FINDING SAFE PATHS IN A FAULTY ENVIRONMENT[1]

by

D. Dolev
IBM Research
San Jose Laboratory

J. Meseguer
Computer Science Laboratory
SRI International

M. C. Pease
Computer Science Laboratory
SRI International

1 Introduction

This paper addresses the problem of finding safe paths through a network, some of whose nodes may be faulty. By a **safe path** we mean one between two nodes that does not contain any faulty node. The kinds of faults that concern us are not limited to those that may cause a failure of a node or link, but include those that may cause a node to distort messages in arbitrary ways. Furthermore, we want a distributed algorithm to allow the network itself to discover suitable paths without depending on a central controller for the analysis. More broadly, we assume that each node has only local knowledge of the network structure.

We can think of a network of mobile radio transmitters as typical of the problem that concerns us. As transmitters move, communication links may be broken unexpectedly, and new links formed; the topology of the network changes unpredictably with time. The rate of change, however, can be assumed to be sufficiently low so that a set of paths will remain useful for some time. We assume, also, that the rate at which information must be transmitted reliably, and the volume that is likely to be involved in a given period of time, is high enough so that it would be very costly to depend on broadcast methods. Under these conditions, it is worthwhile to invest the effort in discovering suitable sets of paths on which we can depend for a significant period before they must be redetermined. We seek an algorithm that will support the communication requirements of such a network.

The algorithm developed here depends on the use of an authentication protocol that allows a node to test whether a message has been altered. To guard against malicious behavior, public-key encryption methods such as those described in references [2, 14] can be employed. If malicious behavior is not an issue, it may be enough to use simple error-detecting codes to guard against accidental

[1]The research reported herein was supported in part by Army Research Office Contract No. DAAG29-79-C-0102 and Department of the Navy Contract No. N00039-80-C-0571.

distortion. For our purposes, we simply assume some suitable method for authentication.

It should be recognized that the use of authenticators does not prevent a faulty node from generating misinformation. In particular, a faulty node can provide misinformation about the route a message has followed. To see this, we must understand that, during one phase of the algorithm, messages are sent backwards from the target towards the initiator.[2] These messages contain the route that has been traced by the message. Each node, when it receives one of these messages, first checks, by using the successive authenticators for the nodes on the asserted path, that it has not been obviously distorted. Suppose, for example, T is the target node and that X receives a message M that asserts it was originated by T and transmitted to X through nodes A and B. The message received by node X can be represented as $S_B S_A S_T M$, where, for node Y, S_Y denotes the authenticator of Y. By working back along this chain, using the inverse operators to S_B, S_A and S_T (the public keys) on the successive fragments of the message, node X can verify that the message it has received is consistent with the claimed route

$$T \rightarrow A \rightarrow B$$

This process protects against the message being garbled, but it does not protect against the route information being truncated. If node B is faulty, for example, the message may actually have arrived at node X along the longer path $T \rightarrow A \rightarrow C \rightarrow B \rightarrow X$. The message received at node B was $S_C S_A S_T M$. In this case, node B has stripped off node C's contribution and relayed the remaining message fragment, with its own authenticator, to node X; this message will pass all tests that can be applied at node X. Authenticators do not prevent a faulty node from misrepresenting the path a message has taken.

The outcome of applying the algorithm is a set of independent paths that the initiator can use to send messages to the target node. The initiator will not know which of these paths are fault-free, but, if it has more than m node-disjoint paths, where m is the number of possibly faulty nodes, at most m of the paths can contain faulty nodes, with the rest fault-free. To ensure reliable transmission, the initiator sends duplicate copies of all important messages along at least m + 1 paths, including with each message its own authentication. The initiator can then be certain that the target will receive (and be able to identify) at least one valid copy of the message. Our algorithm assumes that neither the initiator nor the target are themselves faulty. In case one or both are faulty, no reliable communication between them is possible.

