POLYNOMIAL ALGORITHMS FOR MULTIPLE PROCESSOR AGREEMENT

Danny Dolev

H. Raymond Strong

IBM Research Laboratory
San Jose, CA 95193

ABSTRACT

Reaching agreement in a distributed system while handling malfunctioning behavior is a central issue for reliable computer systems. All previous algorithms for reaching the agreement required an exponential number of messages to be sent, with or without authentication. We give polynomial algorithms for reaching (Byzantine) agreement, both with and without the use of authentication protocols. We also prove that no matter what kind of information is exchanged, there is no way to reach agreement with fewer than t+1 rounds of exchange, where t is the upper bound on the number of faults.

1. INTRODUCTION

In this paper we describe algorithms for achieving agreement among multiple processors. The context for this agreement is a network of unreliable processors that have a means for conducting several synchronized phases of information exchange, after which they must all agree on some set of information. We will assume for simplicity that this set of information consists of a single value from some set of values V.

The type of agreement we will study is called Byzantine Agreement (LSP), Unanimity (Db), or Interactive Consistency (PSL). It results when, in the presence of undetected faulty processors, all correctly operating processors are able to agree either on a value or on the conclusion that the originator of the value is faulty. More explicitly, Byzantine Agreement is achieved when
(I) all correctly operating processors agree on the same value, and
(II) if the sender operates correctly then all correctly operating processors agree on its value.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1982 ACM 0-89791-067-2/82/005/0401 $00.75

Our analysis of the problem is based on the worst case assumption that faulty processors are not predictable and possibly even malicious. An algorithm should sustain any strange behavior of faulty processors, even a collusion to prevent the correctly operating processors from reaching agreement. Even if the correctly operating processors cannot identify the faulty processors, they must still reach Byzantine Agreement. The algorithm should not depend in any way on anticipated behavior of faulty processors.

All previous algorithms for reaching Byzantine Agreement are exponential in the number of messages ($O(n^{t+1})$), where n is the number of processors and t is an upper bound on the number of undetected faulty processors). The new results presented here include algorithms polynomial in the number of bits exchanged, with or without authentication.

We also establish an exact lower bound for the number of phases of information exchange required. This lower bound (t+1) was known for the case in which only unauthenticated messages are exchanged (LF). We have generalized the proof given by Lynch and Fischer to apply to any kind of message.

These results resolve open problems raised by (L), (LSP), (Da), (Db), (PSL) and (LF), and can be used to strengthen the results obtained in the above papers.

The lower bound result is somewhat surprising in our context. It indicates that even though we allow correctly operating processors exchange of any kind of verifiable information, and even though we restrict the possible behavior of faulty processors to simply failing to relay messages, Byzantine Agreement cannot be reached in t or fewer phases. Note that if we relax (I) slightly as in Crusader Agreement (Da), then we can obtain the agreement within two phases.

The algorithms discussed provide a method for a single processor to send a single value to all other processors. Generalizations to many processors sending values to each other will be obvious.

We assume some reliable means of communication by which any correct processor can send a message to any other correct processor. For example, this

reliability might be achieved by sending duplicate messages along many paths in a network. In any case, for this paper, we assume a completely connected, totally reliable communication network, and in counting the total number of messages sent, we ignore any duplication or repetition inherent in the communication medium. Note that we only count the messages sent by correctly operating processors.

All the results obtained in the paper can be extended to networks which are not complete using similar methods to those described by (Da), (Db), and (LPS). The number of phases and likewise the number of messages will increase, but the algorithms will remain polynomial.

For algorithms using authentication, we assume a protocol that will prevent any processor from introducing a new value or message into the information exchange and claiming to have received it from another (DH), (RSA). In a typical authentication protocol (PSL), the transmitter appends a signature to the message to be sent. This signature contains a duplicate of part of the message encoded in such a way that any receiver can verify that the message is authentic and that it was sent by the sender, but no processor can forge the signature of another. Thus no processor can change the content of a message undetectably.

