ANNALS OF THE NEW YORK ACADEMY OF SCIENCES
*Volume 576*

# GRAPH THEORY AND ITS APPLICATIONS: EAST AND WEST

## Proceedings of the First China–USA International Graph Theory Conference

*Edited by Michael F. Capobianco, Meigu Guan, D. Frank Hsu, and Feng Tian*


*The New York Academy of Sciences*
*New York, New York*
*1989*

# Distributed Graph Recognition with Malicious Faults[a]

EDWARD T. ORDMAN
*Department of Mathematical Sciences*
*Memphis State University*
*Memphis, Tennessee 38152*

## INTRODUCTION

Let $G$ be an undirected graph, that is, a set $V$ of vertices and a set $E$ of edges (unordered pairs of vertices). Suppose at each vertex there is a processor, and messages can be passed between two processors whenever there is an edge between them. Our goal in this paper is to examine some algorithms by which these processors can cooperate in the solution of problems. We will be interested in what have become known as *byzantine* problems (see, e.g., Pease *et al.* [7], Toueg *et al.* [8]): we assume that messages travel reliably along each edge, that the processors have synchronized clocks, and that the time to pass a message is not a problem, but that a small number of the processors is maliciously faulty. These faulty processors may originate false messages and may change messages from other processors before forwarding them. (We assume that they do not originate so many false messages as to overload the system.) Our general strategy will be to use replication of messages (by different routes) and voting methods to get a message from one vertex to a distant one; we will not consider cryptographic methods for sending a message. Our methods are motivated substantially by Toueg *et al.* [8]. However, that paper supposed that all processes are in direct communication with each other (i.e., they form a clique). Here we suppose they form a more general graph, and work up to the question, can they identify the graph? We made a less formal report on part of these results in Ordman [5].

Suppose the graph has $n$ vertices (we now treat the words *vertex* and *process* as synonymous), of which at most $f$ are faulty (byzantine faults). Suppose the graph is $d$-connected (vertex-connected, i.e., deleting fewer than $d$ vertices cannot disconnect the graph). We know (see, e.g., Fischer *et al.* [3]) that there is no hope of successful byzantine algorithms unless $d > 2f$ and $n > 3f$.

Define the f-*distance* (motivated by "fault-tolerant" distance) between two vertices and $b$ of $G$ as (1) 0, if $a = b$; (2) 1, if there is an edge $ab$; or (3) the smallest number $s$ such that there are at least $2f + 1$ vertex-disjoint paths from $a$ to $b$, all of length not exceeding $s$. Define the f-*diameter* $d^*$ of $G$ as the largest $f$-distance between any two points of $G$. Note that this is defined only if $G$ is at least $2f + 1$-connected and it is at least as large as the usual diameter of $G$. If $G$ is a clique (every two vertices are on an edge), then $d^*$ is one.

[a]This work was partially supported by U.S. National Science Foundation grant DCR-8503922.

For example, the icosahedron is 5-connected with 12 vertices. If at most two vertices are faulty, the conditions are satisfied; it is not hard to see that the diameter is 3 but $d^* = 4$. (Observe that from each vertex to each other there are 5 vertex-disjoint paths of length not exceeding 4, so $d^* \leq 4$; to show it must be 4, suppose the faulty vertices are adjacent and note that the two vertices adjoining both of them have only two paths of length 3 between them.)

Communication will proceed by synchronous rounds. That is, we suppose that all processes have clocks that agree. In each round a process may in order

(i) receive arbitrarily many messages
(ii) compute
(iii) send arbitrarily many messages (which will be received at the start of the next round).

We make one further significant simplifying assumption. Each process has a distinct identifying number (not necessarily in the range 1 to $n$), knows its number, knows the numbers of the other processes connected to it, and knows when it receives a message who it is from. In particular, a faulty process cannot lie to its immediate neighbors about its name. In the second to fourth sections, we suppose each correct process has available to it a map of the entire graph; in the fifth section, we discuss how this can be constructed without prior knowledge. Throughout the paper we assume that all correct processes share knowledge of an integer $f$ that is an upper bound on the number of faulty processors, and that they know that the graph has at least $3f + 1$ vertices and is at least $2f + 1$-connected.

## A BROADCAST PRIMITIVE

We wish a method of transmitting messages that will assure us that correct messages will be received; that if one correct process accepts a message, it can be confident that others will; and that faulty processes cannot delude some correct process without others at least being aware that there is something going on (that is, "noticing" that an attempt is being made to pass a message). More formally, let *send* and *receive* be simple terms for the act of sending a message down an edge and receiving it. The sender of a message will always sign it and note the time it was sent, that is, the message will be of the form (message from $p_i$ at $t_j$). Then to *broadcast, hear, accept, notice* a message are distinct actions that will be described below. We wish these terms to satisfy the following properties:

*Correctness.* If a correct process $p_0$ broadcasts a message at time $t_0$, all correct processes accept (message from $p_0$ at $t_0$) by time $t_0 + 2d^*$.