It is worth emphasizing that the use of some kind of authentication seems to be necessary in a reliable communication network using an open transmission medium such as radio. It is generally necessary that any node know unambiguously which node is the immediate source of a message. In a network with a closed medium (e.g., where all communication is by hard-wired node-to-node links) the source of a message is identified by observing the link over which it arrived. Where a medium such as radio is used, its source must be identifiable from the content of the message, or through an uncorruptible hand-shaking protocol. This appears to require some form of authentication.

From the above it follows that, when the route is identified by the sequence of authenticators in a message, a faulty node must add its own authenticator. In the truncation example discussed above, node B was able to eliminate node C from the route, but could not fail to include itself. This fact has important implications in the proof of our algorithm.

Our algorithm can be understood as a distributed version of the Ford-Fulkerson flow algorithm [10], or as an improved version of the concurrent algorithm described by Segall [15]. It might be questioned why we have not based our work on Dinits' algorithm [3] or some variation of it such as that developed by Sleator [16], both more efficient for centralized computation. In our earlier report [12], we develop a decentralized algorithm based on Dinits' work. However, Dinits' approach seems to be less efficient in the distributed situation because it may require more stages than does a Ford-Fulkerson type. It is advantageous to minimize the number of stages if we are concerned with the number of messages being transmitted, particularly since each stage includes a broadcast phase. This argument appears to make it convenient to use a Ford-Fulkerson type of algorithm.

[2] We will be consistent in our use of the terms "initiator" and "target." The former indicates the node that initiates the process and that has the need for reliable communication to another node which is called the target. We will use the terms "source" and "receiver" to indicate a particular transmission of a message across a single link, regardless of where the message originated or its ultimate destination.

The algorithm presented here develops incrementally the set of paths in successive stages. Each stage increases the number of paths from the target to the initiator. Obviously it makes no difference for a path to be described either from the target to the initiator or from the initiator to the target; since in our algorithm the paths are developed starting from the target, **we will always represent paths from the target to the initiator**. Each stage consists of three phases as follows:

- Phase 1: The initiator originates a message that identifies the stage and lists the paths between it and the target that have been found in previous stages. (In the first stage, there are no known paths.) This message is transmitted by broadcast mode through the network until it reaches the target. The message carries the initiator's authenticator and is relayed at each node without change--i.e., intermediate nodes check its authenticity but do not add their own authenticators. On receipt of this message at any intermediate node, the node records both the stage number and any relevant local routing information implied by the paths listed in the message. That is, when a non-target node X receives the message it looks to see if it is on any path listed in the message. If it is, it records its predecessor and successor nodes on that path. After receipt of this message the node will ignore any messages from an earlier stage. Thus the broadcast message serves to terminate any residual activity that may be left over from earlier stages.

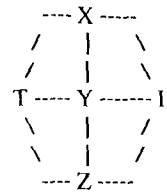
- Phase 2: This phase is initiated by the target when it receives the phase 1 message for a new stage. The target creates a message from the stage number and its own authenticator. It sends this message to all neighbors that are not its successors on any of the paths listed in the phase 1 message. Each node receiving this message adds its own authenticator and sends it to those of its neighbors that are "suitable." The rules that govern which neighbors are suitable for relaying a message and which of those will accept it ensure that no new path that violates the constraints on either the link or node capacities is developoed. This process continues until the initiator is reached. The initiator examines each phase 2 message as it is received to ensure that it is consistent with the paths developed in the previous stage, since a faulty node may have introduced an inconsistency not recognized by any other node. The initiator then stores the accepted messages until phase 3.

- Phase 3: The third phase is executed by the initiator alone, and does not involve sending any messages. This is permissible at any time after at least one phase 2 message has been accepted, up to some specified time-out. During phase 3, the source recomputes its paths from the target, updating its list of paths. After this, it may initiate a new stage by originating a phase 1 message with the new stage number.

The process terminates either when the initiator decides that enough independent paths have been found, or when no authenticatable and acceptable phase 2 messages are returned within the specified time-out period.