Without authentication, we have the case studied by (LSP), (PSL), (Da), (Db), and (LF). In this case, if d is the number of phases and m is the total number of messages, then the following requirements for Byzantine Agreement were previously established:
(1) $n > 3t$ (PSL), and
(2) $d > t$ (LF).

Algorithms presented used $d=t+1$ and $m=O(n^t)$.

Pease, et. al. (PSL), recognized that with authentication (1) is no longer required. The algorithm for this case presented in (LSP) uses the above d and m. Moreover, the proof of (2) given by Lynch and Fischer depends on (1) as well as requiring the very simple kind of message, disallowing any authentication protocol. Here we establish (2) in a general context with authentication. Also, we will present an algorithm for Byzantine Agreement with $d=t+1$ and $m=O(n^2)$, and a modification with $d=t+2$ and $m=O(nt)$.

For reaching Byzantine Agreement without using authentication one needs a much more sophisticated algorithm. Byzantine Agreement is more difficult without authentication because faulty processors can change intermediate values, and because no processor can identify with certainty those that relayed a given message. We present an algorithm with $d=4t+4$ and $m=O(n^5)$ to obtain the Byzantine Agreement. All previous algorithms used the lower bound for d (i.e., t+1) and an exponential number of messages.

2. HISTORIES

In order to give proofs of correctness and especially to establish lower bounds, we will describe the message related behavior of the collection of processors during the phases of information exchange as a single object called a history. We will formally define a history as a sequence of directed graphs called phases. We intend the notion of history to capture any synchronous information exchange behavior, including any number of authentication protocols and the exchange of arbitrary message types. The lower bound result of section 3 can be extended to asynchronous algorithms with a suitable generalization of the notion of phase.

A phase is a directed graph with nodes corresponding to processors and with labels on the edges. A label represents the information sent from a given processor to another during the given phase. We assume that when no message is sent there is no edge. An n processor history is a finite sequence of n node phases, with nodes labelled by the names of the processors, together with a special initial phase called phase 0, such that phase 0 contains only a single inedge to one processor called the sender. (We assume that the inedge at phase 0 carries the value that the sender is to send.) Figures 1 and 2 represent histories with labels and phase 0 omitted.

A subhistory of a history H is a copy of H with some edges removed. For each history H and processor p there is a unique subhistory pH called the subhistory according to p, consisting of only the edges with target p. Thus, the subhistory according to the sender includes the value it is supposed to send even if it sends nothing.

An agreement algorithm on a class of histories C consists of a correctness rule (a function which given a subhistory according to p and an edge in a phase to be added to the history as the next phase, produces a possibly empty label for that edge) and a decision function (a function from subhistories according to processors of histories in C to the union of V with a symbol 0 representing "sender fault"). With respect to a given correctness rule, a processor p is said to be correct at phase k if each edge from p in phase k has the label produced by the correctness rule operating on the subhistory according to p of the previous k-1 phases. A processor p is correct for history H if it is correct at each phase of H. Figure 1 could represent a history in which all processors are correct, while figure 2 could represent a history with the sender faulty. We call a history t-faulty (with respect to a correctness rule) if at most t of its processors are incorrect.

A correctness rule is actually a union of possibly distinct correctness rules, one for each processor. Likewise, the decision function is a union of individual decision functions.

An example of a correctness rule is the rule that each processor simply sign and relay (according

to the authentication protocol) each incoming message of the previous phase to every other processor.

We say Byzantine Agreement can be achieved for n processors with at most t faults within d phases if there is an agreement algorithm for the class C of n processor, t-faulty (with respect to the correctness rule of the algorithm), d phase histories so that the decision function F obeys the rules for Byzantine Agreement:

(I) if p and q are correct for H in C then FpH = FqH, and

(II) if the sender is correct at the first phase of H and p is correct for H in C then FpH = v where v is the sender's value.

