $j = 2t$). An application of this result shows that, for any nonfaulty processes p and q it is the case that xp belongs to A(Vq). That is, the value xp computed by process p as the result of the initialization round is contained in the range of all values received by process q in the initialization round. Since each nonfaulty process q knows

(1) that its range A(Vq) contains all the round 1 values xp for nonfaulty processes p,

(2) the value of C-, and

(3) the guaranteed rate of convergence, it can compute, before the neginilng of round 1, and round number at which it is sure that the values of any two nonfaulty process will be at most C- part.

The total number of rounds that must be executed by a process, not including the initilization round, is $\lceil \log_c (A(V) / C-) \rceil$ where V is the multiset received in the initialization round and c = (n - 3t, 2t).

As in the sunchronous case, different process will calculate different round numbers at which they would like to halt. The same modification, of sending a value out with a special halting tag, works here as well. We obtain lemma 5.9 which is analogous to Leema 5.7.

ASYNCHRONOUS APPROXIMATION ALGORITHM A

Round 1 (First Approximation Round):

```
Input v;
        V <--- SynchExchange(v);
        v <--- ft,t(V);
        H <--- [log_c ( &(V) / C- ) ], where c = c(n-2t,t)
```

```
ound h (2 <= h <= H )  (Approximation Rounds):
                 V <--- SynchExchange(v);
                 v <--- ft,t(V).

ound  H + 1  (Termination Round):
                 Broadcast(<v,halted>);
                 Output v.

 End of Main Algorithm }

         Subroutine SynchExchange(v);
                    Broadcast(v);
                    Collect n responses;
                        Fill in values for halted processes.
                        Fill in default values, if necessary.
                    Return the multiset of responses.
         { End of subroutine }

 End of Algorithm S }
```

lgorithm A is summarized above. The remainder of the proof of heorem 5.2 is analogous to that of Theorem 5.1.

SECTION 4.5.4 : RESILENCE

The algorithms presented in this above have some intersting resilence properties, stronger than those usually claimed for Byzantine agreement algorithms. So far, we have only claimed that the algorithms are resilent to t different processes exhibiting Byzantine faults during the eintre course of the algorithm. However, we can claim more for situations where processes fail and recover repeatedly. Our algorithms actually support resilence to any t Byzantine faulty processes at a time (under suitable definitions of faultiness at a particular time); the total number of faulty processes can be much greater that t, since we can allow different processes to be faulty at different times.

We do not give a formal presentation of four resilence properties. Rather, we just give a brief sketch of the main ideas.

First, consider the sunchrnous case. A faulty process is able to recover easily and reintegrate itself into the algorithm. It can reenter the algorithm at any round, just by sending an arbitary value, collecting values and averaging them as usual to get a new value. The process also needs to obtain an estimate of the number of rounds required before termination. It can obtain such an estimate in the reenty round, just as it could in the first round.

asynchronous case is a little more complicated. A faulty
:ess p needs to rejoin the algorithm at some particular
'nchronous) round; however, it must be careful to rejoin at
: round that is not "out of date". That is, in the absence of
.tional failures of p, it must be guaranteed to receive all of
messges for that and subsequent rounds. Process p could not
ıly wait until it received n-t messages for some particular
ıd k, since those messages might have been delivered very late
messages ffor round k+1 might have already been lost.
ever, it suffices for p to send out a "recovery" message, and
it acknowledgements form n - t processes carrying the number
their current round. Process p knows that the t + 1 st
llest of these round numbers plus 1, is an allowable round
ɔer for it to use for reentry.

recovering process is not able to use the same method of
imating a termination round as it did initial. Therefore, it
ns necessary to modify the asynchronous algorithm to enable
overing processes to obtain termination estimates when needed.
easy modification that works is to have every process
gyback its estimate of the number of rounds to termination on
ry message it sends. Then a recovering process can obtain a
estimate just by taking the t+1st smallest of the estimates
receives at the reentry round.

MODULE SIX

In this module we consider protocols for both synchronous and asynchronous models. All the results are based on distributively flipping a coin, which is usable by a significant majority of the processors.

Thus the algorithms presented in this module are based on producing a coin flip that is essentially global. (A global coin flip has a random outcome that is viewed identically by every processor.) We relax the condition that each process's view of the coin must always be identical, and in fact, the coin may even be somewhat biased.

For this module, We define the consensus problem as follows : processor i has a private binary value vi; at the termination of the protocol all processors have agreed on a common Value V; if all vi were equal initially, the final value agreed upon is this common value.

We shall initially consider the following synchronous model. We are given a system of n processors that can communicate through a completely connected network. The processors act synchronously, where at each step each processor can broadcast a message, recieve all incoming messages, and perform some private computation (possibly involving coin tossing). In the absence of failure, any message sent at time i will be recieved at time i+1. As a result, we view the computation as occuring in rounds, each consisting of transmission, reception, and private computation phases.

Again, n will be used to denote the number of processors, and t will denote an upper bound on the number of failures tolerated. Next protocols for achieving consensus in completely connected networks despite omission faults of various types, is presented, which can tolerate up to a constant fraction of the processors failing: that is, for each protocol and fault type there is a constant $ < 1/2, independent of the value of n, such that the protocol can tolerate as many as t=#n omission faults of the given type.

Refer to the defination of weakly global coin, given in chapter 2. The intution behind this definition is that if $\lfloor n/2 \rfloor +t+1$ processors see the same outcome,then a majority of the processors ($\lfloor n/2 \rfloor + 1$) will use this value in the consensus protocol, and reach consensus in a few more rounds. The essence of weakly global coin procedure is to randomly select a temporary leader, and then to use the leader's local coin flip for the given round. After showing how such a coin can be produced in a variety of omission faults models, we then indicate how to use it to achieve consensus.

The design strategy of the protocols in [6_0] reflects a heuristic rule prevalent in distributed protocol design: It should be possible for simpiler alogrithms to defeat weaker adversaries. In the search for provably good alogrithms that are useful in practise, this rule suggests that some complex protocols have simple counterparts in more realistic fault models. In the case studied here the alogrithm against the adaptive adversary is transparent in comparison to the protocol for the Byzantine case that results from the combined work in [6_3] and [6_5].

In section 4.6.1, we consider the various failure models, for the synchronous case.

In section 4.6.2 and 4.6.3, we consider the tw adaptive adversary models.

In section 4.6.4, the asynchronous case is considered and finally in section 4.6.5 we consider algorithm for achieving consensus using a weakly global coin.

SECTION 4.6.1 : FAILURE MODELS

Correctness proofs for fault - tolerent alogrithms have a game-theoretic character. That is, the alogrithms behave appropriately, even when the faults are being caused by an intelligent adversary. The capabilities attributed to this adversary have a profound effect on the design of alogrithms meant to defeat it. Indeed, there are cases in which no alogrithm is capable of defeating sufficiently powerful adversaries [6_14, 6_20].

In Byzantine fault models, the adversary can control the behaviour of some processors, causing them to send arbitrary messages whenever it likes. Such an adversary is extremely powerful, and defeating it seems to require complex and expensive alogrithms. If one is modelling phisical failures (as opposed to intentional attacks), such an adversary may be unrealistically powerful.

Consider the following example. On october 27, 1980, the ARPANET suffered a catastrophic failure as the result of hardware failures in two processors. Two spurious messages were generated that brought down the whole network for a period of several hours. Clearly, the network protocolswere not capable of surviving even a small number of Byzantine faults. Instead of changing the protocols, hardware error-detection was added in the next generation n processors, reducing the likelyhood of repetition of this Byzantine failure to an extremely small probability [6_23]. Rather than implementing protocols to defeat a Byzantine adversary, the network designers effectively choose to weaken the adversary.

The new ARPANET implementation might be best described by an omission fault model. in which processors never send spurious messages, but some messages s may fail to arrive at their destination. The adversary is thus limited to specifying which messages will be delivered to their destination, and which will not. The failure models we consider here are variants of failure by omission.

For deterministic protocols, an adversary, causing failures to produce the worst possible performance, can determine the outcome of a strategy in advance. With randomization, this is no longer possible, so that it may be advantageous for the adversary to decide its strategy adaptively, as random bits are generated and used. Therefore, in modeling the power of the adversary, it is crucial to specify the extent to which the adversary is adaptive, and the information it has available to determine its strategy. We consider three limitations on the adaptiveness of the adversary. Each of these is concerned solely with the communication system that connects the processors, and thus assumes that the processors are themselves non faulty. However, as we elaborate below, the situation in which processors are allowed to fail in a "fail-stop" manner is a special case of one of models considered in [6_0].

MODELS : Limitations on the adaptives of the adversary

Static Faults:

Throughout the life of a system, messages sent by at most t processors fail to reach their destination on time (within the round they are sent). Most previous work on omission fault model has focused on this type of fault. In the traditional fail-stop model, processors fail by halting prematurely, but the communication network always delivers all messages that have been sent. Within this model, definition of the consensus problem is flawed, since we require that all processors agree on a value, and it is hope less to require a faulty processor to do anything. If we relax this requirement to all nonfaulty processors,it is not hard to see that static communication faults include the case of fail-stop processor faults.

Dynamic-Broadcast:

During each round, messages sent by at most t processors fail to reach their destination (but this may happen to a different set of t processors each round). A processor that sends a message that does not reach its destination is said to be erratic. These models are more general than static fault models. They are similar to models studied in [6_21].

Dynamic-Reception:

Each processor receives all but at most t messages sent to it during every round (so that, if all processors are supposed to broadcast every round, each processor receives at least n - t messages). However any two processors may fail to hear from a different set of t others. These models are more general than dynamic-broadcast models, and are similar to the models we use for the asynchronous case.

We present alogrithms for dynamic-broadcast and dynamic-reception models. Because these models are more general than the fail-stop or static models, these alogrithms will work in these cases as well.

In addition to the limitations on the adaptives of the adversary mentioned above, we consider two different limitations on the knowledge available to the adversary in determining its strategy.

MODELS:Limitations on the knowledge available to the adversary

Message-Oblivious:

The adversary's choice of failure, that is, which messages will not be delivered, is independent of the contents of the messages. However, this choice can depend, for example, on the pattern of communication or on the length of messages.Before giving a more precise definition, we first introduce a formal description of a synchronous execution of a protocol in this model.

At round $k + 1$ of a protocol, the prior k rounds of execution can be described in the following way. Consider a layered, directed graph consisting of $k + 1$ vertices for each processor p, (p, i), $i = 1, \ldots, k + 1$, where there is an edge from (p, i) to $(q, i + 1)$ whenever p sends a message to q at round i. A subgraph of this graph represents the messages actually delivered. These graphs will be known as the transmission and reception graphs, and together will be reffered to as the communication pattern. To complete the description of the prior execution, we add labels to the edges of the distribution graph,where the labels correspond to the contents of the messages. We define the ith layer of these graphs to be the subgraphs induced on the vertices with second coordinate i and $i + 1$.