*Unforgeability.* If a correct process $p_j$ accepts (message from $p_i$ at $t_0$) and $p_i$ is correct, then $p_i$ did broadcast that message at time $t_0$.

*Relay.* If any correct process accepts (message from $p_i$ at $t_j$) at time $t_k$, then all correct processes accept this message no later than time $t_k + 2d^*$.

*Notice.* If any correct process accepts (message from $p_i$ at $t_j$) at any time, then every correct process *noticed* this message no later than time $t_j + 1 + d^* \leq t_j + 2d^*$.

All correct messages sent are of one of the two forms

$$(\text{init, } \textit{message} \text{ from } p_i \text{ at } t_j), (p_i, t_j) \cdot \cdot \cdot (p_k, t_k)$$

$$(\text{echo, } \textit{message} \text{ from } p_i \text{ at } t_j), (p_k, t_k) \cdot \cdot \cdot (p_m, t_m)$$

where the number of ordered pairs after the first may be from 1 to $n$; all the $p_m$ are distinct except for the explicit repetition of $p_i$ in the first case; and the $t_m$ in consecutive ordered pairs (excluding the first) are consecutive integers. Every correct process ignores anything it receives that is not a correct message.

For example,

$$(\text{init, "hi there" from } p_2 \text{ at 5}) (p_2, 5) (p_4, 6) (p_3, 7)$$

and

$$(\text{echo, "hi there" from } p_2 \text{ at 5}) (p_3, 7) (p_1, 8) (p_4, 9)$$

are correct messages.

The front pair is called the *preface;* the other parts are *signatures.*

On receiving any correctly formed message, a correct process examines to see that it is not one of signatories, and the last time mentioned is time (now $-1$). If these conditions are met, it appends the pair (me, now) to the end and forwards the message to each of its neighbors not already listed in the signatures. (We will modify this provision in the following section.)

A process $p_n$ is said to have *received a message by* f + 1 *disjoint routes* if it receives $f + 1$ correct messages with the same preface and first signature and no $p_i$ appearing more than once in the following signatures.

A process is said to *hear* a message (a prefix and first signature) if (1) it started it itself last round; or (2) it receives it directly from a neighbor, that is, with only the one signature of that neighbor; or (3) it receives it by $f + 1$ disjoint routes.

We are now able to briefly state the broadcast algorithm itself. A process that originates a message using this algorithm is said to *broadcast* the message.

## Broadcast Algorithm

To *broadcast* a message at time $t_i$ a process $p_i$ sends (init, message from $p_i$ at $t_i$) $(p_i, t_i)$ to all its neighbors.

If a process $p_j$ hears (init, message from $p_i$ at $t_i$) at (the start of round) $t_j$, it sends (echo, message from $p_i$ at $t_i$) $(p_j, t_j)$ to all its neighbors (at the end of round $t_j$).

If a process hears any (echo, message from $p_i$ at $t_i$), it is said to *notice* the message (message from $p_i$ at $t_i$).

If a process hears the same "echo" prefix with $f + 1$ distinct first signatures, it sends a similar echo prefix (with itself as the first signatory) if it has not done so earlier.

If a process hears the same echo prefix with $2f + 1$ distinct first signatures, it *accepts* the message (message from $p_i$ at $t_i$). □

Before we prove the main desired properties of this algorithm, we give two lemmas.

LEMMA 1: If a correct process $p_i$ originates a prefix (sends init or echo with one signature) at time $t_i$, all correct processes hear it by time $t_i + d^*$.

*Proof:* $t_i$ hears it automatically; everyone connected by one edge to $p_i$ hears it on receipt; anyone further away is connected to $p_i$ by at least $f + 1$ correct vertex-disjoint paths of length not exceeding $d^*$, and so hears it after at most $d^*$ rounds. □

LEMMA 2: If a correct process accepts (message from $p_i$ at $t_i$), at least one correct process received (init, message from $p_i$ at $t_i$) $(p_i, t_i)$ at time $t_i + 1$.

*Proof:* The process that accepted the message heard the appropriate echo prefix with at least $2f + 1$ distinct first signatures, so at least $f + 1$ correct processes originated those echos. Consider the first round when a correct process originated such an echo; it can't have heard $f + 1$ echos previously, so it must have heard an init. Now one of these cases occurs.