Any path found by this algorithm that does not contain a faulty node will be a valid path. To see this, consider the "augmented network" derived from the actual one by adding links from each faulty node to every other node (except the source) to which it is not originally linked. All paths found by the algorithm are valid paths in this augmented network-- therefore those that do not use faulty nodes must be valid paths in the original network.

It might appear that the process, if carried to completion, would lead to a maximal set of possible paths between the initiator and the target in the augmented network. This is not so, however, since a faulty node may respond incorrectly to phase 2 messages. In particular, consider the following network:



An Exemplary Network

Suppose, node Y is faulty and that, in the first stage, the path

$$T \rightarrow X \rightarrow Y \rightarrow Z \rightarrow I$$

from the target to the initiator, is developed. In phase 2 of the second stage, the link from Z to I is saturated in the flow-theoretic sense; therefore the phase 2 message is sent from I only to X and Y. Node Y is assumed faulty and may, therefore, fail to relay the message. Node X's neighbors are T and Y; the link from T to X is however already saturated, so X sends its message only to Y. Node Y, being faulty, can again refuse to relay the message. If so, no new path is found and the process terminates. In effect, the faulty node, Y, is able to maintain the path through itself at the expense of two other fault-free paths.

The example discussed represents an extreme case. In general, each faulty node, located in the worst possible place and misbehaving in

the most destructive way, can prevent the discovery of, at most, two fault-free paths, while forcing the continued use of a path through itself. Hence, if there can be as many as m faulty nodes, the connectivity of the network between T and I must be at least (3m + 1), if at least one fault-free path from the initiator to the target is to be found in this manner.

The main idea behind the proof of the algorithm depends on the use of the augmented network defined above. We compare the actual network, the augmented network, and the the directed network determined by the residual flow left by a set of paths. It is shown that the connectivity 3m + 1 mentioned above is actually sufficient for our algorithm to find at least one reliable path.

As a measure of the complexity of the algorithm, the basic procedures require the transfer of $O((m + k) \times n)$ authenticators per link where n is the order of the network, m is the maximum number of faults, and k is the required number of fault-free paths. If we seek the maximal number of paths between the initiator and the target, a variation of the algorithm requires the transfer of $O(|E|)$ authenticators per link, where $|E|$ is the number of links. Under some conditions, this bound can be substantially reduced.

Acknowledgements

We would very much like to thank Richard Schwatz for his most valuable help in improving the exposition. Possible mistakes remain our sole responsibility.

2 Path-Finding Walks

Let G = (N,E) be an undirected graph with set of nodes N, and set of edges E. Nodes will be denoted by upper case letters like A, B, etc. . T and I will denote two distinguished nonadjacent nodes of G caled the **target** and the **initiator**. $p_G(T,I)$ shall denote the maximum number of node-disjoint paths, or **connectivity**, from T to I.

Connectivity problems in an undirected graph can be translated into flow problems for an associated directed network. This requires "splitting" of the nodes to transform "node capacities" into edge capacities.

Let $G^{\S} = (N^{\S},E^{\S})$ be the directed network constructed from the original G by splitting each node A in N into two nodes denoted A^1 and A^2, and having edges: (i) $A^1 \rightarrow A^2$ for each node A different from T and I; (ii) $A^2 \rightarrow B^1$ and $B^2 \rightarrow A^1$ for each undirected edge $A \rightarrow B$ in E. All the edges of $G^{\S}$ have unit capacity. It is well-known [1] that the integer $p_G(T,I)$ is the maximum flow from T^2 to I^1 in the network $G^{\S}$.

Every collection of node-disjoint paths from T to I in G corresponds to a unitary flow (i.e., the flow is either 0 or 1 for each edge) from T^2 to I^1 in the network $G^{\S}$. Conversely, any such flow corresponds to a collection of node-disjoint T-I paths, after eventual elimination of useless loops. In particular, a path p = TABC...I in G corresponds to a unit flow along the path $T^2A^1A^2B^1B^2C^1C^2...I^1$ in $G^{\S}$. For each path p in G, the corresponding path in $G^{\S}$ shall be denoted by $p^{\S}$. There is a variant of $G^{\S}$ which can be used to find new paths in addition to a given collection P of node-disjoint paths.