Each processor p's view of the communication pattern consists of the subgraphs of nodes labelled by p, together with the labeled out-edges of those nodes in the transmission graph (the messages p sent), and the in-edges in the reception graph (the messages p received). A protocol for p determines a distribution of a new local state, out-edges and labels for node $(p, k + 1)$, as a function on of p's local state and p's view of the first k layers of the communication pattern, together with p's input value. An adversary determines a distribution of in-edges for the k + 1st layer of the reception graph as a function of the n processor protocols and input values, the first k layers of the communication pattern, and the k + 1st layer of the transmission graph. An adversary is message-oblivious if for any given input vector to the processors, any communication pattern up to round k, and any kth layer of the transmission graph,the probability distribution of the kth layer of the reception graph is independent of the labels of the communication pattern through the first k layers (inclusive).

In [6_6], a weaker probabilistic adversary was considered, called a fair scheduler. At round i, a fair scheduler delivers to processor p a random subset n-t messages out of all messages sent to processor at this round. Furthermmore, set of messages sent to different processors are mutually independent. Bracha and Toueg have demonstrated a constant fraction of failures for executions under fair schedulers.

Message-Dependent:

This model places fewer restrictions on the adversary's knowledge of communication in the network.

The adversary is limited to polynomial resources (time and space), but its choice of failures may depend on the contents of the messages.

Note that these definitions assume that the adversary has full knowledge of the hardware and software running at each processor and of the communication over the network (subject to the limitations above), but does not know the local state of individual processors during execution (which may depend on the outcome of local coin tosses not observed by the adversary). For example, it will be important that discription keys are stored in local memory and are local part part of the local state. We assume that the initial values can be seen by the adversary. For each combination of adaptiveness and knowledge constraints, we present an alogrithm to achieve consensus in constant expected time.

SECTION 4.6.2 : THE MESSAGE-OBLIVIOUS CASE

In this section we show how to toss a weakly global coin in message-oblivious models. For the dynamic-broadcast failure model,the coin will have the property that for each outcome (heads or tails), there is some constant probability of that outcome being received by every processor. For the dynamic-reception failure model, there is some constant probability that for each outcome, at least $\lfloor n/2 \rfloor + t + 1$ processors will receive that outcome, provided t is bounded away from n/4.

The algorithms is perhaps the most natural one. A leader randomly volunteers, and this leader tosses a coin. More precisely, consider the following alogrithms:the procedure LEADER produces a local biased bit where the probabilty of a 1 ("I volunteer") is equal to 1/n; the procedure RANDOM BIT produces a local unbiased bit.

Code for processor P:

```
1. function COIN_TOSS_1 :
2. lp <- LEADER
3. Cp <- RANDOME BIT
4. broadcast (Cp,lp)
5. receive all (C, l) messages
6. if all messages received with l=1 have the same C
7. then COIN_TOSS_1 <- C of these messages
8. else COIN_TOSS_1 <- local coin toss
```

Refer to Theorem 6.1 .

The protocol can also be viewed in the following way. The tossing of the 1/n biased coin is an approach to obtain a distribution where the maximum of n trials is likely to be unique. In this context, the leader is the processor r who tossed the unique maximum. All processors receive the other processor's values, determine the maximum and hence the leader, and choose the unbiased bit of this processor. By choosing other distributions it is easy to see that the probability of a unique leader can be pushed arbitrarily close to 1. In implementing the protocol, this means that it is possible to trade off additional bits transmitted in order to reduce the expected number of rounds to reach consensus. For example, if the leader identification consists of 3 log n unbiased bits binstead of a single bit 1, there is a very high probabilty, >= 1-1 / 2n, that the maximum of n bit-sequences will be unique.

Refer to Theorem 6.2 .

By modifying the protocol, it is possible to significantly strengthen the number of faults tolerated in the dynamic-reception fault model. Before giving this new protocol, we first describe a basic building block that will be useful in several constructions.

SIMULATING DYNAMIC-BROADCASTS WITHIN A DYNAMIC-RECEPTION MODEL

We shall show that three rounds of broadcasting within the synchronous dynamic-broadcast while maintaining the property of message-obliviousness. The simulation consists of one round in which each processor broadcasts the original desired message for dynamic-broadcast (To simplify the discussion, we asume every processor has such a message to send.)In the following two rounds, every processor sends his message plus his view of every other processor's message.

We begin by showing that after executing this protocol, there is a set of at least n - t processors whose message has been relayed to all n processors, assuming that $t < n/2$. This is done by a simple counting argument. Consider the second round of the simulation. We show now that there must be at least one processor p whose second round messages reach t+1 processors. If all processors reach no more than t, then m. the total number of messages successfully y transmitted in the second round, is at most $M <= nt$. But each processor receives at least n - t, so that $n(n - t) <= M$ .Thus we get $n(n - t) <= nt$, contracting the assumption that $t < n/2$. Every processor receives at least n - t messages in each round, so that processor p must have attempted to relay at least this many messages to each processor in round two. Since there are t +1 processors that have been relayed these messages at the end of round two (from p), every processor will be relayed these messages from one of the t+1 processors by the end of round three.

This proves that this three round dynamic-reception simulation gives us the structure of one round of dynamic- broadcast. It is not hard to see that one fewer round of echoing is not sufficient to guarantee the structure of a dynamic-broadcast round.

We now show that message-obliviousness is preserved by this simulation. First notice that the pattern of sending and receiving messages in the simulation itself does not depend on message contents. From the definition of a message oblivious adversary, the ith layer of the reception graph is independent of the labeling of the transmission graph given the pattern of communication up to this point. The analogous statement holds for the i+1st layer, given the layer and the previous pattern of communication. From the definition of conditional probability, we get that the probability of any communication for both the ith and i+1st layers is independent of the previous labelings of the pattern of communication. In this protocol, this implies that the set of at least n-t processors that reach at least t+1 processors two rounds later, is independent of the contents of the messages sent. Once this set reaches t+1 processors the adversary cannot stop the set of messages from reaching all n processors in the next round. Since the set is independent of the contents of the messages sent, the pattern of the successful transmissions in the contents of the messages sent,the pattern of the contents of the messages. Thus we have shown that message-obliviousness is preserved.

The above two-round echoing scheme is a general tool. Applying it for the case of producing a weakly global coin, we get the following modified procedure.

Code for processor P with two-round echoing:

```
1. function COIN_TOSS_2;
2. lp <- LEADER
3. Cp <- RANDOME BIT
4. broadcast (Cp, lp)
5. receive all (C, l) messages
6. broadcast (Cp, lp) and all (C1, l1) pairs received
7. receive all compound (C1, l1)...., (Cn, ln) messages
8. broadcast (Cp, lp) and all (C1, l1) pairs received
9. receive all compound (C1, l1)...., (Cn, ln) messages
10. if all messages received with l=1 have the same C
11. then COIN_TOSS_2 <- C of these messages
12. else COIN_TOSS_2 <- local coin toss
```

Although the echoing in this protocol requires a factor of n more bits o be transmitted, it can tolerate up to $t = \lceil n/2 \rceil - 1$ failures and the fraction of processors whose messages reach every one is at least $(n - t) /n$

Summary of above result is given in form of Theorem 6.3.

It is critical to the correctness of this protocol that the adversary's choice of messages delivered each round be independent of the contents of the messages. a stronger adaptive adversary might simply check each message as it is sent ;if the processor is a potential leader (its message is (b,1)), then the adversary blocks the message. This stronger adversary can also be defeated, as long as the contents of the messages are intelligible to him. In this case, any attempt at blocking the leader's message is still an essentially random act,because the adversary cannot understand the messages. This suggests that encryption would be useful tool in designing a protocol that can defeat a more powerful adversary.

SECTION 4.6.3 : THE MESSAGE-DEPENDENT CASE

In this section we show how cryptographic techniques can be used to toss a weakly global coin in the presence of an adaptive adversary using a message-dependent strategy. We prove that if the adversary can block the weakly global coin, then it can break the cryptosystem. Therefore, if we assume that the cryptosystem is secure, and that the adversary is limited to polynomial computing resources, then it cannot prevent consensus within constant expected time.

Let E be a probabilistic encryption scheme that hides one bit [6_15]. We breifly describe the properties that E should possess. Given a natural number h, the security parameter, E maps the 1 at random into a string o in a set O subset of $\{0, 1\}^h$ and maps the bit 0 at random into a string z in a set Z subset of $\{0, 1\}^h$. Given a random string r an element of O U Z, we assume that no polynomial time alogrithm (that is, polynomial in h) can distinguish the case r belongs to Q from r belongs to Z with success probability greater than (1 / 2) + (1 / n^c) for any constant c > 0. On the other hand, there is a polynomial-time algorithm that,given additional secret information, distinguishes between the two cases with probability 1. The scheme E can be based on any trapdoor function [6_23]. In particular, the familar RSA cryptosystem can be used, with o encrypted by E(x), where x is chosen at random among all numbers in Zn with least significant bit 1 [6_1]. (For example,we assume that RSA is hard to invert) It is important to reiterate that the main theorem of this section is based on the following hypothesis:

(*) The encryption function E cannot be inverted in random polynomial time without the secret trapdoor information.

We first make the assumption that all processors use the same public key E whose decryption key that all hold (but to which the adversary has no access)At the end of this section we indicate how this assumption can be removed, at some expense in the number of faults tolerated.

The only modification to the alogrithm of the previous section is to replace the broadcasting of (C,1) (line 4 of the COIN_TOSS1 function) by the broadcasting of (E(C), E(1)).

The modified code is given below:

Code for processor P:

```
1. function COIN_TOSS_3 :
2. lp <- LEADE
3. Cp <- RANDOM BIT
4. broadcast ( E(Cp), E(lp) )
5. receive and decrypt all (C, l) messages
6. if all messages received with l=1 have the same C
7. then COIN_TOSS_3 <- C of these messages
8. else COIN_TOSS_3 <- local coin toss
```

Theorem 6.4 proves that the new protocol is as hard to break as the cryptosystem it uses. This Theorem is based on the assumption that the processors have already agreed on a common public key E. This represents an additional assumption about the initial state of the system. At the cost of a more complex protocol, this asssumption can be avoided.

<u>WEAKLY GLOBAL COINS WITHOUT COMMON PUBLIC KEYS</u>

The problem of key distribution can be solved by having each processor p broadcast its own (individually generated) public key Ep. This is necessagry so that other processors can send encrypted messages to p. Provided $t < n/2$ the algorithms below will flip a weakly global coin.

In the dynamic broadcast model, processors spend an extra initial round broadcasting their public keys. This is done with every toss execution. This guarantees that there are n - t processors whose public keys are knwn to everyone. During the first round of coin toss broadcast, each propcessor encrypts messages with the public key of the receipent, or sens nothing if the recipients public key is not knwn. In a second round of broadcast, all first round messages are broadvast in the clear (unencrypted).