Note that we do not define Byzantine Agreement for n<3 or for t>n. In the context of an authentication protocol, the class C of histories is assumed limited to those consistent with the semantics of authentication.

3. THE LOWER BOUND RESULT

Theorem 1. (LSP) Byzantine Agreement with authentication can be achieved for n processors with at most t faults within t+1 phases, assuming n>t+1.

The following lower bound result is the principal result of this section. It shows that the result of Theorem 1 is tight.

Theorem 2. Byzantine Agreement cannot be achieved for n processors with at most t faults within t or fewer phases, provided n>t+1.

The proof of Theorem 2 is inspired by, but a nontrivial generalization of, the proof given by Lynch and Fischer for the restricted case without authentication (LF). Lynch and Fischer used the n>3t result of (PSL) to show that any algorithm for Byzantine Agreement must be uniform. Assuming uniformity, they established an equivalence relation on their version of t-faulty histories and obtained a contradiction by showing that too many histories were contained in a single equivalence class. Their proof of this equivalence relied on the ability to preserve equivalence while changing one message at a time. Their proof of this ability, without using the uniformity assumption, is essentially the proof of the base case in the induction that follows.

Outline of Proof of Theorem 2:

Assume that Byzantine Agreement can be achieved for some n>t+1 within t phases. Let R be the correctness rule and let F be the decision function on subhistories such that <R,F> achieves Byzantine Agreement on n processor, depth t, t-faulty histories.

Let C be the class of n processor, depth t, t-faulty histories that have a critical sequence such that all incorrect processors appear on the sequence and any incorrect node appears at or below the level corresponding to the order its label appears on the sequence. Define an equivalence relation on histories in C by saying H is equivalent to H' if, whenever p is correct for H and q is correct for H', then FqH' = FpH.

Note that C includes histories in which all processors behave correctly. Since we assume V has more than one value, this means that there must be histories in C that are not equivalent. But, as we will show, C is a single equivalence class. Under an appropriate definition of <R,F>, both Figure 1 and Figure 2 could describe histories from the set C. However, in Figure 2 the result of the algorithm must be independent of any information from the sender since the sender sends nothing. This fact is the key idea behind the contradiction we obtain.

We say that a processor is hidden at phase k if it has no outedges at k or any later phase. We will also refer to the node at phase k as hidden if the processor is. In particular it is easy to show by induction on the phase k that, if r is a node representing a processor at phase k of a history H in C, then
(a) there is a history H' in C, equivalent to H, identical to H through phase k except for outedges of r, with r correct and all processors correct after phase k, and
(b) if all other nodes at phase k are correct, then there is a history H' in C, equivalent to H, identical to H through phase k except for outedges of r, with r hidden and all other processors correct after phase k.
Note that if a processor labels a hidden node, then changing the information on its inedge cannot affect the subhistory according to any other processor. In Figure 2 the sender is hidden at phase 1.

The induction proceeds one edge at a time making changes not witnessed by some correct processor (here we use the fact that n > t+1 to guarantee the existence of such a correct processor). It shows that we can correct a node at any phase or hide a node if all other nodes at its phase are correct, and that the resulting history will be in C and equivalent to the one from which we started, while all changes will be to the outedges of the particular node and to edges at later phases. Thus every history in C is equivalent to any history in which the root is hidden and all other processors are correct.

This completes the outline of proof of Theorem 2, a complete proof can be found in (DS).□

Remark. Whenever it is defined, Byzantine Agreement can be achieved for n processors within n-1 phases.

Thus the provision n>t+1 is necessary for the lower bound of Theorem 2.

4. POLYNOMIAL ALGORITHMS USING AUTHENTICATION

As mentioned in the introduction, we assume the existence of some authentication technique that prevents the faulty processors from undetectably changing the content of messages.

For purposes of counting messages we supply the following specific syntax for the labels on the edges of directed graphs called phases.

(1) The set of values V is contained in the set of atomic messages

(2) A label is either
an atomic message |
an authentication |
a sequence of labels;

(3) An authentication is a label of the form
(labela)p,
where p is the name of a processor and labela is a label; and

(4) a sequence of labels is a label of the form
labela,labelb
where labela and labelb are labels.