Definition: For $P = \{p_1, ..., p_n\}$ a collection of node-disjoint T-I paths in G, the **residual network R[G,P] associated with** P is the graph obtained from $G^{\S}$ by replacing each edge $X \rightarrow Y$ by an edge $Y \rightarrow X$, wherever Y is the succesor of X in $p^{\S}$, for p a path in P. All the edges of R[G,P] assume unit capacity.

Lemma 1: (cf. [1], Lemma 6.1.) The maximum number of node-disjoint directed T^2-I^1 paths in R[G,P] is exactly

$$p_G(T,I) - |P|$$

where |P| denotes the cardinality of P. ∎

Definition: A **legal walk** for P in G is a T^2-I^1 path in R[G,P], for P a set of T-I paths in G. A string of nodes of G, TABC...I, is also called a **legal walk in reduced form** if it can be obtained from a legal walk by first eliminating the superscripts 1, 2, and then identifying any two contiguous repeated nodes. Similarly, a path beginning at T^2 in R[G,P] is called a **legal segment** for P in G, and its reduced version is defined in the same way.

The following characterization of legal walks and legal segments is used in the proof of correctness of the algorithm. It basically says that legal walks are composed of segments of nodes not occurring in any path (denoted by A's below,) and segments of nodes corresponding to portions of paths in reverse order (denoted by B's below.)

Lemma 2: A string of nodes is the reduced version of a legal walk for P in G if and only if it has the form

$T\ A_{1\,1}...A_{1\,k_1}\ B_{1\,1}...B_{1\,h_1}\ A_{2\,1}...A_{2\,k_2}...B_{m\,1}...B_{m\,h_m}\ A_{m+1\,1}...A_{m+1\,k_{m+1}}\ I$

where

(i) nodes adjacent in the string are also adjacent in G and no repeated nodes are adjacent in the string;

(ii) $m \geq 0$, and each node $A_{i\,j}$ occurs only once in the string and does not occur in P;

(iii) each node $B_{i\,j}$ occurs in P, and does not occur more than twice in the string; each segment $B_{j\,1}...B_{j\,h_j}$ has length $hj \geq 2$, and there is a path in P of the form $T\ X_1...X_n\ B_{j\,h_j}...B_{j\,1}\ Y_1...Y_k\ I$, with $n,k \geq 0$.

Similarly, a string of nodes $T\ C_1...C_k$ with $C_k \neq I$ is the reduced version of a legal segment for P in G if and only if nodes adjacent in the string are adjacent in G and, in addition either

(a) the string $T\ C_1...C_k\ I$

or

(b) the string $T\ C_1...C_kC_{k+1}\ I$ (where C_k occurs in a path of P, C_{k-1} is not its successor in the path, and C_{k+1} is its predecessor)

is the reduced version of a legal walk for P in G^c, where G^c denotes the **complete** graph (i.e., it has all possible edges) on the set of nodes N. ∎

If we see a legal walk and a set of paths as flows (in R[G,P] and $G^§$ respectively,) they can be superimposed, as in [1] pg. 98, to obtain another flow corresponding to a set of paths P' with cardinality |P|. P' can be computed directly from P and the reduced version w of the legal walk as the set of paths **Regen**(P,w) determined by the edges E[P] occurring in P after (i) removing all edges $A \rightarrow B$ such that AB or BA occurs in w; (ii) adding all edges $A \rightarrow B$ not occurring in P, for AB occurring in w.

3 The Algorithm

Let G = (N,E) be an undirected graph, with I and T denoting the **initiator** and **target** nodes, respectively. We assume only one initiator and one target at a time; the general case of simultaneous attempts by several initiator nodes could be handled in parallel in a similar way.