The code follows:

Code for processor P:

```
1. function COIN_TOSS_4:
2. generate and broadcast encryption key Ep.
3. receive all Eq messages
4. lp <-- LEADER
5. Cp <-- RANDOM BIT
6. for each Eq received in step 3 send ( Eq(Cp), Eq(lp) )
7. receive and decrypts all (C, l) messagess
8. broadcast all (C, l) messages
10. if all messages received with l=1 have the same C
11. then COIN_TOSS_4 <-- C of these messages
12. else COIN_TOSS_4 <-- local coin toss
```

As before, consider the case that there is a unique leader chosen during the first round of the coin toss. Since the first round messages are encrypted, an argument exactly analogous to that for Theorem 6.4 establishes that the leadear's messges will be received in step 7 by at least n-t recipients with probability attest 1/2. Since n - t > t, one of these recipients will forward the leader's messages to everyone during the final clear round, steps 8 and 9. Thus COIN_TOSS_4 produces a weakly global coin in the dynamic broadcast model for t < n/2.

In the dynamic reception case, processors run the dynamic broadcast algorilthm under the simulation from Section 4.6.2, running three rounds of broadcasting and forwarding to implement one round of the dynamic broadcast algorlithm. this applies to steps 2-3, 6-7 and 8-9 in the code. One additional change must be made to the dynamic broadcst algorilthm - the simulation asssumes that the same message is brodcast each round. Thus, the vector of encrypted values must be broadcast in step 6;

6'. broadcast < (E1(Cp), E1(1p)) .... (En(Cp), Ep(1p)) >, where (Ei(Cp), Ei(1p)) = "?" if Ei not received.

By invoking the same counting arugment as before there must be at least n-t processors whose encryption keys are transmitted to everyone and these n-t processors will all in turn receive the encrypted messages of at lest n-t processors. Again an argument analogous to the proof of Theorem 4 shows that when there is a single leader, there is a constant probability that will be one of the latter n-t processors. since n-t>t, the leader's message will then be successfully forwarded to all the processors in t he ensuing clear rounds. This is summarized as Theorem 6.5

SECTION 4.6.4 : THE ASYNCHRONOUS CASE

In the section we abandon the assumption that processor run in synchronous rounds. Processors may run arbitrarily fast or slow, and messages may arrive out of order, or take arbitrarily long to arrive even in the absence of failures. We make the following assumption about the nature of failures in the asynchronous model.

Refer to the defination of asynchronous failure. The definition implies that if m messages are sent by distinct processors to the same processorp, then p eventually receives at least m-t of those messages.

We consider two failure models for the asynchronous case, the asynchronous message-oblivious and asynchronous message-dependent model. These both assume the asynchronous failure assumption, adding, respectively, the message oblivous and message-dependent limitations from the synchronous case. In these models, the adversary has full control of the order and timming of arriving messages and of the rates of internal clociks, and is therfore more powerful than in the synchronous case. the adversary is limited in only two ways. The constaints of the faulure assumption require it to eventually delilver enough messages and the message oblivious and message depend limitations restrict the information it may use determine its strategy.

Message-Oblivious:

The adversary's order of events (and, in the particular, choice of delayed and underlivered messages) is independent of the contents of the messges. Before giving a more precise definition, we first introduce a formal description of an asynchronous execution of a protocol. This definition is taken from Fischer et al. [6_14]. An execution is a sequence of events that can be applied, in that order starting from the initial configuration of the system. An even(m, p) is the receipt of a message m that is either the empty message or is from processor p's message buffer (that is, a message that was previously sent to p and not received yet). As in the synchronous case, each processor's protocols determine, upon the receipt of a message, a distribution of actions(the new local state and up to n message sent). These message are then placed in the addressees' message buffers. The adversary determines, as a function of the protocols, the input vector and the asyncrhous execution, a distribution over the set of possible next events. An adversary is message-oblivious if for any given set of protocols(including the input vector to the processors) and any past execution(specified by events (EV1, Ev2,

is independent of the message contents of nonempty messages of the first k events.

Message-Dependent:

The adversary is limited to polynomial resources (time and space, but its choice of failures may depend on the contents of the messages. In general defining the notion of time for an asynchronous system is not a simple matter (see [6_2] and [6_13]). However the protocols we are using are of a restricted type, in which time is naturally defined. These protocols all consist of alternating broadcast and reception phases. In the broadcast phase a processor sends a message to all n processors. In the reception phase, the processor waits to receive messages from exactly n - t processors. This is followed by a local computation the next broadcast phase, and so on. We assume that processors begin each consensus protocol with the same value in their local round counter. In these algorithms, processor append the current value of the round counter to each message. Each processor counts local rounds, consisting of a broadcasting phase and a reception phase. During the reception phase, the processor waits for exactly n-t messages with the current round number(some of which may already be received and stored locally). For simplicity we assume that extra messages with a given round number are discarded. In general, no processor should wait for more than n - t messages from a given round,since failures may prevent more than this many messages from ever arriving. The definition of local time guarantees that no processor is more than one round a haead of the majorilty of other processors(recall that t<n/2). Of course, the slowest processors could lag far behind.

In spite of the adversary's increased power in the asynchronous case, a two round echoing variant of the syncrhonous algorithm will still gurantee that agreemehnt is reached in constant expected time, provided $t < ((3 - \sqrt{5}) / 2n$ is approximately equal to 0.38n.

Before we give the proof let us first remark on the difficulties arising in the asynchronous versus the synchronous case. One might be tempted to argue that exactly the same proofs work, since "once the coin tosses are hidden (by assumption or by encryption), the adversary cannot know which messages to block and so everything works just as it did in the syncronous case." This naive argument is incorrect because an adversary can, in general infer information about messages from the way that processors who receive these messages react to them. If the reaction of each processor to n-t coin-toss messages is sufficent to infer that a single processor volunteered the adversary can successively deliver different subsets of messags to different processors, implementing a simple elimination procedure to determine the indentity of the leader. The leader's messages can then be held back from the remaining processors until they have finished the coin toss, renderint the leader useless. (Notice that the adversary could not perform such elimation in the synchronous case, where trhe reponse of processors is not observeable until after the end of the round, by which time every processor already received its infoming messages for the current round). To exemplify these notions suppose we deal with a different protocol in which a processor that received n - t messages with round number i, among which a unique message is a leader's message, sends its next message to that leader only (and broadcasts to all n processors otherwise). In such case the identity of the ith round leader can be inferred from the (unlabled) communication pattern alone. Thus a message oblivious adversary can block the leaders messages to all other processor.

It is possible to hide the identity of the leader within the consensus algorithm, by making the communication pattern identity of the leader. However consensus protocols are meant as general purpose tools and it is not possible to anticipate faully the context in which they may be run. Thus once any processor leaves the coin toss or agreement protocol it may behave in an arbitrary way, releasing arbiltraty information to the adversary (such as publishing cryptographic keys). These protocols must ensure that information leaked by the faster processor will not jeopardize correctness by allowing the adversary undue influence over the slower processors. The asunchronous protocols below use the imposed round structure and explicit synchronization rounds to satisfy these requirements.

Specifically in the cast that there is a single leader the identity of the leader is hidden at least until the fastest processor completes the execution of the procotol. If the leader is persuasice the coin has the additional property that the majority value of the coin (i.e. the unique value assumed by $\lfloor n/2 \rfloor +1$ processors) has been determined by this point. This is an important property for asynchrnous coin tosses to have, in particular for this application.

Because of the round structure we impose the leaders messages are only effective if they are among the first n-t messages for that round to arrive at $\lfloor n/2 \rfloor + t + 1$ other processors. For the asynchronous case this will be the definition of a persuasive processor for a given round. These algorithms work by guaranteeing a positive constant probability that a single volunteer will be persuasive. Without making it explicit in the code, we implicitly assume that a round counter is locally maintained and incremented by each processor. When we say that a processor receives n - t messages we mean that it reads messages from its buffer until receiving n - t messsages with its current round number.

The code for the asynchronous, message oblivious model is as follows:

Code for processor P:

```
1. function ASYNCHRONOUS_COIN_TOSS_1:
2. lp <-- LEADER
3. Cp <-- RANDOM BIT
4. broadcast (Cp, lp)
5. receive the first n-t(C, l) messages with current round number
6. broadcast the vector <(C1,l1).....(Cn,ln)> where (C,li)="?" if
        not received
7. receive n-t vectors <(C1,lt),...(Cn,ln)>    with current round
        number
8. receive n-t vectors <(C1, l1),...(Cn, ln)>  where (C1,li) with
        current round number
9. receive n-t vectors <(C1, l1),...(Cn, ln)> with  current round
        number
10. if all the messages received with l=1 have the same C
11. then COIN_TOSS_1 <-- C of these messages
12. else COIN_TOSS_1 <-- local coin toss
```

We call step 4 the coin distribution phase, step 6 the first echoing phase, and step 8 the second echoing phase.

Refer to Theorem 6.6, which proves the main result.

To defeat a message dependent adversary in the asynchrnous case, we make the same alteration as in the synchrnous case, encrypting the random bits.

Code for processor P:

```
1. function ASYNCHRONOUS_COIN_TOSS_2:
2. lp <-- LEADER
3. Ci <-- RANDOM BIT
4. broadcast ( E(Cp), E(lp) )
5. receive the first n-t(E(C), E(l)) messages with current round
       number
6. broadcast the vector <E(C1,l1).....(ECn,ln)> where (C,li)="?"
       if not received
7. receive n-t vectors <E(C1,lt),...E(Cn,ln)> with  current round
       number
8. receive n-t vectors <E(C1, l1),...E(Cn, ln)> where (C1,Ii) with
       current round number
9. receive n-t vectors <E(C1, l1),...E(Cn, ln)> with current round
       number
10. if all the messages received with l = 1 have the same C
11. then COIN_TOSS_2 <-- C of these messages
12. else COIN_TOSS_2 <-- local coin toss
```

Refer to theorem 6.7. and 6.8 .

SECTION 4.6.5 : USING A WEAKLY GLOBAL COIN

In this section we present an agreement algorithm that can be implemented using a weakly global coin. For simplicity of presentation the algorithm given here is binary (reaching agreement on one bit), and is basically a modification of those in [6_4] and [6_6].

We begin with an informal description of the algorithm. The algorithm is organized as a series of epochs of message exchange. Each epoch consists of several rounds. The round structure is provided automatically in the synchronous model. In the asychronous models, the round structure is imposed locally by each processor, as was discussed earlier. In this case,reaching consensus in "constant expected time" means that each processor will complete the protocol within a constant expected number of local rounds.