Note that (a,b,c)p is not the same label as (a)p,(b)p,(c)p.

Label a is part of label b if either:
(i) a=b, or
(ii) there is a label c and a processor p such that a is part of c and b=(c)p, or
(iii) there are labels c and d such that b = c,d and a is part of c or d.

A message is a label with no commas.

Thus, at any phase any processor can send any message to any other processor, except that no processor can alter an authenticated message received at a previous phase and forward it as an authenticated message at the next phase, nor can any processor pretend to have received an authenticated message it did not receive and forward that as an authenticated message. For this section, attention will be restricted to histories consistent with the semantics of authentication. In particular, if (a)q is part of a label at phase k from processor p then either p = q or (a)q appears as part of a label on an inedge to p in a previous phase.

Theorem 3. Byzantine Agreement can be achieved for n processors with at most t faults within t+1 phases using at most $O(n^2)$ messages.

Outline of proof of Theorem 3:

Our correctness rule will insure that no processor relays more than two messages to any other, regardless of the number of messages received or the number of distinct paths incoming messages may have travelled. A value is said to arrive correctly at a processor if it arrives during the kth phase authenticated by k distinct processors. A processor relays a value only if it arrives correctly and is either the first or the second correctly arriving value seen by the processor. The decision function produces v if, and only if, v is the only value arriving correctly at the processor; otherwise, a default value corresponding to sender fault is produced. We need only run the algorithm for t+1 phases because any value that correctly arrives during phase t+1 has been seen at an earlier phase by a correct processor.□

Theorem 4. Byzantine Agreement can be achieved for n processors with at most t faults within t+2 phases using at most O(nt) messages.

Outline of proof of Theorem 4:

We restrict the correctness rule of the proof of Theorem 3 by arbitrarily choosing t+1 processors to be relay processors and requiring any non-relay processor to send messages only to relay processors (the sender is not a relay processor). All arguments remain the same except that it takes two phases for a correct processor to communicate relevant information to all other correct processors, so we require t+2 phases.□

5. POLYNOMIAL ALGORITHMS WITHOUT USING AUTHENTICATION

The polynomial algorithm presented in this section makes use of the intuition gained in the previous section to attack the general problem. Since we cannot use authentication, we attempt to provide a substitute. Analysis of the previous algorithms reveals that authentication performs two essential tasks:

(1) preventing faulty processors from introducing new values thus insuring that all values considered were actually sent by the sender, and

(2) providing a proof of progress, showing that a value arriving at a given phase has been relayed by other processors in previous phases.

The two thresholds, LOW and HIGH, and the notion of proof of progress defined below provide these properties without using authentication.

Theorem 5. Byzantine Agreement can be achieved for n processors with at most t faults within 4t+4 phases using at most $O(n^5)$ messages without authentication, provided that n>3t.

Outline of proof of Theorem 5:

In order to describe the correctness rule and decision function for our algorithm, we must first describe the specific messages we will use. We do this in the context of describing units of information to be recorded by a correctly operating processor, some of which will be transmitted as messages. These units will be called assertions. There are three types of assertions:

type a - of the form a(v), for v in V, interpreted as "sender s sent value v at phase 1;"
type b - of the form b(v,y,k), for v in V, y a processor, k a phase number, interpreted as "y sent a(v) at phase k;" and
type c - of the form c(v,x,y,k), for v in V, x and y processors, k a phase number, interpreted as "x sent b(v,y,k)."

The original messages sent by the sender will be considered to be a special case of type a. All other messages specified by our algorithm will be either type a or type b assertions.

We will need two threshold numbers, LOW and HIGH with the following properties:

$$LOW > t,$$
$$HIGH > LOW + t - 1, \text{ and}$$
$$n > HIGH + t - 1.$$

Thus the number of correctly operating processors will be greater than or equal to HIGH, while the number of faulty processors will be less than LOW. Note that these requirements imply that $n > 3t$. One example of numbers satisfying the requirements is $LOW = t+1$ and $HIGH = 2t+1$.