(i) It sent the init itself. Then it sent it to at least $2f + 1$ processes so at least one correct process got the init.
(ii) It received the init directly. Since it is a correct process, at least one correct process heard the init.
(iii) It received the init by $f + 1$ disjoint routes. In that case, at least one of those paths contains no faulty processes, so the second signature in that path was of a correct process and the path was correctly reported, so that second signature gives the name of a correct process that received the init directly. □

We now prove the properties of the broadcast algorithm.

CORRECTNESS: If a correct process $p_0$ broadcasts a message at time $t_0$, all correct processes accept (message from $p_0$ at $t_0$) by time $t_0 + 2d^*$.

*Proof:* If $p_0$ sends the init prefix, all correct processes hear the init by $t_0 + d^*$, so all correct processes send an echo by then and all correct processes hear echos originating from at least $n - f > 2f$ distinct processes by time $t_0 + 2d^*$. □

UNFORGEABILITY: If a correct process $p_j$ accepts (message from $p_i$ at $t_0$) and $p_i$ is correct, then $p_i$ did broadcast that message at time $t_0$.

*Proof:* If a correct process accepts a message from $p_i$, at least one correct process heard an init directly from $p_i$. But then, since $p_i$ is correct, it must have broadcast the message. □

RELAY: If any correct process accepts (message from $p_i$ at $t_j$) at time $t_k$, then all correct processes accept this message no later than time $t_k + 2d^*$.

*Proof:* If $p_k$ accepts it and is correct, then it heard $2f + 1$ echos, so at least $f + 1$ correct processes started sending echos no later than time $t_k$. Hence, all correct processes hear at least $f + 1$ echos by time $t_k + d^*$ and all correct processes then send echos; thus, all correct processes hear at least $n - f > 2f$ distinct echos by time $t_k + 2d^*$ and accept the message. □

NOTICE: If any correct process accepts (message from $p_i$ at $t_j$) at any time, then every correct process *noticed* this message no later than time $t_j + 1 + d^* \leq t_j + 2d^*$.

*Proof:* If $p_k$ accepts this message, then at least one correct process heard an init directly from $p_l$ at time $t_j + 1$. Thus a correct process started an echo by time $t_j + 1$ and all correct processes heard it by time $t_l + 1 + d^*$. □

## THE BROADCAST PRIMITIVE, WITH RESTRICTED NUMBER OF COPIES

The algorithm of the previous section has the disadvantage that at each stage, any process receiving a message forwards it to all its neighbors who have not yet signed it. This means the number of messages grows exponentially with the number of rounds (each process has at least $2n + 1$ neighbors). If the structure of the network is adequately known, we can restrict the number of messages necessary between correct processes.

Suppose each process has a table showing (1) all other processes in the graph, and (2) for each nonadjacent process, $2f + 1$ disjoint routes to that process. For example, process 5 might know that it can reach process 7 by the routes 5–2–7, 6–3–7, 5–1–8–7, 5–6–4–7, and 5–12–9–10–7. (If these routes are as short as possible, none will have length exceeding $d^*$.) Now whenever process 5 starts a new message of the form (init, *message* from $p_5$ at $t_i$) ($p_5$, $t_i$) it actually sends one copy directly to each immediate neighbor, and $2f + 1$ copies to each other process, one along each route, and it appends the route as a part of the prefix. It does the same thing when starting an echo; the prefix might look like (echo, *message*. . . , route). Whenever a process receives a message, it notes where it is in the route; if it is the last vertex listed, it need not forward the message; otherwise, it forwards it only to the next vertex listed.

We can now estimate the number of messages involved in sending a message. Disregarding faulty processes (assuming all $n$ processes are correct), a process broadcasting a message sends it to $n - 1$ others, each by at most $2f + 1$ routes of at most length $d^*$, thus causing no more than $nd^*(2f + 1) < n^3$ transmissions; every correct process will in due course issue an echo, with each echo causing another (less than) $n^3$ transmissions, for a total of less than $n^4$ transmissions. Obviously, this system still causes a great deal of duplication (5 will apparently send the message to 7 by $2f + 1$ distinct routes, and will also send it by way of 7 many times to reach other processes), but it does establish that the number of messages is polynomial.