We describe the algorithm for the processor P. (All processors run the same code). Epoch and round numbers are always the first two components of each message. The variable CURRENT holds the value that processor p currently favors as the answer of the agreement algorithm. At the start of the algorithm CURRENT is set to processor P's input value. In the first round of each epoch, processor P broadcasts CURRENT. Based on the round_1 messages recieved, processor P changes CURRENT. If it sees at least $\lfloor n/2 \rfloor$ +1 round-1 messages for some particular value, then it assign that value to CURRENT; otherwise, it assigns the distinguished value "?" to CURRENT. In the second round of each epoch, processor P broadcasts the new CURRENT. This is followed by a synchronization round, in which all processors broadcast waiting messages,then wait until n-t such messages are recieved. the guarantees that at least n-t processors have finished the previous round before the fastest processor leaves this round. Next, the COIN TOSS subroutine is run . (O course, in an asynchronous model this statement is a bit imprecise, since the subroutine is first initiated at the point that the fastest processor reaches the subroutine call.) Based on the round-2 messages received,processor P either changes CURRENT again, or decides on an answer and exits the algorithm at the end of next the epoch. Let ANS be the most frequent value (other than "?") in round-2 messages recieved by P. Let NUM be the number of such messages. There are three cases depending on the value of NUM. If NUM >= $\lfloor n/2 \rfloor$ NUM >= 1, then processor P assigns the value ANS to the variable CURRENT and continues the algorithm. If NUM = 0, then processor P assigns the result of the coin toss to the variable CURRENT, and continues the algorithm.

Code for processor P:

```
1.  procedure AGREEMENT(INPUT):
2.  CURRENT <- INPUT
3.  TERM.NEXT <- "OFF"
4.  for e <- 1 to INFINITY do
5.  broadcast(e, 1, CURRENT)
6.  receive (e, 1, *) messages
7.  if for some v there are >= $\lfloor n/2 \rfloor$ + 1 messages(e,1,v)
8.     then CURRENT <- v
9.     else CURRENT <- "?"
10. broadcast(e, 2, CURRENT)
```

```
11. receive(e, 2, *) messages
12. if  there exists v not equal to "?" such that
              (e, 2, v) was recieved
13.     then ANS <- the value v not equal to "?" such that
                      (e, 2, v)  messages are most frequent
14.     else ANS is undefined
15. NUM <- number of occurences of (e, 2, ANS) messages
16. broadcast(e, 3, "waiting")
17. receive (e, 3, "waiting") messages
18. COIN <- COIN_TOSS
19. if TERM.NEXT = "ON" then terminate
20. if NUM >= |_ n/2 _| + 1 then decides ANS. set CURRENT <- ANS
                                and TERM.NEXT <- "ON"
21.     else if NUM >= 1
22.             then CURRENT <- ANS
23.             else CURRENT <- COIN
```

We make several remarks about the algorithm. COIN TOSS, depending on the fault model, is one of the protocols described earlier for producing a weakly global coin. In message descriptions, "*" is a wild-card character that matches anything. Notice that processor has decided, it participates in the protocol for another epoch. Although not explicitly given in the code, during this extra epoch the processor ignors all "receive" commands, since otherwise it may be left waiting for messages from processors that have already terminated. The extra epoch is needed because, once the first processor decidesand terminates, the other processors may not decide until the next epoch (as we argue below). The extra broadcast by decided processors are solely to ensure that these "trady" processors recieve a sufficient number of messages during each round of that epoch. (Recall that in the asynchronous fault models, processors must wait for n-1 message during each reception.)

If the input values are sufficiently biased towards a particular value, the protocol will reach agreement in one epoch. If this is not the case, the protocol uses the weakly global COIN TOSS function to prevent the system (abetted by the adversary) from "hovering" at an indeterminate point indefenitely. With each call to COIN TOSS, there is a constant probably that the outcome will biase the system sufficiently to reach agreement quickly. Thus, agreement will be reached in constant expected time.
Define value as legal. input to the algorithm either 0 or 1. Specially , "?" is not a value.
Lemma 6.9 is used in proving the desired properties of the agreement algorithm.

Theorem 6.10 will establish that this algorithm never produces conflicting decisions and that in each epoch there is at least one coin-toss value that will lead to termination of the algorithm.

Consider theorem 6.10, with reference to the following key notations.

The value ANS is critical in the analysis of the protocol. At any instant of an execution of the protocol, an epoch e is bivalent if for both v = 0 and v= 1 there exists an execution of the protocol that continues from the that instantaneous position, for which there exists a processor that has ANS value in epoch e equal to v. Furthermore, let ke be the number of processors that have not determined whether ANS is 0 , 1 or undefined for epoch e at the point that the fastest processor begins the coin-toss for each epoch e. Note that in all the syncronous models discussed, k = 0 at the point that the COIN TOSS protocol is executed in round e. This may not be the case in the asynchronous cases, where the epoch may still be bivalent at the point when the fastest processor initiates the execution of COIN TOSS for that epoch. However, the round of "waiting" messages ensures that at the point when the COIN TOSS is first initiated, ke is at most t (since the fastest processor must have received n - t "waiting" messages in order to continue, and these processors have already executed through step 16. Note that if an epoch is bivalent, then any processor that has already determined ANS at this point has ANS = "undefined".

All of the variants of the coin-toss procedure that we have considered take a constant number of rounds. Combining Theorem 6.10 with the various versions of the coin-toss procedure, we get Theorem 6.11 .

It is natural to ask whetherthe number of erratic processors tolerated can be significantly improved. A result of Bracha and Toueg [6_6] shows that no randomized concensus protocol can tolerate more than n/2 fail stop faults in an asynchronous model.

MODULE SEVEN

Two different kinds of Byzantine Agreement for distributed systems with processor faults are defined and compared. The different kinds of byzantine Agreement for distributed systems with processor faults are defined and compared. The first is required when coordinated actions may be performed by each participant at different times. This kind is called Simultaneous Byzantine Agreement (SBA).

This module deals with the number of rounds of message exchange required to reach Byzantine Agreement of either kind (BA). If an algorithm allow its participants to reach Byzantine agreement in every execution in which at most t participants are faulty, then the algorithm is said to tolerate t faults. It is well known that any BA algorithm that tolerates t faults(with $t < n - t$ where n denotes the total numder of processors) must run at least $t + 1$ rounds in some execution. However, it might be supposed that in executions where the number f of actual faults is small compared to t, the numbeer of rounds could be correspondingly small. A corollary of our first result states that (when $t < n - 1$) any algorithm for SBA must run t+1 rounds in some execution where there are no faults. For EBA (with $t < n - 1$), a lower bound of min(t+1,f+2) rounds is proved. Finally, an algorithm for EBA is presented that achieves the lower bound, provided that t is on the order of the total number of processors.

The context for this study is a network of n processors that are able to conduct synchronized rounds of information exchange, each round consisting of message transmission, message receipt and processing. In the following, n will always denote the number of processors. We assume that the network is completely connected and that only processors can fail.

In the Byzantine fault case, no assumption is made about the behavior of faulty processors. During an execution of an algorithm, a processor is said to be correct if it follows the specifications of the algorithm; otherwise, it is said to be faulty.

We assume that the agreement to be reached concerns a single value that is initially given as input to one processor, called the origin. This value is taken from a known set of values. All processors, called are assumed to know when the input is given to the origin. Each processor is to give exactly one output value after some number of rounds of information exchange with the other participating processors.

Refer to chapter 2 for defination of eventual agreement and simultaneous agreement.

When no assumption is made about the behaviour of the faulty processors, we modify the term agreement with the adjective Byzantine. Thus, we have the terms eventual Byzantine agreement (EBA) and simultaneous Byzantine agreement (SBA). A protocol or algorithm guarantees (Byzantine) agreement in some set of executions if, in each execution of the set, all correct processors reach a (Byzantine)agreement.

Note that a processor may give its output in one round and also continue to send messages to other processors in that and subsequent rounds. In this case, the processor has not finished all rounds of message exchange required by its algorithm when it gives its output. A processor is said to have stopped in round r, if it has given its output by round r+1, and otherwise sends no messages in any round after r. In an execution of an algorithm for reaching agreement, we count the number of rounds between initial input and final stopping of all correct processors as the number of rounds required by the algorithm.

If an algorithm allows its participants to reach Byzantine agreement in every execution in which at most t participants are faulty, then the algorithm is said to tolerate t faults. Here, we investigate the number of rounds required to reach agreement as a function of the number of actual faults and the number faults to be tolerated.

Suppose A' is an algorithm that tolerate N-2 faults, requiring a maximum of k rounds. Let A' be the algorithm obtained by modifying A so that, no matter what happens, each processor stops after k rounds, the origin always gives as output its input value, and each other processor gives as output the value A would give, if any, or a default value, otherwise. Inspection of the definition of agreement shows that A' tolerates any number of faults. Hence, we assume $t > n - 1$, unless otherwise indicated. The initial work of peaseet al. [7_21] showed that agreement in the presence of upto t faults could be reached by round $t + 1$, provided the number of processors was sufficiently large. Later, $t + 1$ was shown to be a lower bound on the number of rounds required in the worst case [7_3, 7_9, 7_15]. A natural question arises from this worst case bound: Can an algorithm for agreement be constructed to handle up to t faults so that whenever the number f of actual faults is smaller than t, the number of rounds required to reach aggrement is smaller than $t + 1$? Section 4.7.1 and Section 4.7.2 present lower bounds for this problem.

Later, Dwork and Moses extended this bound (first published in [7_12]) by studying a closely related problem in which each processor has an input [7_13]. A function from the set of faulty processors to integers that gives the round number at which each processor failed is called a pattern. Dwork and Moses give a lower bound on the number of rounds required for their problem as a function of the pattern. Their bound is easily shown to be a bound for the problem as well by choosing the worst pattern.

The (t + 1)-lower bound and also that of Dwork and Moses-hold when the set of faults to be tolerated is restricted to a very simple type of fault called a crash fault. When a processor suffers a crash fault, it sends a subset of messages it is specified to send in one round and simply ceases to operate from then on. However, Theorem 7.2.1 even holds if the faulty behavior is further restricted to a class of Faults called orderly crash faults.

In section 4.7.1, the lower bound for SBA is shown and proved. In this section, we count the number of rounds of information exchange required to complete the actions specified by the protocol, not the number of rounds required for all incorrect processors to have produced an output value. Since giving its output early cannot help a processors to stop earlier, we assume that a processor saves its output until the round after it last sends a message to another processor. This assumption is a notational convenience and is made without loss of generality. It is easy to convert any simultaneous agreement algorithm to one in which all correct processors stop before they give their outputs and outputs are given no later than in the unconverted algorithm. It is easy to convert an eventual agreement algorithm so that one round after every correct processor knows its output value, every correct processor has stopped.

Extending the proof method of Section 4.7.1 we show in Section 4.7.2 that EBA requires at least min(f+2, t+1) rounds. Our proof works only for crash faults and we do not know how to prove this result for orderly crash faults.

Finally Section 4.7.3, presents an algorithm for EBA that achieves the lower bound, provided $n > \max(4t, 2t^2 - 2t + 2)$. This algorithm does not depend on any authentication protocol. It requires min(f + 2, t+1) rounds to reach EBA using a polynomial (in both n and t) number of bits of information exchange. Previous early stopping EBA algorithms did not achieve the lower bound but did work for n> 3t. Refer to [7_8], [7_24] and [7_1].