Information kept by a processor will be recorded in two categories: known, and committed. If an assertion is recorded as committed it will also be recorded as known. Type c assertions will not be messages and will only be recorded as known.

In our proofs we will let AC stand for the set of correctly operating processors. When we refer to a type b or c assertion without mentioning the phase number but simply asserting one exists, we will write * for the phase number, as in b(v,q,*). Also we will freely replace a processor by a set of processors to refer to the appropriate set of assertions, as in c(v,AC,q,*).

We say <z(2),...,z(j)> is a nonrepeating partial sequence if z is a partial function from the integers between 2 and j inclusive to the set of processors such that when z(i) and z(j) are both defined and z(i) = z(j) then i = j. We say such a nonrepeating partial sequence <z(2),...,z(j)> is a proof of progress with support M if at least half of its positions are defined, it does not contain the sender, and, for each i with z(i) defined, c(v,M,z(i),i) is known.

We now give the correctness rule for our algorithm by specifying the following rules for correct operation for each processor:

1. At phase 1 the sender s broadcasts the value v to all processors and records a(v), b(v,s,2),and c(v,s,s,2) as known. Each processor p that receives exactly one value v from the sender records a(v), b(v,p,2), and c(v,p,p,2) as known. At subsequent phases, the sender will not be distinguished from other processors.

2. At any phase k > 1, each processor broadcasts each assertion of type a or b that became known at the previous phase. On receipt of a type a (type b) message, the appropriate assertion of type b (type c) is recorded as known. When processor p records b(v,y,i) as known, it should also record c(v,p,y,i) as known.

3. If the number of processors x such that c(v,x,y,i) is known is at least LOW, then record b(v,y,i) as known.

4. For processor p to record b(v,y,i) as committed at phase i+1, we require a set of processors M of cardinality HIGH such that there exists a nonrepeating partial sequence <z(2),...,z(i)=y> that is a proof of progress with support M.

5. If the number of processors y, such that b(v,y,i) is committed for some i, is at least LOW (HIGH) then record a(v) as known (committed).

The algorithm is to be run for 2(LOW+t-1)+4 phases after which the decision function produces v exactly when a(v) is the only assertion of type a recorded as committed.

We count only the messages sent in accordance with the correctness rules. This number is easily seen to be bounded by $O(n^5)$, since the number of values with which correct processors must deal is at most n-1, and the number of distinct messages is dominated by the number of distinct type b messages, each of which is sent at most once from each processor to each other processor, while for each value there are fewer than n(2(LOW+t-1)+4) distinct assertions of type b.

The outline of proof of Theorem 5 will be completed by the following five lemmas:

Lemma 1. If the sender correctly sends v to all processors at phase 1, then each correctly operating processor will commit a(v) at phase 3.

The proof of Lemma 1 is straightforward.

Lemma 2. If p in AC commits b(v,q,i), then there is a nonrepeating partial sequence <z(2),...,z(i)=q> that is a proof of progress with support AC.

Outline of Proof of Lemma 2:

Processor p in AC can only commit b(v,q,i) at phase i+1, at which time there is a nonrepeating partial sequence z with z(i)=q that is a proof of progress with support M where |M|=HIGH. Since M has HIGH elements, it has LOW correctly operating elements, so we can replace the above support by a subset D of AC with |D|=LOW. Note that support by correct processors is simultaneously known to all processors since correct processors always broadcast their messages. Thus the proof of progress for p is also a proof of progress for any processor in AC and at phase i+1 each processor

in AC knows b(v,z(j),j) for each j for which z(j) is defined. Consequently, by phase i+2 each processor in AC has z as a proof of progress with support AC.□

Lemma 3. If p in AC sends a(v) at phase i, then at phase i+1, every processor in AC will commit b(v,p,i).