Unfortunately, *finding* the $2f + 1$ routes from each node to each other required by the preceding algorithm appears to be a difficult process (Even [2, chaps. 5 and 6]; Garey and Johnson [4, p. 214, "Kth Shortest Path"]). These references suggest that finding $2f + 1$ shortest paths can be done in time polynomial in $n$ for fixed $f$, or that we can find $2f + 1$ paths if we do not insist that they all have length not over $d^*$, but that finding $2f + 1$ shortest paths is NP-hard for arbitrary $f$. Luckily, finding the sets of $2f + 1$ disjoint paths need be done only once and (if each process has a map of the graph), can be done locally without message-passing. If computation is much faster than message-passing, it is well worth doing a single hard (even exponential) calculation to avoid having exponentially many messages required. Alternatively, it may be possible to give a method by which a process can analyze the message traffic it

receives to assure that no more than polynomially many need be forwarded each time by each process.

## A BYZANTINE GENERALS ALGORITHM

Because the broadcast primitive in the previous sections has been designed to have the same essential properties of that in Toueg *et al.* [8], we can solve the Byzantine Generals problem in the same way they do. No claim is made that this solution is particularly efficient; it is chosen to be conceptually simple in the present context.

Suppose that we have a graph of processes, as before; suppose that each process has a map of the graph, or as in the preceding section, that each correct process has a set of $2f + 1$ disjoint routes to each process other than itself and its immediate neighbors. Also, we suppose that each correct process knows $d^*$, the maximum length of any of the communication paths needed. Hence, the broadcast algorithm can be carried out using a polynomial number of messages.

We now describe the *Byzantine Generals* problem. Suppose that at time (round) 1 one process $p_0$, (the *general*) sends a message (we assume that this message is not NULL). We wish to establish a fixed future time by which all processes can agree on what message was sent. Note that if $p_0$ is known to be a correct process, this is easy; all processes have received it by round $2d^*$. The difficulty arises because $p_0$ may be faulty, and may send different messages to different $p_i$. We require both of the following.

*Correctness.* If the general is correct, all correct processes must decide on the message it sent.

*Agreement.* Whether or not the general is correct, all correct processes must decide on the same message.

(Note that if $p_0$ is faulty, one possibility is that the correct $p_i$ will agree on the message "NULL".)

To mimic the Toueg [8] algorithm, we organize the computation into *stages*, each stage involving $2d^*$ rounds (one round, you will recall, was of the form receive–compute–send). Thus if a message is originated at the start of a stage by a correct process, it will be accepted by all correct processes by the end of that stage.

The algorithm is this: At round 0 (the beginning) the "general" $p_0$ initiates the message, signed by $p_0$ at time 0. We regard the next round as the first round of stage 1. All processes participate in the broadcast algorithm each round thereafter. At each later stage (that is, after each round divisible by $2d^*$), each process notes if it has noticed or accepted any messages in the present stage. It then proceeds as follows.

If it is now stage $r$ and I have *accept*ed messages with identical "message" texts that were *init*iated (broadcast) by $r$ distinct processes in $r$ distinct stages (where the process broadcasting it in stage 0 was the general), I *decide* on this message, and broadcast it at the start of stage $r + 1$.

If it is now stage $r$ and I have *notice*d no message as having been broadcast by $r$ distinct processes in $r$ distinct stages, then I *decide* on "NULL" and stop.

The following lemmas are adapted from Toueg *et al.* [8].

LEMMA 3: If the general is correct and properly broadcasts *message*, then *message* is decided by all correct processes at the end of stage 1.

*Proof:* By round $2d^*$, hence during stage 1, all correct processes accept the message from the general originated at time 0. □

LEMMA 4: If any correct process decides on a non-NULL "message" at stage $s$, all correct processes do so at stage $s + 1$.

*Proof:* If $p_i$ decides at state $s$, it has accepted $s$ appropriate messages. By the relay property, every correct process will have accepted those messages by time $s + 1$ and by correctness, they'll have accepted the message from $p_i$ also. □

LEMMA 5: If any correct process decides on a non-NULL message at stage $s$, no correct process decides NULL and stops before stage $s + 2$.

*Proof:* If $p_i$ decides on "message" at stage $s$, then it accepts $s$ messages started by $s$ distinct processes at $s$ distinct stages by then. Then by the *notice* property, every correct process noticed "message" originated by $s$ distinct processes in $s$ distinct stages. Any message that will ever be accepted is always noticed by the end of the stage after which it started; hence, for each stage, a copy of the appropriate message was noticed at that stage. □

LEMMA 6: If no correct process has decided on a non-NULL message by the end of stage $f + 1$, then all correct processes have decided NULL by the end of stage $f + 1$.

*Proof:* In this case, no correct process has broadcast, so at most $f$ processes have broadcast. Thus no correct process has noticed any message that has been broadcast by more than $f$ distinct processes. Accordingly, by the end of stage $f + 1$, each correct process will decide NULL and stop. □

THEOREM 1: This algorithm satisfies the Byzantine Generals problem and every correct process decides after at most $f + 2$ stages.