MODEL FOR EXECUTION OF AN AGREEMENT ALGORITHM

This model is used in both lower-bound proofs and in the presentation of the algorithm. It is similar to the one previously given by Dolev and Strong [7_11]. The formal framework represents a round of an execution as a directed graph with labeled edges and nodes and as follows.

Let V denote a set of possible values (including the value 0 and 1) and let MSG denote a set of possible messages. A history is an infinite sequence of rounds. Each round consists of a directed labeled graph with nodes corresponding to a set p of n participating processors, together with special source and sink nodes (that are not in p).There is an edge corresponding to every ordered pair of nodes. Each edge is labeled by an element of Msg (the message sent), an element of V (a value), or an empty label (indicating no message). For notational cnvenience, each history begins with round 0, in which the edge coming from the source outside p to the origin is labeled with the input value from V. All other edge have the empty label at round 0. At any subsequent round, any node may have the edge from it to the sink node outside p labeled with its output value. During this round and subsequently all other edges from this node, carry the empty label. If node p has such an edge to the sink at round k, then p has stopped (information exchange) at round k-1 and its output value is the value on the edge to the sink.

Messages (labels) on edge directed toward p in round k are said to be received by p at round k. Likewise, message directed from p in round i are said tom be sent by p at round k. If H is a history we write pH for the view of H according to p which consists of the sequence of subgraphs of the rounds of H that have all the labeled nodes but only the edges that are adjacent to p. We also write Hk and pHk for the initial sequence of H from its beginning throygh round K and its view according to p, repectively.

A protocol (or algorithm) A takes as input an initial subsequence off a view of a history according to a processor and produces an ordered set of labeled edges directed from that processor for the next round. Let U(A, t) be the set off all histories on a fixed set of processors in which all correct processors follow a and in which at most t processors fail to follow A. (In each section, were strict U(A, t) to histories that have only failure of certain types. Also, we write U for U(A< t)), each when the arguments A and t are clear from the context.)

A t resilient agreement algorithm A such that in each history of U(A, t), each correct processor stops in some round and the processors reach agreement.

Note that a history includes the names of the processors, and the view of a history according to one processor is assumed to include the names of all its neighbours (in the completely connected network), whether they have sent it messages or not. Thus, an agreement algorithm need not be uniform and the actions it prescribes can depend on the name of the processor acting and on the names of its targets.

SECTION 4.7.1 : THE LOWER BOUND FOR SBA

In section this section and the following section, we restrict attention to histories in which the only way a processor can fail to follow its algorithm is to fail to send some or all of its prescribed messages in one round and remain silent the read after. This is the notion of a crash failure and is a close relative of the notion of a "fail-stop processor" [7_22]. Note that in the round in which a processor has a crash failure, it may send any message at any subsequent round.

In proving the lower bound in this section, we further restrict the failure mode to orderly crash failures.

A processor fails during the first round in which it does not send all messages required by algorithm A. A processor that fails in round r, sends no messages in each succeeding round.

Our lower-bound proofs are based on establishing certain equivalences among histories. Let A be an agreement algorithm that guarantees SBA in the presence of at most t orderly crash faults. Let p be a fixed set of n processors. Recall that U(A,t) is the set of histories with only orderly crash faults in which algorithm A is employed by all correct processors, and the number of faulty processors does not exceed t.We introduce two equivalence relations on the set U(A,t). These equivalences are also defined for the set of K round initial sequences of such histories, for any K. These are witness equivalence and output equivalence and are defined in chapter 2. Also refer to the defination of serial history, pattern, subpattern, conservative extension and silencing given in chapter 2.

Note that each history of U in which there are no faults is a serial history.

The uniqueness od silencing is guaranteed because conditions (1) and (3) completely determine the behavior of p. For the remaining processors, observe that (2) forces all processors that are correct in Hk to follow a in all subsequent rounds and processors faulty by round k cannot send any messages after round K(we have restricted to crash failures). If history H has processors other than P that fail after round K,then H'resembles the conservative extensionof Hk on those processors because they do not fail in H'However, the silencing of P at K is not necessarily the conservative extension of its K round initial sequence because A may not call for P to send any messages in round K, but it might call for p to send messages later. Since we want p to remain silent from round k on, we must allow for the possibility that p fails in some round after round k. Note that if additng p to set of aulty processors of H does not raise its cardinality above it, then the silencing of p at round k of H is in U.

Refer to the defination of a candidate given in chapter 2.
Finally the main result given as Theoren 7.2.1 . This theorem is proved in the following text.

We base the proof on a sequence of lemmas that contain ideas from several previous related proofs [7_8, 7_11, 7_12]. Suppose for the rest of this section that algorithm A guarantees SBA for each history with at most t orderly crash faults. Assume that there is a serial history H in which reaches SBA in fewer than min(n - 1, t + 1) rounds. If t > n - 2, then A guarantees SBA for each history with at most t' = n - 2 orderly crash faults and reaches SBA in H in fewer than n - 1 = t' + 1 rounds. Thus, a counterexample with t > n - 2 would provide a counterexample with t' = n - 2. Hence, we assume (without loss of generality) that n is at least t+2.

Refer to lemma 7.2.2 .

In the rest of the proof, we show how to alter serial histories in a way that preserves witness equivalence, but changes the number of faults and the place of their occurrence. In any history H of U in which p fails to follow algorithm A, there is a first message specified by A that p fails to send. Also in any round of H in U in which p sends any messages, there is a last message sent by p (in the order specified by A). We call an outedge e of p in a round of a history H significant if algorithm A specifies a message to travel over that edge and this message is either the last message sent by p in this round of history or the first message specified by in the entire history that p fails to send. Since we only orderly crash faults, the message on any significant edge is either correct or absent but not both. We show how to alter the states of messages on selected edges from absent to correct, or vice versa, producing witness equivalent initial sequences of histories and eventually producing a desired result. In particular, we are able to correct any faulty processor or cause any processor to fail in any round that does not violate the requirement that the resulting history be serial.

To finish the proof of Theorem 7.2.1 consider the assumed history H, in which A reaces SBA in t or fewer rounds. Let v be the output value of the correct processors in H, let v' be a value different from v, and let J be the fault free (serial) history with v'. By the agreement condition, all processors have to output v' in J. On the other hand, by Lemma 7.2.4 Ht and Jt are witness equivalent. By Lemma 7.2.2, H and J are output equivalent. This means that in J all processors had to outputd v, a contradiction.

SECTION 4.7.2: THE LOWER BOUND FOR EBA

Next, we consider the question of early stopping for EBA and prove a lower bound similar to through stronger that the one in [7_11]. In this section, we restrict attention to histories in which all failures are crash failures. Let A be a t-resilent agreement algorithm that is supposed to guarantee EBA in U(A, t). Note that U(A,t) has a different defination in this section; faults in histories of U(A,t) may be crash faults rather than the orderly crash faults of Section 4.7.1. When we refer to a conservative extension in this section, we mean a history defined as in the previous section but with respect to the current U.

Refer to chapter 2 for defination of critical history.

For this section we require versions of the notions of serial and candidate that are parameterized by f, that is f-serial and f-candidate (Refer to chapter 2).

Theoren 7.3.1 gives the main result and proof of the theorem follows.

As we argued in the proof of Theorem 7.2.1 a counterexample with $t > n - 2$ would provide a counterexample with $t = n - 2$. Thus we assume (without loss of generality) that $t < n - t$. Suppose that algorithm A reaches EBA within min(t, f + 1) rounds in every history of U with at most f faults.

First we give a straignghtforward derivation of a contradiction in the case f = 0. Assume A is a t-resilence agreement algorithm that uses only min(t, 1) rounds to reach EBA in any history of U wiith no faults. If t = 0 then processors send no messages to other processors: othewise when there are no faults, processors send messages to other processors only in round 1 and all processors send messages to other processors only in round 1 and all processors give the input value as output in round 2. Let H0 be the preliminary round that gives input 0 to the origin and let H be its conservative extension. Each correct processor of H must give output 0 in round min(r + 1, 2). Let K0 be the preliminary round that gives input 1 to orgin and let k be its conservative extension. Each correct processor of K must give output 1 in round min(t + 1, 2). In at least one of H and K the origin must send at least one message in round 1, for otherwise any processor except the origin would have identical views in the two histories. Thus, t must be greater than 0. Without loss of generality, assume that th origin sends a message to processor p in round 1 of H.

Let J1 be identical to H1 except that the origin fails in J1 after sending only its message to p and let J be the conservative extension of J1. (If the origin sends only one message in round 1 of H, then let J=H). Then J has at most one creash fault and is a history in U. Now pH1=pJ1 so p gives output 0 in round 2 of both H and J. Thus, any correct processor in J must eventually give output 0. Since t>0 and n-1>t, we have n.2. Hence there is a processor q that is neither the origin nor p. If the origin sent no message to q in round 1 of K, then we would have nor p. If the origin sent to message to q in round 1 of k, then we would have qK1=qJ1. But q gives output 1 in round 2 of K and q gives output 0 in some round of J. Therefore the origin must send a message to q in round 1 of K. Let L1 be identical to K1 except that the origin fails in round 1 by sending only its message to p (if any) and let L be the conservative extension of L1. Then L has one crash fault and is a history in U Since pK1 = pL1, p gives output 1 in round 2 of K and L, so any correct processor of L must eventually give output 1. Since p sends no messages to other processors after round 1 in any of the histories H, J, K, and L, we have qJ = qL. But this contradicts the fact that q must output 0 in J and 1 in L.

Now we assume f <= 1. Since we assume n - 1 > t, there are at least two correct processors in any history of U. In any history of U with at most f faults there can be no critical edge in round min(t, f + 1) because all correct processors have stopped by round min(t, f + 1) (giving their outputs by min(t + 1, f + 2)) and changing a value over any single edge cannot affect the output of more than a single correct processor. We first show that in any f-serial hstory there is no critical edge in round f from a processor that is an f candidate in round. Then we show that all f-serial histories, including all histories with no faults, are output equivalent. As in the proof of Theorem 7.2.1, we then argue that histories with distinct inputs and no faults must have the same outputs contradicting part (ii) of the definition of agreement. Implies contradiction, Thus the proof. (refer to Lemma 7.3.2 and 7.3.3).

SECTION 4.7.3: THE EBA ALGORITHM

In this section, we describe an algorithm for eventual Byzantine agreement that achievs the lower bounds of the previous sections, provided that n is sufficient larger than t. The algorithm will tolerate up to t Byzantine faults. (We no longer restrict attention to crash faults.) The key to understanding this algorithm is the notrion of separation, which will be described more formally below. Informally when a faulty processor sens different information to two sunbsets f correct processors, it seprates one set from another. The algorithm keeps track of two rounds of informtion exchange at a time, so a fault that separates from each other in one round will be discovered by all correct processors in the next round. In order to avoid discovery by all correct processors, a fault may only separate from others a set of the size of the number of unknown potential faults that must be tolreated.