Outline of Proof of Lemma 3:

Suppose p in AC sends a(v) at phase i. In case i = 2 the proof is straightforward and follows that of Lemma 1. Assume i > 2 so that there is some q for which p commits b(v,q,i-2) at phase i-1. By Lemma 2, there is a proof of progress z with support AC such that z(i-2) = q and it is a proof of progress for any correct processor at phase i. At phase i+1 each processor in AC knows c(v,AC,p,i) and thus can add p to the proof of progress z, allowing it to commit b(v,p,i).□

Lemma 4. If a correctly operating processor commits a(v) at phase k, then all correctly operating processors will have committed a(v) by phase k+2.

Outline of Proof of Lemma 4:

Suppose p in AC commits a(v) at phase k. By phase k, p has committed b(v,M,*) with |M|=HIGH. Thus there is a subset D of AC such that each processor in AC has committed b(v,D,*) by phase k and |D|=LOW. Consequently each processor of AC knows a(v) by phase k and sends a(v) by phase k+1. Thus by Lemma 3, each processor of AC commits b(v,AC,*) by phase k+2 and is able to also commit a(v).□

Lemma 5. If any correctly operating processor has committed a(v) by phase 2(LOW+t-1)+4, then all have.

Outline of Proof of Lemma 5:

Let k = 2(LOW+t-1). If p in AC commits a(v) by phase k+2, then all AC commit a(v) by phase k+4 according to Lemma 4. Suppose p in AC commits a(v) after phase k+2. Then p commits some b(v,q,i) with i > k+1. Let z = <z(2),...,z(i)=q> be the relevant proof of progress with HIGH support. Note that each defined z(j) actually sent a(v) at phase j. Half the z(j) are defined and none are the sender. Also note that if the sender were in AC, then all would have committed a(v) at phase 3. Thus at least LOW correctly operating processors sent a(v) by phase k+1. Let this subset of AC be D. By Lemma 4, each processor in AC commits b(v,D,*) by phase k+2. Thus each processor in AC knows a(v) by phase k+2, sends a(v) by phase k+3, and commits b(v,AC,*) and a(v) by phase k+4.□

This completes the outline of proof of Theorem 5.□

6. CONCLUSION

The main contribution of this paper is that for the first time Byzantine Agreement is feasible. We believe that better algorithms can be developed, and that the ideas behind Byzantine Agreement can be applied to other issues of reliability for systems of processors.

The lower bound of t+1 phases with authentication means that we must look elsewhere to achieve Byzantine Agreement quickly. In fact we must relax some requirement because the lower bound of Theorem 2 applies no matter what kind of message we send. Although the proof is given in the context of synchronous phases, any purported asynchronous algorithm for Byzantine Agreement would certainly be imbeddable within the synchronous phase context by simply imposing the phases on its behavior.

One possibility would be to look at algorithms that probably achieve Byzantine Agreement. In a probabilistic context, if we had a realistic upper bound t on the number of possible faults, then we would likely also have information on the probability of exactly t faults, exactly t-1 faults, etc.

While they do not reduce the minimum number of phases required, our algorithms do reduce the total number of messages required for Byzantine Agreement from exponential to polynomial in the number of processors. We have assumed complete and reliable communication among the processors. Note that this may be achieved in an unreliable and incompletely connected network (Da) and that if we are given an algorithm for reliable communication of a message using a number of messages polynomial in the number of processors, then we can convert that algorithm to one which achieves Byzantine Agreement in a polynomial number of messages.

Our algorithmic results suggest the following open problem: What is the tradeoff between phases and messages required for Byzantine Agreement? The algorithm without authentication requires four times as many phases as the lower bound. However the lower bound is tight, since there exist algorithms that use only t+1 phases, but require an exponential number of messages.

Acknowledgements:

The authors thank Nancy Lynch for helpful suggestions about this manuscript. After completing this work, the authors received a somewhat similar proof of Theorem 2 from M. Merritt (DLM).