A. ALGORITHM FOR A NODE OTHER THAN I

Each node maintains the following variables

init	for the name of the **initiator**
targ	for the name of the **target**
st	for the stage number received from the initiator
succ	for the successor in the path
pred	for the predecessor in the pah

All variables have initial value **nil**. By convention **nil** < n, for all integers n.

A.1. When receiving a message $M = S_j(msg,s\#)$ from a neighbor X

(comment, s# is the stage number; msg is the name T of the target if s# = 1, otherwise a list of T-I paths)

Case 1.1. If s# > st, then

init := I ;
targ := T ;
st := s# ;
succ := the node's successor in a path in msg, if any ;
pred := the node's predecessor in a path in msg, if any ;

send M to every neighbor except X

Case 1.2. If s# ≤ st, then

ignore the message

A.2. When receiving a message $M = S_{Bn}...S_{B1}(s\#)$ from a neighbor X, with $n \geq 1$, $B1 \neq I$, and $Bn = X$

Case 2.1. If st < s#, then

wait until receiving a message of the form $S_j(msg,s\#)$;
process this last message as in A.1, and go to Case 2.2

Case 2.2. If either st > s#, or X = pred, or $A1 \neq$ targ, or some Ai appears more than twice or twice and contiguously, then

ignore the message

(comment: by the case split st ≠ nil)

Case 2.3. If either succ = pred = nil, or X = succ, then

sign M and send it to every neighbor except X, targ, and any Ai appearing twice in the list of signatures

(comment: by case split st = s#)

Case 2.4. If succ and pred not nil, and X ≠ succ, and pred ≠ targ, then

sign M and send it to pred

(comment: by case split st = s#)

B. ALGORITHM FOR THE TARGET

The target T maintains the following variables:

init	for the name of the **initiator**
st	for the stage number
srs	for the list of its succesors in the developed paths

All variables have initial value **nil**. As before **nil** < n, for all integers n.

When receiving a message of the form $M = S_1(msg,s\#)$

(comment: when s# = 1, then msg = T)

Case 1. If s# > st, then

init := I ;
st := s# ;
srs := the successors of T in msg if s# > 1, **nil** if s# = 1 ;

sign message (st), and send it to every neighbor except those in srs

(comment: s# = s+1)

Case 2. If s# ≤ st, then

ignore the message

C. ALGORITHM FOR THE INITIATOR

The initiator tries to find k nonfaulty paths to the target, under the assumption that no more than m nodes are faulty. If t_0 is the maximum time that it takes for a node to process a message, and |N| is the total number of nodes, we define **timeout** = $3 \times |N| \times t_0$. The transmitter keeps the following variables:

τ	an elapsed time counter
st	for the stage number
P	for the current paths

all variables have initial value **nil**

C.1. If $\tau >$ **timeout**, or |P| = k+t, then

STOP

(comment: either k nonfaulty paths have been found or no more paths can be found)

C.2. When receiving a message of the form $S_X...S_1(s\#)$ from neighbor X

Case 2.1. If s# < st, or the string T...XI is not a legal walk in reduced form for P in the complete graph on the set of nodes occuring in either the paths of P, or the string T...XI, then

ignore the message

(comment: to decide if T...XI is a legal walk for P in the complete graph, the initiator checks the conditions in Lemma 2)

Case 2.2. otherwise

st := st + 1 ;
P := **Regen**(P,T...XI) ;

sign message (P,st), and send it to every neighbor

4 Proof of Correctness of the Algorithm

Theorem 3: If the undirected graph G = (N,E) has connectivity $p_G(T,I) \geq 3m + k$, between nonadjacent nodes T and I, and the nodes of G perform the algorithm of last section with I as initiator, T as target, and no more than m of the other nodes faulty, the initiator I will have at least k node-disjoint T-I paths of G not involving any faulty nodes among its list P of paths, before the timeout of stage m + k.