*Proof:* If the general is correct, all correct processes decide on the correct message in stage 1 by Lemma 3. If the general is incorrect and any correct process decides on a non-NULL message, some correct process must do so by the end of stage $f + 1$ and then all correct processes accept that message by the end of stage $f + 2$. Finally, if no correct process decides on a non-NULL message, all correct processes decide NULL by the end of stage $f + 1$. □

Several further properties of this algorithm can be found in Toueg *et al.* [8]. One difference between our presentation and that one is that that paper considers only the single non-NULL message "1". As we need to have a variety of messages available, we make the following modification to the algorithm:

> Where the present algorithm would have me "decide" on a non-NULL message, substitute "tentatively decide". At stage $f + 2$, if I have tentatively decided on just one message, decide on it; if I have tentatively decided on more than one message, decide NULL.

THEOREM 2: If any correct process tentatively decides on more than one non-NULL message, then all correct processes do so (and the general is faulty).

*Proof:* No correct process can tentatively decide on a non-NULL message unless that message was broadcast by the general. If the general broadcast more than one non-NULL message, the general is faulty. The earlier lemmas establish that if any correct process tentatively accepts any message, all do by the end of stage $f + 2$. □

If the general broadcasts at most one message, then each correct process broadcasts at most one message; hence if we are using the broadcast algorithm of the preceding section with polynomially many messages, this Byzantine Generals algorithm requires only $n$ times as many (still polynomially many) messages. (Of course, we cannot control the number of messages sent by the faulty processes.) It would be nice to guarantee that even if the faulty processes sent arbitrarily many messages, the correct processes can limit their own messages to polynomially many. This does not seem easy in this case, when the faulty processors can originate arbitrarily many broadcast algorithms and correct processes not adjacent to the general must probably participate in many of them at least for awhile.

## A BYZANTINE GRAPH RECOGNITION ALGORITHM, PART 1

The algorithm in the last section supposes that each process knows what $d^*$ is at the beginning; to work with only polynomially many messages it requires that each process know in advance $2f + 1$ routes to each process other than itself and its immediate neighbors. In this section we consider a case when the processes do not have this advance information. We suppose that each process has a unique identifier; it knows its own identifier and those of its immediate neighbors; it knows $f$ and knows that the graph has at least $3f + 1$ vertices and is at least $2f + 1$-connected. It need not, however, know the number $n$ of vertices in the graph or have any other information besides that just given. We suppose that all processes in the graph begin participating in the algorithm at an agreed upon time, called 0. We wish all correct processes to agree, in a finite time that will become known to all, on a map of the graph. Note that we cannot hope to have the agreed graph coincide with the actual graph: if $p_i$ and $p_j$ are faulty processes, no other process can ever really know if there is or is not an edge from $p_i$ to $p_j$. For the moment, we will settle for the following.

*Agreement.* All correct processes agree on the same image of a graph $G'$.

*Partial Correctness.* $G'$ has the same set of vertices as $G$. Every edge of $G$ that has a correct process at at least one end appears in $G'$. Every edge of $G'$ that is not in $G$ has a faulty process at at least one end.

By way of motivation, let us examine the similar problem without faulty processes; assume the graph has at least one vertex and is connected. At time 0, let each process send to its neighbors its adjacency list. At each subsequent round, each process:

(1) receives some adjacency lists
(2) makes a list of each process that meets these two conditions:
  (a) it appears in at least one adjacency list that has been received
  (b) its own adjacency list has not yet been received.
  (If this list is empty, this process can stop at the end of this round)
(3) forwards each adjacency list it has received for the first time to every neighbor it has not received it from.

Note that if $d$ is the diameter of $G$, then every process will have heard from every other process after $d$ or fewer rounds; once it has heard from all processes, the stopping condition in (2) will be satisfied and it will have a complete map of the graph. The

number of messages sent is clearly polynomial, since each of $n$ vertices forwards each of $n$ adjacency lists (of length less than $n$) once each to all or a subset of its neighbors (so less than $n$ times).

This algorithm fails badly to extend to the byzantine case. A simple problem is that a faulty process may never send us its adjacency list; a more serious difficulty is that each faulty process can invent fictitious processes connected to the rest of the graph through it; a correct process may continue indefinitely to receive adjacency lists from ever more distant fictitious processes. Once we can put an upper bound on $d$ (and hence on $d^*$), we can use the broadcast and/or Byzantine Generals algorithms to help us; the hard part is knowing when to stop listening and estimate $d$.