7. REFERENCES

(DH) W. Diffie and M. Hellman, "New direction in cryptography," IEEE Trans. on Inform. IT-22,6(1976), 644-654.

(Da) D. Dolev, "The Byzantine Generals Strike Again," Journal of Algorithms, vol. 3, no. 1, 1982.

(Db) D. Dolev, "Unanimity in an Unknown and Unreliable Environment," 22nd Annual Symposium on Foundations of Computer Science, pp. 159-168, 1981.

(DS) D. Dolev and H. R. Strong, "Authenticated Algorithms for Byzantine Agreement," submitted for publication; see also "Polynomial algorithms for multiple processor agreement," IBM Research Report RJ3342 (1981).

(DLM) R. A. DeMillo, N. A. Lynch, and M. Merritt, "Cryptographic Protocols," in these proceedings.

(L) L. Lamport, "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," Technical Report, Computer Science Laboratory, SRI International, 1981.

(LSP) L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," ACM Trans. on Programing Languages and Systems, to appear.

(LF) N. Lynch, and M. Fischer, "A Lower Bound for the Time to Assure Interactive Consistency," submitted for publication.

(PSL) M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," JACM, vol. 27, no. 2, pp. 228-234, 1980.

(RSA) R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Comm. ACM 21 (1978), 120-126.

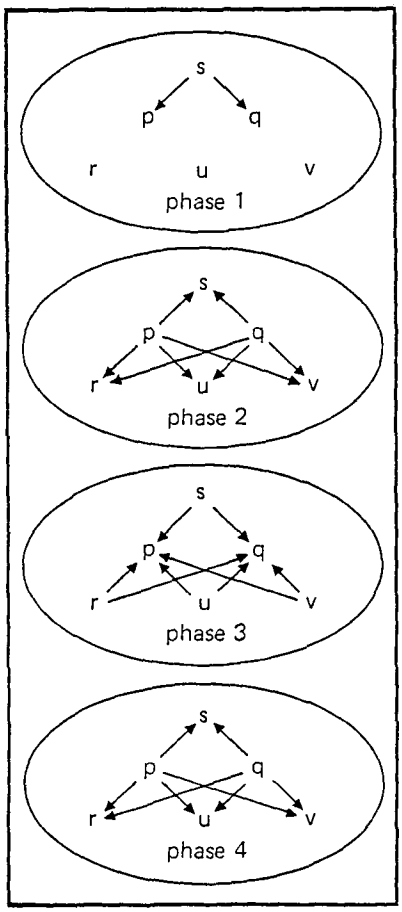

Figure 1. A six processor four phase history with edge labels and phase 0 omitted.

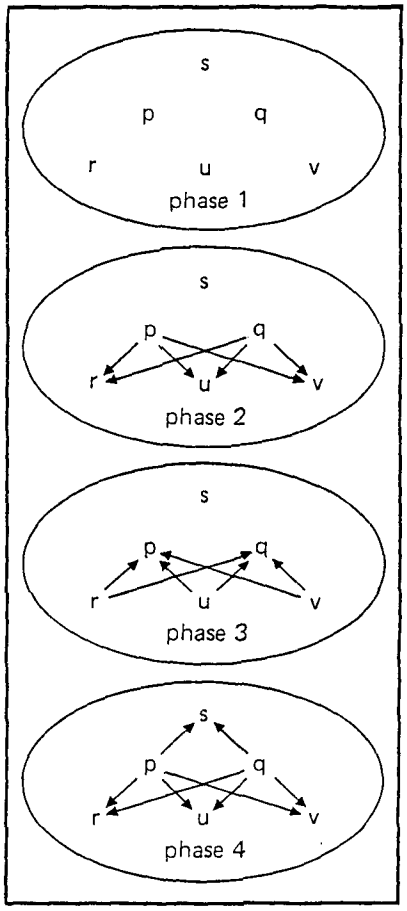

Figure 2. The result of hiding sender s at phase 1.