Proof. A faulty node can eliminate a collection of most-recent signatures from a message. Since the only information the initiator has at any given time is the list of already developed paths and the list of its neighbors, faulty nodes could cause the initiator to accept legal walks for the paths P in the complete graph on N which are not legal in G. Thus the new paths developed by the initiator from this information will not be in general valid paths of G.

We first show that the list P developed by the initiator is always a list of valid paths in a graph G^{aug}, called the **augmentation** of G, and obtained from G by connecting each faulty node to all other nodes except the initiator.

Lemma 4: At any stage the paths of P are valid paths in G^{aug}

Proof. Using induction, it is enough to assume the result for the P obtained at the end of stage n, and then show that if in stage n+1 the initiator accepts as legal string, the string will be legal in G^{aug}. This is equivalent to showing that if the string received by the initiator is not a legal walk for P in G^{aug}, then it is not a legal walk for P in the complete graph for N. Let T $A_1 ... A_g$ I be such a string. By cases A.2.2 - A.2.4 in the algorithm the string must contain a faulty node with no nodes occurring either more than twice or twice and contiguously. Let T $A_1 ... A_h$ be the smallest illegal segment for P in G^{aug}. A_h must be faulty; otherwise T $A_1 ... A_{h-1}$ would be illegal, by the cases A.2.2 - A.2.4 mentioned above. Note also that A_{h-1} must belong to one of the paths in P (otherwise T $A_1 ... A_h$ would be legal.) For the same reason A_{h-2} cannot be the successor of A_{h-1} in such path. Thus A_h is not the predecessor of A_{h-1} in the path. Therefore by condition (iii) in Lemma 2, T $A_1 ... A_h$ is not a legal segment for P in the complete graph for N, completing the proof of the lemma. ∎

Since a path in G^{aug} not involving faulty nodes is also a path in G, to complete the proof of the theorem we have only to show that if the initiator broadcasts a list P of j paths in G^{aug} at the beginning of phase j + 1 (for $0 \leq j \leq m + k - 1$), then the initiator will receive a list of signatures corresponding to a legal walk for P in G^{aug} before the counter τ reaches **timeout**. By case A.2 of the algorithm, this is equivalent to showing that the faulty nodes cannot block all the paths of R[G,P]. Since the initiator can detect illegal walks for P in G^{aug} (by the proof of Lemma 4,) the worst case is to assume that the faulty nodes do not answer any message during phase j + 1.

Note that, in any case, the message from the initiator to the target reaches the target, since only the existence of m+1 node-disjoint I-T paths in G is needed. To see that a message sent from the target will reach the initiator without passing through any faulty node, note that, by Lemma 1, R[G,P] has

$$3m + k - j \geq 2m + 1$$

node-disjoint paths from the target to the initiator. The faulty nodes not answering a message corresponds to the removal of at most 2m nodes in R[G,P]. But this leaves one or more node-disjoint paths not involving faulty nodes in R[G,P]. Any such path has length smaller than $2 \times |N|$. Thus the message will reach the initiator before **timeout**, which finishes the proof. ∎

5 Complexity Analysis

The algorithm presented above requires the initiator to broadcast all the paths it has obtained in the previous stage at the beginnig of each new stage. Broadcasting requires at most 2|E| messages, since each processor forwards the information at most once to each neighbor. The second phase of each stage involves authenticated lists of processors being developed as the messages are transmitted through the network from the target to the initiator. Counting an authenticated signature as one, if the initiator wants k reliable paths, the total complexity is $O((k+m) \times |N| \times |E|)$. To find the maximum number of paths takes up to |N| stages, which adds up to $O(|N|^2 \times |E|)$.