The key to this problem is this: As long as there is a correct process we haven't heard from, there are $f + 1$ correct paths from us to it; we are getting messages forwarded along $f + 1$ disjoint paths. While a faulty process can send us a false adjacency list by $f + 1$ or more paths, it cannot cause an adjacency list from a fictitious process to reach us by more than $f$ disjoint paths, since each such path must contain a faulty process.

Before stating the algorithm, we prove a simple graph-theoretical lemma.

LEMMA 7: If $G$ is a graph with $n$ vertices and is $2f + 1$-connected, between any two nonadjacent vertices there are $2f + 1$ (vertex-) disjoint paths of length not exceeding $n - 2f - 1$. Further, there is a graph for which this is attained.

*Proof:* Let $a$ and $b$ be nonadjacent vertices. By Menger's theorem (see, e.g., Even [2]), there are $2f + 1$ vertex-disjoint paths between them. Let one of these be as long as possible. The other $2f$ paths each use at least one vertex of $G$, and $a$ and $b$ occupy vertices. Thus at most $n - 2f - 2$ vertices are available to be interior vertices of the long path, and it consists of at most $n - 2f - 1$ edges. To see that this can happen, join all points of a cycle with $n - 2f + 1$ points to all points of a complete graph on $2f - 1$ points. To construct $2f + 1$ disjoint paths between points that are two apart on the cycle, we have the following: the path of length 2 on the cycle; $2f - 1$ paths of length 2 via the complete graph; and the long path around the far side of the cycle, of length $n - 2f - 1$. □

Below we give an algorithm that determines the size (number of vertices) in a graph with byzantine failures. The control variable $H$ that gets us out of the main loop is defined as follows: $H$ is the greatest number such that there are $H$ processes such that:

(a) we have "heard" an adjacency list from each of them
(b) each reports the existence of a neighbor from which we have not yet heard an adjacency list
(c) there is a set of messages we have received, one from each of these $H$ processes, by $H$ mutually disjoint paths.

A BYZANTINE GRAPH SIZE ALGORITHM

For each process: At round 0:
BROADCAST my own adjacency list (using the broadcast primitive of the second section, broadcast a message consisting of "$p_i$ is adjacent to $p_j, \ldots, p_k$");

Store a "candidate set" of vertices and of adjacency lists, each of which may be marked "heard" or "expected". Initially this set consists of myself and my own adjacency list, marked "heard".

At round $t > 0$:

Participate in the BROADCAST algorithm;

When a new adjacency list is "heard", add that vertex and adjacency list to the candidate set.

Compute $H :=$ the number of vertices that have sent me adjacency lists via disjoint paths whose "heard" adjacency lists contain vertices not yet "heard".

IF H > f THEN

For any vertex previously marked "expected" but not yet "heard", mark it "heard" {it is faulty};

For any vertex included in "heard" adjacency lists of at least $f+1$ vertices but not yet "heard", add it to the candidate set marked "expected";

ELSE {if H $< f+1$}

n′ := number of "heard" vertices;
d′ := n′ − 2f − 1;
IF t > d′ THEN EXIT (this algorithm) ENDIF;
ENDIF. □

We need to show that the "candidate graph" produced in each correct process by this algorithm has all the vertices of $G$ and only those of $G$; that all the correct processes thus agree on $n = n'$ and on $d'$; and that all correct processes determine $d'$ no later than round $d'$. Notice that the adjacency lists "heard" may differ from process to process and may not for any process represent a reasonable graph; a faulty process may have more than one adjacency list "heard", and it is possible that $p_i$ reports $p_j$ is adjacent to it but $p_j$ claims $p_i$ is not adjacent to it.

LEMMA 8: No correct process stops before it has heard a message from every correct process.

*Proof:* We show that at round $t > 0$, if correct process $p_i$ has not heard from correct process $p_j$, then $p_i$ has $H$ at least $f + 1$. Suppose $p_i$ has not heard a message from $p_j$. Then there are at least $f + 1$ disjoint paths from $p_j$ to $p_i$ passing through correct processes. On each of those paths there is a closest process to $p_j$ from which $p_i$ has heard a message; these are all distinct. (By round 1, $p_i$ has heard from all the processes adjacent to it, hence from one on each of these paths.) Thus there are at least $f + 1$ vertices that have been heard from that report adjacency to vertices ($p_j$ or something closer to $p_i$) not yet heard from, and $H$ is at least $f + 1$. □

LEMMA 9: Every correct process hears a message from every other correct process by time $d^*$.