Thus, t faults cannot separate more than t^2 correct processors from other correct processors without at least one of them discovered . The idea behind the algorithm is that when n is larger than max(4t, $2t^2 - 2t + 2$), this algorithm will allow correct processors to obtain the agreement value at the end of any value at the end of any round in which no fault gives itself away and to stop within one additional round.

Recall that we count only the rounds of information exchange among the processors. The preliminary input and final output rounds are only used to simply the description of the algorlithm.

We use the following notation:

P	denotes the set of names of participoating processors.
s	the name of the origin,
X	a symbol not in P and
V	the set of possible input values.

Let 0 be an element of V,
let * a special value not in V (representing "undefined")
and let V' be the union of V and {*}.

To run the algorithm, each processor maintains a data structure consisting of two type of variables; variables containing values from the set V' and variables containilng sets of processor names. For each othe strings s, ps, and pqs, where p and q run over all the elements of P, we associte a variable of the first type. The values stored in these variable will be interpreted as representing information received from the appropriate processors. Thus, for example, the value stored in s will be interpreted as the value sent by the origin of the agreement. The value stored in qs will be intepred as the value q said that q said that s sent to it. Notice that s denotes both the origin and the variable associated with it. The pseudocode of the algorithm uses s only as a variable and not as a name for the origin.

With the string X and with pX, for every p in P, we associate a variable of the second type. Values stored in strings ending in X will be interpreted as representing information received from the processors about faults. Thus, the set stored in X will be a set of processors known to be faulty. Thus, the set stored in X will be a set of processors kknown tobe faulty. The set stored in qX will be the set of processors q claims to be faulty.

We refer to the variable as strings.
Strings ending in s will be initialized to value 0.
Strings ending in X will all be initialilzed to the empty set.

We use the following convention for naming sets of strings:

Let Q be any subset of P, let p be in P, and let R be any name for a set according to this convention. Then
Qs = {qs | q is in Q},
QX = {qX | q is in Q},
pR = {pr |r is in R}, and
QR = {qr |q is in Q and r is in R}

Thus, for example, Ps is the set of strings of length 2 that end with s, and pPs is the set of strings of length 3 that begin with p and end with s.

Here, we introduce a simple one round process that is the heart of many agreement algorithms. We give this process the name ROUND. Each processor executes ROUND during every round from round 3 until it stops. We also introduce a variant of ROUND called ROUND 2 that is executed in round 2 and collects the orignal information in Ps. Round has two functions:

(1) to exchange information on Ps with all other processors to produce values for PPs that are then reduced to values for Ps

(2) to exchange information on X with all other procesor to produce values for PX and to use PPs and PX to discover faults.

It is expexted to operate synchronously with all participating processors sendilng information to all and then receiving information from all. If two processors are correct, it is assumed that their information is correctly exchanged. It uses two auxilliary processors, DETECT and REDUCE, which are defined below.

We assume that a processors sends messages to itself and process them as part of all the messges it receives.

Note that in ROUND 2 each processor sends the value it has stored in s and receives the corresponding values from all processors. It stores the value received from processor p in ps. Thus, ROUND2 has the instruction "RECEIVE ps from each p in P."

The action of each participating processor executing ROUND2 is as follows:

```
ROUND2:  /*for round 2*/
begin;
    SEND s to all processors;
    RECEIVE ps from each p in P;
    (if ps is not received from p then set ps := s)
    (if  Ps does not contain at least n-t identical values
         then  put the origin in X;)
end ROUND2.
```

Note that in ROUND each processor sends the value it has stored in Ps and X to all processors and then receives corresponding values from every processor. The values received for Ps and X from processor p are stored in pPs and pX respectively. Thus ROUND has the instruction, "RECEIVE pPs, pX from each p in P." The action of each participating processor executing ROUND is as given on the next page.

```
ROUND:    /*for rounds after round 2*/

begin;

    SEND Ps, X to all processors;

    RECEIVE pPs, pX from each p in P;

    (if p ius already in X
         then default its values for pPs to 0)

    (if  p is not already in X but it does not send pPs and  pX
         then for each q in p set pqs:=* and leave pX unchanged)

    DETECT;

    for each p and q in P if pqs=*then st pqs :=s;

    REDUCE;

end ROUND
```

A correct processor may put the name of the origin in X during the execution of ROUND2, but only if Ps does not contain n-t identical values so that the origin must be faulty. In later rounds, the process DETECT is the only way correct processors add names to the set of known faulty processors kept in X. DETECT is designed so that correct processors will never add names of correct processors to X, and therefore, at any time the largest possible number of faulty processors that a gilven correct processors has not discovered is t - | X |.

Since correct processors may stop at different times - the difference can be at most one round as will be seen later-one has to take care that a correct processor that has already stopped and therefore does not send messagesw anymore is not considered to be faulty. This is achiveved by first settilng ariable pqs for which no value from p has been reveived to the undefined value"*". If p is not found faulty by DETECT, then pqs will later be set to the actual value of s.

If more than t - | X | processors claim that they have put proccessor q in theilr set of known faulty processors then any correct processor can safelyu put q in X (some other correct processors put q in its X first).

In this algorithm, correct processors send identical data to all participants. A property that will be preserved by REDUCE is that if p,q, and r are correct processors then the value stored in pqs and rqs by any correct processor will be identical. Thus if the multiset of values stored in (P-X) qs does not have at least n-t identical values then q must be faulty.

The action of each participating processor executing DETECT is as given on the next page.

```
DETECT:
begin;
    for each q in P-X;
        if | { p | p is in P - X and q is in pX } | > t - | X |
            or  P - X contains two sets A and B each of
            cardinalilty >= t such  that  Aqs and Bqs both have
            only values in V, but no value occurs in both Aqs
            and Bqs
        then add q to X and default the values of qPs to 0;
    end for each q;
end DETECT.
```

The process REDUCE uses values of PPs to update the value of Ps using a majority vote.Let g be the smallest integer greater than n/2. In order to obtain the new value for string ps, a majority vote is taken over the values of the string pPs.Note that all these values are obtained directly from p. There is no voting by others here on what p said as it is doen by DETECT for q. If p is correct then it sends the same data (Ps) to each participant; all correct participants will have the same value for ps after REDUCE. These values ps determine the further action to be taken by each processor. If correct processors all have the same set Ps, then they behave identically and reach agreement very quickly.

The action of each participating processor executing REDUCE is as follows:

```
REDUCE;
begin;
    for each p in P;
        if pPs has at least g strings with value v
        then ps:=v
        else ps :=0;
    end for each p;
end REDUCE.
```

For the remainder of this section we assume that :

$$n > \max(4t, 2 (t - 1)^2))$$

so that the following properties are true of the majority threshold.

1) $2g > n$;

2) $n - 2t >= g$;

3) $n - t - (t - 1)^2 >= g$.

We use these properties of g to show that undelete faults cannot cause correct processors xz to reach different values for s.

The algorithm will be called EAGREE. It takes a value as input in round 0. If no value is received the string s is left with its initial value 0. We use the existence of a value other than 0 stored ins inround 0 to indicate that the processor executing the code is the origin. All processors execute the same code. If a processor has a value other 0 stored in s at the end of round 0, then it sends that value to all processors in round 1. We assume that no processor except the origin can have a value stored in s other than 0. If the input value is 0, the origin acts just like the other participants and sends nothing. Receiving nothing from the origin in the first round is interpreted as receiving 0 from the origin. This is just a convenince all processors know the name(s) of the origin. This simply allows us to write EAGREE in a uniform way without mentioning explicitly the same of the processors executing the code. Correct processor using EAGREE reach EBA by round min(f + 2, t + 1) At the end of the algorithm the variable s at each correct processor will hold the output value. Note that round 0 and the output round involve no information exchange among the processor and are not counted when we discuss the number of rounds required to reach agreement.

The action of each participating processor executing EAGREE is as given on the next page

```
EAGREE:
begin ;
          i:=0;              /* round 0 - the input round */
          RECEIVE s AS INPUT;
          (if nothing is received leave s unchanged)
          i:=1;              /* round 1 */
          if s is not equal to 0 then SEND s to all processors;
          RECEIVE s;
          (if nothing is received from  the origin,
                                               leave s unchanged)
          do i:=2 to t+1
               if i=2 then ROUND2 else ROUND;
               if Ps has at least g identical values v
                    then s := v;
                    else s := 0;
               if Ps has at least n-t identical; values
                    then leave this do loop;
          end do;
          i=i+1;             /* output for this processor */
          OUTPUT s;
end EAGREE.
```

Recall that $2g > n$ so that this algorithm is well defined.

Refer to Theorem 7.4.1, the proof of the theorem will be provided in the following series of lemmas 7.4.2 to 7.4.9

We say that value v is persistent at round i, if at least g correct processors have stored in s at the end of round i. Recall that a processor is said to stop in round i, if the only action it takes in round i+1 is to output its value. We say that a processor in convicted in round i if it has at least n - t identical values stored in Ps at the end of round i. Note that if a correct processor is convicted in round i, then it stops in round i. Also if a correct processor stops in round i < t + 1, then it is convinced in round i. However, a processor may stop in round t + 1 without being convicted. In this case it gives its value for s as output without having n - t identical values in Ps.

In order to keep any value from becoming persistent in a round, the faults must send distinct sets of values Ps to different sets of the correct processors. In fact, these sets Ps must reduce to distinct values. We say that a fault P separates sets A and B of correct processors if it sends them sets ps so that after REDUCE, no member of A has a value stored in Ps that is same as that of a member of B. We call any set of correct processors a witness set if its cardinality is at least t and at most n - 2t.

If $n > \max(4t, 2(t + (t - 1)^2))$, then using Eagree the correct processors reach eventually agreement by Leema 7.4.9 (condition (i)) and Lemma 7.4.5 (condition (ii)). By Lemma 7.4.8 and its specification EAGREE requires at most min(f + 2, t + 1) rounds of information exchange. This completes the proof of Theorem 7.4.1.

CHAPTER FIVE

CONCLUSION AND FUTURE DIRECTION

The problems of obtaining interactive consitency appears to be quite fundamental to the design of fault tolerant system in which executive control is distributed. In the SIFT [1_4] fault-tolerant computer under development at SRI, the need for an interactive consitency algorithm arises in at least three aspects of the design:

(1) synchronized of clocks

(2) stabilization of input from sensors, and

(3) agreement on results of dignostic tests.

In the preliminary stages of the design of this system, it was naively assumed that simple majority voting schemes could be devised to treat these situations. The gradual realization that simple majorities are insufficent led to the results reported in the first module of chapter 4.

The algorithm presented in module 1, are intended to demonstrate that such algorithms exist. The construction of efficient algorithms and algorithms that work under the assumption of restricted communications is a topic for future research.

Other questions that are considered include those of reaching approximate aggrement and reaching agreement under various probabilistic assumptions.