These bounds exceed the upper bound on the number of messages needed if one just broadcasts all the information to the initiator and lets the initiator find the paths. Such a straightforward algorithm requires $O(|E|^2)$ messages. But it requires the initiator to have $O(|E|)$ space for finding the connnectivity. Observe that the same number of messages is required for finding even one reliable path. $|E|$ denotes here the number of edges in the augmented graph G^{aug}. The algorithm we have presented above can be improved to take $O(|E|^2)$ messages for finding the maximum number of paths, while requiring even less for finding k reliable paths. During each stage the initiator sends only the new path that has been found in the last stage. With that information, every processor can update its local routing information to help the initiator obtain another node-disjoint path at the end of the stage. This change requires continued broadcasting from the initiator to the target, even when a new stage has been started. A processor has to wait until it receives the required updating information; if a processor does not receive the information, it does not proceed, since it is disconnected from the initiator by faulty processors.

The requirement of $3m + k$ connectivity reflects worst case behavior. As a matter of fact, to find k nonfaulty paths the size of the minimal cut (in the sense of network flow) from the initiator to the target has to be at least $3m' + k$, where m' is the number of faulty processors on the minimal cut. It may also happen that other parts of the network have low connectivity -- and the number of faults can exceed m', or even $(3m' + k)$, without harm, as long as they are scattered.

6 Concluding Remarks

The algorithm introduced is not the only one that can overcome malicious behavior while still finding communication paths. Moreover, more efficient algorithms likely exist. However, trying to adapt more efficient centralized flow algorithms does not necessarily help, since in general those algorithms make use of data structures which have no efficient implementation in a distributed context.

The work described here is among the first to explore the area of fault-tolerant algorithms which do not restrict the types of possible faults. If the possibility of nodes generating misinformation were not an issue, the problem would be much simpler.

Another important problem is to develop algorithms to find safe paths without requiring authentication. Algorithms without authentication have been developed for reaching agreement in a faulty network [12-13, 5-7]. Ideas in these algorithms could help in solving the problem.

7 References

1. S. Even, Graph Algorithms, Computer Science Press, Woodland Hills, California.

2. W. Diffie and M. Hellman, "New directions in cryptography," IEEE Trans. on Information, IT-22,6 pp 644-654 (1976).

3. E.A. Dinits,"Algorithm for the solution of maximal flow in a network with power estimation," Soviet Mathematics, Report No. 11, pp 1277-1280 (1970).

4. D. Dolev, "The Byzantine Generals Strike Again," Journal of Algorithms, Vol. 3, No. 1, (1982).

5. D. Dolev, "Unanimity in an Unknown and Unreliable Environment," 22nd IEEE Ann. Symp. Found. Comp. Sci., pp 159-168 (1981).

6. D. Dolev, M. Fischer, R. Fowler, N. Lynch, and R. Strong, "Efficient Byzantine Agreement Without Authentication," in preparation.

7. D. Dolev and H. R. Strong, "Polynomial Algorithm for Multiple Processor Agreement," Proc. 14th ACM Sigact Symp. Theor. Computing, May 1982; see also IBM Res. Report RJ3342 (1981).

8. D. Dolev and H. R. Strong, "Authenticated Algorithms for Byzantine Agreement," submitted for publication.

9. D. Dolev and A. C. Yao, "On the Security of Public Key Protocols," Proc. 22nd IEEE Symp. Found. Comp. Sci., pp 350-357 (1981).

10. L. R. Ford and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, New Jersey, 1962.

11. L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," ACM Trans. Programming and Systems, to appear.

12. M. C. Pease, J. G. Meseguer, and D. Dolev, "A decentralized algorithm for network connectivity," Technical Report CSL-127, SRI International, Menlo Park, California (July 1981).

13. M. C. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults", J. ACM, Vol. 27. No. 2, pp 228-234 (1980).

14. R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," Comm. ACM, Vol. 21, pp 120-126 (1978).

15. A. Segall, "Decentralized maximum flow algorithms," submitted for publication.

16. D. K. Sleator, "An O{nm(log n)} algorithm for maximum network flow," Ph. D. dissertation, Department of Computer Science, Stanford University, Stanford, California (Nov. 1980).