*Proof:* Given correct processes $p_i$ and $p_j$, they are adjacent or there are $2f + 1$ disjoint paths between them with the longest not exceeding length $d^*$. Hence, the

message sent by $p_i$ at the start of the algorithm reaches $p_j$ by at least $f + 1$ routes within that time. □

LEMMA 10: By time $d^* + 1$ every correct process will have every vertex of $G$ marked "heard" in its candidate graph.

*Proof:* By time $d^*$ it will have heard an adjacency list from every correct process. But each faulty process has at least $f + 1$ correct processes adjacent to it; so if no message from that faulty process has been heard by time $d^*$, it will become "expected" no later than that round and will be marked "heard" the following round. □

LEMMA 11: No process not in $G$ will ever be marked "heard" by a correct process.

*Proof:* A correct $p_i$ may receive messages purporting to be originated by fictitious vertices, but each such message must have come from a faulty process and must therefore have a signature of a faulty process among its signatures; hence, it can arrive by at most $f$ disjoint routes and will not be "heard". A vertex marked "heard" by first being marked "expected" must be adjacent to $f + 1$ "heard" processes, at least one of which must be a correct one; so it must really be in $G$. □

LEMMA 12: At all times after $d^*$, $H$ will be $f$ or less for every correct process.

*Proof:* By this time the candidate graph will have all correct nodes and all faulty nodes marked "heard". No message will have been heard from a correct node except a correct message (for no incorrect message can have arrived by $f + 1$ disjoint routes), so at most $f$ processes (the incorrect ones) can have sent adjacency lists that have been heard and report not-yet-heard (hence fictitious) vertices. □

THEOREM 3: By time $d^* + 1$, all correct processes will have a list of precisely the vertices of $G$; all will have the same value of $n'$ and all will have the same value of $d'$ that is no less than $d^*$.

*Proof:* By the lemmas, each correct process will have the same correct list of vertices, hence the same $n'$ and $d'$; by Lemma 7, $d'$ will be no less than $d^*$. □

One could make this proof slightly more sophisticated. For example, any vertex from which more than one message was "heard" could be marked "faulty"; and a vertex that became "expected" and then "heard" without a message actually being heard could also be marked "faulty". It is not immediately obvious that everyone else would know that these were faulty, however, so one might not be able to completely ignore them thereafter. On the other hand, by the round after $H$ falls below $f + 1$, the correct process has a complete list of vertices; thereafter, messages from or concerning fictitious vertices can be safely ignored (and not forwarded).

## A BYZANTINE GRAPH RECOGNITION ALGORITHM, PART 2

By the previous section, all correct processes can agree by time $h - 2f - 1$ on a list of vertices of the graph, the value of $n$, and a value $d' = n - 2f - 1$ that is at least as large as $d^*$. They can now continue the exchange begun in the size algorithm of the last section, as follows.

At time 0 each correct process broadcasts its adjacency list and proceeds as in the size algorithm. Once $d'$ is agreed upon, it proceeds to use it as $d^*$ in the broadcast and Byzantine Generals algorithms. With respect to each of the $n$ processes, the processes now engage in a Byzantine Generals algorithm with respect to the adjacency list broadcast by that process. (The default, if no adjacency list is decided upon for a process, or if more than one adjacency list is decided upon for a process, is to decide on the NULL adjacency list for that process.) Upon completion of the Byzantine Generals algorithm (after at most $f + 2$ stages each of $2d'$ rounds), each correct process will have the same set of vertices and adjacency lists in its candidate graph. It then converts this set of adjacency lists into the adjacency list of a graph by including in the graph any edge reported from either end: that is, if $p_i$ reports it is adjacent to $p_j$, then $p_i$ should be added to the $p_j$ adjacency list if it is not already there. □

THEOREM 4: The graph $G'$ resulting from the preceding algorithm satisfies the conditions of agreement and partial correctness.

*Proof:* The graph $G'$ is the same for all correct processes; it has the correct vertices; it has all edges of $G$, except possibly those connecting two faulty vertices; it has an edge not in $G$ only if that edge was reported by a faulty process and hence has a faulty vertex at at least one end. □

We would like to be able to make a stronger claim about the graph $G'$. Can we now replace $d'$ (which is typically much too large) with a correct value of $d^*$? Can we compute, for each pair $p_i$ and $p_j$ of processes, a set of $2f + 1$ (or some other number) of disjoint paths guaranteed to contain $f + 1$ correct paths, so that we can in subsequent calculations restrict the number of (correct) messages to be polynomial in $n$? Here is a condition that might be attractive:

*Algorithmic Adequacy.* For each pair of correct processes $p_i$ and $p_j$ of $G'$, either (1) they are adjacent in both $G'$ and $G$; or (2) they are nonadjacent in both, and there are $2f + 1$ disjoint paths between them in $G'$ of which at least $f + 1$ are paths in $G$ and do not pass through faulty vertices of $G$.