In module 2 we could obtain a solution to Weak Byzantine Generals Problem, which is a weaker version of original Byzantine General Problem / metaphor. Byzantine general metaphor is essentially the same problem appeared in module 1.

In module 3, it has been shown that the problem of fault-tolerant cooperative computing cannot be solved in a totally asynchronous model of computation. This does not mean that such problems cannot be practically solved; rather, it means that a more refined model/of distributed computing that reflects realistic assumptions about processor and communication timmings, is needed. These models were considered in modules 4 to 7.

In module 4, probabilistic consensus protocols for asynchronous system with fair schedulers is considered . For a system with fail-stop processors, we showed that $\lfloor (n+1)/2 \rfloor$ correct processes are necessary and sufficient for achieving consensus. In a system with malicious processes, we showed that $\lceil (2n+1)/3 \rceil$ correct processes are necessary and sufficient for achieving consensus. Finally asynchronous byzantine agreement protocol is given along with necessary proofs.

In module 5, problem of approximate agreement on real numbers by processes in a disributed system, is presented. Simple approximation functions are used in two simple-to-implement algorithms for acheiveing approximate agreement - one for a synchronous distributed system and the other for an asychronous system.

The algorithms presented here have the undesirable property that the faulty processes by their actions in the first round can cause the range of values received by correct processes to be arbitratily large, and hence can cause the time to convergence to be arbitratily long. It appears that some of the ideas of[5_2] can also be used to obtain improved initialization rounds for the algorithms that eliminate this possibility.

For future work, we can state a variant of the approximation problem that uses a fixed number r of rounds and in which e is not predetermined. Each process starts with a real value, as before r rounds, the processes must output their final values. The validity condition is the same as before. The object of the algorithm is to ensure the best possible agreement, expressed as a ration of the new diameter of the nonfaulty processes' values to the original diameter. For given n, t, and r, we would like to know the best ratio.

In Module 6, using simple protocols, it is shown how to achieve consensus in constant expected time, within a variety of fail-stop and ommision failure models. Significantly, the strongest models considered are completely asynchronous. All the results were based on distributively flipping coin, which is usable by a significant majority of the processors.

One limitation of the adversary that was crucial for the performance of the protocols in module 6 is that the adversary does not know the internal state of processors, even when they are made faulty. The reason for this requirement is that otherwise by delivering all messages to one specific processor, the adversary can find out the identity of the unique leader by examining the state of the receiving processor. The adversary can then block the messages of the unique leader from reaching all other processors.

A simple modifiation of the protocols given in modeule 6, can make them immune to an adversary who can "peek into the memory" of failed processors. The basic idea is that instead of sending a pair of (possibly encrypted) bits ("leader" bit, "coin" bit), to all processors, a secret sharing scheme with threshold t can be used. The message to processor i will consist of the ith piece of the secret . Suppose the adversary makes up to t processors faulty and gets to see the contents of their memory. This does not help in understanding the contents of any senders message. In particular the adversary cannot use these pieces to identity the unique leader. To reconstruct the secret, all processors later br$\overset{a}{o}$dcast all the piece of secrets that they have received. The adversary cannot prevent such reconstuction of the secret of any nonfaulty sender, since any t+1 pieces can be used. It appears that this approach can be carried out in all va ants of the adversary model that were considered in module 6. This would yield consensus protocols with constant expected running time for t < $n, which tolerate an adversary who knows the internal state of up to t failed processors.

Finally we note that our protocols of module 6 do not work in the presence of even a single Byzantine failure. A faulty processor can simply claim, at every round, that is a leader thus rendering the coin tossing subroutine ineffective. It remain an intersting question to obtain Byzantine ag~~g~~reement p~~ø~~rocedures that are both as simple as efficient.

In module 7, two kinds of Byzantine Agreement are defined and compared. These are Eventual Byzantine Agreement (EBA) and Simultaneous Byzantine Agreement (SBA). The lower bounds of these algorithms are also shown in the module. Several unauthenticated deterministic EBA algorithms are known; but none attains the lower bounds shown in this module, for all n and t with n > 3t.

The question even remains open for authenticated algorithms: Is there a deterministic EBA algorithm that attains the lowerbounds for all n and t with n>3t when the faults are restricted not to corrupt a given authentication protocol? When the faults are restricted to crash, however, the lower bounds are known to be attainable: Fishcher and Lamport provide a simple algorithm for EBA that acheives early stopping by round f+2 (M. Fischer and L.Lamport, private communications).

Finally, I conclude with the note that, his field is relatively new, and considerable work in the field started just a decade back. Much work needs to be done here, to solve the problems related to the field. In the text given above, Some ideas have been given for future research which can be exploited.

BIBLOGRAPHY

[1_0] M.PEASE, R.SHOSTAK, AND L.LAMPORT, Reaching Agreement in the Presence of Faults. J. ACM, Vol 27, No. 2, Apr. 1980 228-234.

[1_1] DAVIES,D., AND WAKERLY,J. Synchrnization and matching in redundant systems. IEEE Trans of compters. C-27, 6 (June 1978), 531-539.

[1_2] DIFFIE,W., AND HELLMAN,M., New direction in cryptography IEEE Trans. Inform. Theory IT-22, 6 (Nov. 1976) 644-654.

[1_3] RIVEST,R.L., SHAMIR,A., AND ADLEMAN,L.A., A method for obtaining digital signatures and public key cryptosystems Comm ACM 21, 2 (Feb 1978) 120-126.

[1_4] WENSLEY,J.H. ET AL SIFT : design and analysis of a fault tolerant computer for aircraft control. Proc IEEE 66, 10 (Oct 1978) 1240-1255.

[2_0] L. LAMPORT, The Weak Byzantine General Problem, J. ACM, Vol 30, No. 3, July 1983, pp 668-676.

[2_1] LAMPORT. L, SHOSHTAK, R., AND PEASE, M. The Byzantine Generals Problems.Trans.Prog Lang. Syst.4.3 (July 1982), 382-401.

[2_2] same as [1_0]

[2_3] GRAY,J.Notes on database operating systems. In operating Systems, an Advantage Course Lecture Notes in Computer Science 60, R. Beyer, R.M. Graham and G. Seegmuller, Eds Springer Verlag, New York 1978, pp 393-481.

[3_0] MICHEAL J.FISCHER, NANCY A. LYNCH, MICHAEL S. PATERSON Impossibility of Distributed Consensus with One faulty Process, J. ACM, Vol 32, No. 2, April 1985, pp 374-382.

[3_1] ATTIYA,C., Dolev,D., AND GIL, J., Asynchronous Byzantine consensus. In Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing (Vancouver,B.c., Canada, Aug.27-29).ACM,New york, 1984, pp.119-133.

[3_2] Ben_OR,M. Another advantage of free choice: Completely asynchronous agreement protocols. In proceedings of the 2nd annual ACM Symposium on Principles of Distributed Computing (Montreal, Quebec, Canada, Aug.17-19) ACM, New York, 1983, pp. 27-30.

[3_3] Bracha,G. An asynchronous [(n-1)/3]-resilient consensus protocol. In Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed computing (Vancouver,B.C., Canada, Aug. 17-19). ACM, New York, 1984 pp.154-162.

[3_4] BRACHA,G., AND Toueg,S. Resilient consnsus protocols. In proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (Montreal, Quebec, Canada, Aug. 17-19). ACM, New York, 1983, pp. 12-26.

[3_5] DEMMILO,R.A., Lynch.,N.A.,AND MERRITT.M.J. Cryprographic protocols.In Proceedings of the 14th Annual ACM Symposium on Theory of Computing (San Francisco, Calif., May 5-7). ACM, New York, 1982, pp. 383-400.

[3_6] Dolev,D.,AND STRONG.H.R. Distributed commit with bounded waiting. In proceedings of the 2nd Annual IEEE Symposium on Reliability in Distributed Software and Database systems. IEEE, New York, 1982, pp. 53-60.

[3_7] Dolev,D., AND STRONG,H.r. Polynomial algorithms for multiple processor agreement. In proceedings of the 14th Annual ACM Symposium on Theory of Computing (San Francisco, Calif., May 5-7). ACM, New York, 1982, pp.401-407.

[3_8] Dolev,D.,FOWLER,R.,LYNCH,N.,AND STRONG,H.R. An efficient algorilthm for Byzantine agreement wihotu authentication. Inf control 52,3(1983), 257-274.

[3_9] same as [5_0]

[3_10] DOLEV,D., LYNCH,N., AND STORCKMEYER,L. Consensus in the presence of partial synchrmnoy. In Proceedings of the 3rd annual ACM Symposium on Principles of Distirbuted Computing (Vancouver, B.C., Canada Aug 27-29). ACM New York 1984 pp 103-118.

[3_11] FISHCER,M., AND LYNCH N. A lower bound for th time to assure interactive consistency. Inf Proc. Lett. 14.4 (1982) 183-186.

[3_12] FISHCER, M., LYNCH,N., AND PATERSON, M. Impossibility of distributed consens with one faulty process. In proceedings of the 2nd Annual ACM SIGACT SIGMOD Symposium on Principles of Database System (Atlanta Ga., Mar 21-23)ACM New York 1983, pp 1- 7.

[3_13] GARCIA-MOLINA, H. Elections in a distributed computing system IEEE Trans Comput (1982) 48-59.

[3-14] LAMPORT L., SHOSTAK,R., AND PEASE,M. The Byzantine Generals problem. ACM TRANS. pROG LANG. sYST 4,3 (JULY 1982) 382-401.

[3_15] LAMPSON,B. Replicated commit. CSL Notebook Entry Xerox Palo Alto Research Center, Palo alto CAlif., 1981.

[3_16] LAMPSON, B., AND STURGIS,H. Crash recovery in a distributed data storge system. Manuscript, Xerox Palo Alto research centre Palo Alto Calif 1979.

[3_17] LINDSAY, B.F., SELINGER, P.G., FALTIERI, C., GRAY, J.N., LORIE,R.A. PRICE, T.G.,PUTZOLU,F.,TRAIGER,I.L.,AND WADE, B.W. Notes on distributed databases. IBM Res Rep RJ2571, IBM Reasearch Division San Jose Calif 1979.

[3_18] LYNCH,N.,FISCHER,M.,AND FOWLER,R, A simple and efficient Byzantine Generals algorithm, In proceedings of the 2nd Annual IEEE Symposium on reliability in Distributed software and Database systems. IEEE New York 1982,pp 46-52.

[3_19] same as [1_0]

[3_20] RABIN,M.Randomized Byzantine Generals. In proceedings of the 24th annual IEEE Symposium on Foundations of computer science IEEE, New York 1983, 403-409.

[3_21] REED, D. Naming and syncrnization in a decentrallized computer system. PhD dissetation Technical Report MIT/LCS/TR-205, Massachusetts Institute ofd Technol;goy. Cambridge Mass 1978.