The intent of the *algorithmic adequacy* condition is that once those $2f + 1$ paths are found, subsequent broadcast algorithms can be done in polynomial time. No claim is made that the sets of paths described in this condition can be found in polynomial time. While the fact that these paths can be found will allow the calculation of an analog of $d^*$ for $G'$, no claim is made as to its relationship to the original $d^*$ of $G$. It might well be possible to find fewer than $2f + 1$ paths in $G'$ and have them contain $f + 1$ correct paths of $G$. Finally, we cannot prove the *algorithmic adequacy* condition without strengthening our hypotheses on $G$ and do not know if the strengthening is required; so our solution is now rather unsatisfactory.

THEOREM 5: If (in addition to the usual hypotheses), the graph $G$ is $\lfloor 5f/2 \rfloor + 1$-connected, then the graph $G'$ satisfies the condition of algorithmic adequacy.

*Proof:* By partial correctness, two correct processes are adjacent in $G'$ if and only if they are adjacent in $G$. Let $p_i$ and $p_j$ be nonadjacent in $G'$, so they are nonadjacent in $G$. By hypothesis there are $2f + 1 + \lfloor f/2 \rfloor$ paths from $p_i$ to $p_j$ in $G$. Each of these paths is

a path in $G'$ unless one of its edges is lacking from $G'$; but an edge of $G$ cannot be lacking in $G'$ unless it connects two faulty vertices. Since neither $p_i$ nor $p_j$ is faulty, one of our paths can be absent in $G'$ only if it contains two faulty vertices in $G$; thus at most $\lfloor f/2 \rfloor$ paths are absent. The remaining $2f + 1$ paths are present in $G'$. Further, any $2f + 1$ paths in $G'$ involve at most $f$ faulty vertices; so $f + 1$ are paths between correct vertices only, and thus all their edges are present in $G$ as desired. □

We could now determine a value corresponding to $d^*$ and a set of paths allowing communication with polynomially many (correct) messages as follows: For each pair of vertices $p_i$, $p_j$, find a collection of $2f + 1$ disjoint paths between them in which the longest path is of minimum length. (If $2f + 1$ paths cannot be found, at least one of $p_i$ or $p_j$ is faulty; drop the pair from further consideration.) Let $d'^*$ be the longest path in the union of these sets.

Now a message will always travel from any correct node to any other by $f + 1$ disjoint paths within $d'^*$ steps, as desired. Each vertex need only send $2f + 1$ "addressed" copies of each message to a particular other vertex; if it receives a message with instructions to forward it immediately to a vertex to which it has no edge (i.e., its forwarding instructions require use of an edge of $G'$ that is not in $G$), then the vertex to which it is asked to forward it is faulty and it need not forward that message.

## FURTHER REMARKS

Several issues raised herein suggest a need for further study.

Given a graph $G$ with known adjacency matrix, how should we approach in practice finding $2f + 1$ disjoint paths between each pair of vertices? There are well-known algorithms for finding a collection of paths, using maximum-flow algorithms [2], but finding the shortest such paths is reportedly NP-complete [4]. The wide broadcasting strategy in which every process broadcasts to every other by every possible route leads roughly to $n$ messages each with nearly $(n - 1)!$ signature strings before we gain enough information to even partially restrict things. Clearly, the earlier we can cut down on duplicate messages, the better.

Can we do better on the connection between $G$ and the agreed-upon image $G'$? Broadcasting in $G'$ by the wide broadcasting strategy will get through (all the edges of $G$ are present): Is there really no strategy that will get through with only polynomially many messages? This seems most unlikely.

Finally, the algorithms presented here depend on having all processes begin together at time 0. In [6] I eliminate that requirement by using techniques adapted from Burns and Lynch [1].

## REFERENCES

1. BURNS, J. E. & N. A. LYNCH. 1985. The byzantine firing squad problem. Rep. MIT Laboratory for Computer Science, Apr. 1985. (To be published in Advances in Computing Research.)
2. EVEN, S. 1979. Graph Algorithms. Computer Science Press. Potomac, Md.
3. FISCHER, M. J., N. A. LYNCH & M. MERRITT. 1985. Easy impossibility proofs for distributed consensus problems. *In* Proceedings 4th ACM Symposium on Principles of Distributed Computing (Minaki, Ont., Canada), pp. 59–70.