[3_22] ROSENKRANTZ,D.J. STREARNS, R.E., AND LEWIS, P.M., II System level concurrecncy control for distributed database systems ACM Trans. Database Syst. 3, 2 (June 1978) 178-198.

[3_23] SKEEN,D., A decentrallized termination protocol. In proceedings of the 2nd Annual IEEE Symposium of Reliability in Distributed Software and database systems. IEEE, New York 1982,pp 27-32.

[3_24] SKEEN,D., AND STONEBRAKER, M. A formal model of crash recovery in a distributed system IEEE Trans Softw. Engineering SE-9, 3(May 1983), 219-228.

[3_25] TOUEG, S. Randomized Byzantine Agreements. In Proceeding of the 3rd Annual ACM Symposium on Primnciples of Distributed Computing (Vancouver, B.c. Canada Aug 27-29). ACM New York 1984,pp.163,pp. 163-178.

[4_0] GABRIEL BRACHA AND SAM TOUEG,Asynchronous Consensus and Broadcast Protocols, JACM, Vol 32, No. 2, April 1985, pp. 374-382.

[4_1] same as [3_1]

[4_2] DOLEV, D. Unanimity in an unknown and unreliable environment. In Proceedings of the 22nd Annual Symposium on Foundation of Computer Science (Nashville,Tenn.,Oct.) IEEE, New YOrk, 1981,pp 159-168.

[4_3] same as [3_0]

[4_4] ISSACSON,D.L., AND MADSEN,R.W. MARKOV CHAINS THEORY and PRACTICE. WILEY NEW YORK 1976, PP. 89-100.

[4_5] ITAI,A. AND RODEH,M. Symmetry breaking in distributive networks. In Proceedings of the 22nd Annual Symposium on Foundation of Computer Science (Nashville, Tenn. Oct) IEEE, New York, 1981, pp. 150-158.

[4_6] same as [3_14]

[4_7] same as [1_0]

[4_8] RABIN,M.,AND LEHMANN,D.On the advantages of free choice: A symmetric and fully distributed solution to the dinnng philosophers problem. In Proceedings of the 8th ACM Symposium on the Principles of Programming Languages (Williamsburg, Va Jan 26- 28). ACM New York 1981, pp 133-138.

[4_9] SCHLICHTING,R.D., AND SCHNEIDER,F.B Fail stop processes: An approach to designilg fault tolerant computing systems. ACM Trans, Ciomput. Sys (Aug 1983), 222-238.

[5_0] DANNY DOLEV, NANCY A. LYNCH, SHLOMIT S.PINTER, EUGENE W. STARK AND WILLIAM E.WEIHL. Reaching Approximate Agreement in the Presence of Faults. JACM Vol 33, 3(July 1986), pp 499 - 516.

[5_1] same as [3_2]

[5_2] same as [3_3]

[5_3] same as [3_4]

[5_4] same as [3_7]

[5_5] DOLEV,D.,DWORK.,C., AND STOCKMEYER,L. On the the minimal synchrnous needed for distrib;uted consens. In Proceedings of 24th Annual Symposium on Foundation of Computer Science(Nov.)IEEE, New York, 1983,pp.393-402.

[5_6] same as [3_9]

[5_7] same as [3_11]

[5_8] FISHCER, M.J.LYNCH,N.A., AND PATERSON,M.S. Impossibility of distributed consens problems Distr COmputt. 1,1(1986)

[5_9] same as [3_0]

[5_10] same as [3_14]

[5_11] LUNDELIUS.,AND LYNCH,N.A. A new fault-tolerant algorithm for clock synchronization. In Proceedings of 3rd ACM Symposium on Principles of distributed Computing (Vancouver, B.C., Canada, Aug 27-29). ACM , New York, 1984 pp. 75-88.

[5_10] same as [1_0]

[6_0] BENNY CHOR, MICHAEL MERRITT, AND DAVID B. SHMOYS, Simple Constant-Time Consensus Protocols in Realistic Failure Models, JACM Vol 36, No. 3, July 1989, pp 591-614.

[6_1] ALEXI. W., CHOR, B.,GOLDREICH,.O., AND SCHNORR,C.P. RSA and Rabin : Certain parts are as hard as the wholw SIAM J Comput 17(1988). 194-209

[6_2] AWERBUCH. B. Complexity of network synchronization J.ACM 32.4(Oct 1985) 804-823

[6_3] AWERBUCH, B., BLDUM, M., CHOR, B., GOLDWASSERS AND MICALL.S. How to implement Bracha's O(log n) Byzantine agreement algorithm. Unpublished manuscript.

[6_4] same as [3_2]

[6_5] BRACHA.G.An O(log n) expected rounds randomized Byzantine generals algorithm J ACM 34. 4(Oct 1987), pp)910-920.

[6_4] same as [4_0]

[6_7] CHOR. B. AND COAN, B. A simple and efficient randomized Byzantine agreement algorithm IEEE Trans. softw. Eng SE-II,6(1984),531-539.

[6_8] CHOR B.MERRITT, M,.AND SHMOYS, D.B. Simple constant time consensus protocols in realilstic failute models. In Proceedings of the 4th Annual ACM Symposium on Priciples of Distributed Computing ACM New York 1985, pp, 152-162.

[6_9] COAN.B.Acheiving consensus in fault tolerant distributed systems : Protocols. lower bounds and simulations. PhD. dissertiation MIt Cambridge Mass 1987.

[6_10] same as [3_7]

[6_11] DWORK, C. SHMOYS, D., AND STORCKMEYER, L. Flipping persuasively in constant expected time. In proceedings of the 27th Symposium of Foundation on Computer Science IEEE New York, 1986, pp 222-232.

[6_12] FELDMAN, P.., AND MICALL, S, Optimal aglorithms for Byzantine agreement. in Proceedings of the 20the Annual ACM Symposium on Theroy of computing ACM New York 1988 pp. 148-161.

[6_13] FISHER.M. J., AND LYNCH, N.A. On describing the behavour and implementation of distributed systems. Theroet COmputt Sci 13(1981),17-43.

[6_14] same as [3_0]

[6_15] GOLDWASSER, S., AND MICALI., S. Probabilstic encryption. J.Computt. Syst. Sci. 28,2(1984), 270-299.

[6_16] GOLDWASSER,S., AND MICALI.,AND RACKOFF. C. The knowledge complexity of interactive proof system SIAM J. Comput. 18,1(1989)186-208.

[6_17] KARLIN,A.R., AND YAO,A.C. Probabilistic lower bounds for Byntaine agreement and clock synchrnization. Unpublished manuscript.

[6_18] same as [3_14]

[6_20] same as [1_0]

[6_21]. PINTER. S. Distributd Computation Systems. PhD. dissertation Boston Univ., Boston. Mass., 1983.

[6_22] same as [3_20]

[6_23] ROSEN, E.C. Vulnerability of network control protocols: An example ACM SIGSOFT softw. Eng Notes, 6, 1(1981),6-8.

[6_24] SHAMIR, A. How to share a secret Commun ACM 22,11 (Nov. 1979), 612-613.

[6_25] YAO, A.C. Theory and applications of trapdoor functions. In Proceedings of the 23rd IEEE Symposium on Foundation of Computer Sciece. IEEE, New York 1982 80-91.

[7_0] DANNY DOLEV, RUEDIGER REISCHUK AND H. RAYMOND STRONG Early Stopping in Byzatine Agreement, JACM, Vol 37, No. 4, October 1990, pp 720-741

[7_1] Coan.B. A communication-efficient canonical form fault - tolerant distributed protocols. In procceedings of the 5th Annual ACM,New York,1986,pp.63-72.

[7_2] Cristian.F.Aghili,H.Strong,R,and Dolev,D.Atomic broadcast : from simple message diffusion to Vydantine agrrement in proceedings of the 15th International Conference on Fault Tolerant Coputing (june).1985.pp.1-7.

[7_3] Demillo.R.a., Lynch,N.A.. and Merrit.M.J. Cryptographic protocol. In Proceedings of the 14th ACM SIGACT symposium on Theory of computing(San Francisco,Calif, May 5-7)ACM,New York, 1982,pp.383-400.

[7_4] same as [4_2]

[7_5] Dolev.D. The Byzatine generals strike again, J.Algo, 3 (1982), 14-30.

[7_6] same as [3_8]

[7_7] Dolev.D.AND REISCHUK.R, Bounds on information exchange for Byzantine agreement.J.ACM 32,1(Jan,1985).191-204

[7_8] Dolev.D.REISCHUK,R.and Strong,R. "Eventual" is earlier than "immediate". In Preceedings of the 23rd IEEE annual Symposium on Foundations of cumputer Science. IEEE,New York.1982,pp.196-203.

[7_9] same as [3_7]

[7_9] same as [3_6]

[7_11] Dolev.D.,and Strong,H.R.Requirements for agreement in a distributed system. In proceedings of the 2nd international Symposium on Distributed Data Bases(sept.).1982,pp.115-129.

[7_12] Dolev.D.,AND STRONG.H.R. Authenticated algorithm for Byzantine agreement. SIAMJ>Conput 12 (1983).656-666.

[7_13] DWORK,C.,AND MOSES, Y. Knowledge and common knowledge in a Byzantine environment I:Crash failures. In J.Halpern., ed., Thoeretical Aspects of Reasoning about knowledge (Mionetery calif., Mar 19-22) Morgan Kaufman, San Mateo, Calif 1986, pp 149-170.

[7_14] Fishcher, M., FOWLER,R., AND LYNCH , N . A simple and efficient Bynzantine generals algorithms. In proceeding of the 2nd Symposium on Reliability in Distributed Software and Database Systems (pittsburgh, Pa., July). 1982, pp. 46-52.

[7_15] same as [3_11]

[7_16] same as [2_0]

[7_17] LAMPORT, L. Using time instead of timeout for fault-tlerant distributed systems. ACM Prog Lang Syst 6,2(Apr 1984 1984) 254- 280.

[7_18] LAMPORT,L, AND MELLILAR SMITH, PM Synchrnozing clocks in the presence of faults J ACM 32,1(Jan 1985) 52-78.

[7_19] same as [3_14]

[7_20] MOSES,Y., AND WAARTS, O. Coordinated transversal : (t + 1) - round Bynzantine agreement in polynomial time. In Proceedings of the 29th Annual Sympsoium on Foundation of Computer Science. IEEE, New York 1988.

[7_21] same as [1_0]

[7_22] SCHNEIDER, F. Byantine generals in action Implementing fall stop processors ACM Trans COmput. Syst. 2,2 (May 1984) 146-154.

[7_23] SRIKANTH,T.,AND TOUEG,S. Byzantine agreement made simple : Simulation authentiacation without signature .Distt Comput. 2(1987),80-94.

[7_24] TOUEG.S.,PERRY.K., AND SRIKANTH.T. Fast disributed agreement SIAM j. Comput. 16(1987) 445-457.

JAWAHARLAL NEHRU UNIVERSITY LIBRARY