Unanimity in an Unknown and Unreliable Environment

by
D. Dolev†
Computer Science Department
Stanford University

Abstract

Can unanimity be achieved in an unknown and unreliable distributed system? We analyze two extreme models of networks: one in which all the routes of communication are known, and the other in which not even the topology of the network is known. We prove that independently of the model, unanimity is achievable *if and only if* the number of faulty processors in the system is

1. less than one half of the connectivity of the system's network, and
2. less than one third of the total number of processors.

In cases where unanimity is achievable, an algorithm to obtain it is given.

§1 Introduction

The major issue analyzed in this paper is *reaching unanimity in an unreliable distributed system.* Unanimity means complete agreement by every pair of processors on the value of every processor in the system. We look for unanimity even if the topology of the network is not known to the individual processors. The major task is to circumvent errors without losing unanimity. This can be achieved if all the reliable members of the system agree upon the content of the messages in the system. They have to agree on especially those messages corresponding to the faulty parts of the system, even if the faulty parts cannot be uniquely identified.

The assumption is that a faulty processor can do whatever it likes. Thus, a faulty processor can behave very strangely: It can alter the information relayed through itself; it can block such information from being relayed; it can incorrectly reroute the information, and in the worst case, it can send conflicting information to different members of the system. These are examples of malfunctioning behaviors; in general we would like not to restrict ourselves to certain assumptions about faulty behavior.

Some of the processors may consider a faulty processor to be a reliable one and a reliable processor to be faulty. Obviously, there is a limit to the number of faulty processors a system can tolerate. The problem of achieving unanimity is further complicated by the following compound question:

> Under what conditions does unanimity remain valid and what is the system's unanimity threshold?

In the general case one does not know which processors are faulty. Moreover, in most cases one will never be able to find it out. To understand the reason for this, imagine yourself as a processor in a distributed system that receives a message from a processor z. Assume that you want to make sure that z is reliable. So, you inquire what are the messages the rest of the system received. You find out that the message you have received differs from all the rest. *Is z a faulty processor?* The answer is not necessarily positive. The possibility that the only reliable processors in the system are you and z always exists.

To overcome this logical pathology you should make your decision assuming an upper bound on the number of faulty processors in the system. That is, if the system contains more faulty processors than that upper bound, it will no longer matter what decision you are going to make. Throughout this paper t denotes the upper bound on the number of faulty processors in the system. Knowing t a receiver can sometimes determine that the sending processor is unreliable. This can be achieved in cases of "too many" conflicting versions of the sender's message.

Unfortunately, in many cases faulty processors can

† This work was supported in part by Chaim Weizmann Postdoctoral Fellowship.
Current address: IBM Research Center, San Jose Laboratory, 5600 Cottle Rd., San Jose, CA 95193.

CH1695-6/81/0000/0159$00.75 © 1981 IEEE

still remain anonymous and indistinguishable from the reliable ones. Consider first the problem of a processor, say z, that sends its message to every other processor in the system. If more than one message is received, the reliable processors have to determine and agree upon which value was actually sent by z. Consider the following agreement that has to be achieved in the presence of faulty processors:

B1) All the reliable processors agree on the same message.

B2) If z is reliable, then all the reliable processors agree on its message.

This agreement was named the Byzantine Generals problem by Lamport, Shostak and Pease [LSP80]. Here it will be refered to as the *Byzantine Agreement*.

A characterization of a network's tolerance to faults for a given type of agreement and for a given type of network is t, the upper bound on the number of faults a network can sustain while still reaching the agreement. The upper bound t should depend on the topology of the network. For example, if the network graph is a tree, it is clear that almost every faulty processor can prevent the system from achieving the Byzantine Agreement.

The first work toward finding the threshold t for some networks was done by Pease, Shostak, and Lamport and is described in their paper "Reaching Agreement in the Presence of Faults" [PSL80]. In that paper and in an extended one, "The Byzantine Generals Problem" [LSP80], the authors analyze the problem of reaching the Byzantine Agreement in a complete network in which every pair of processors are directly connected. They present an algorithm to circumvent $(n-1)/3$ faulty processors, where n is the number of processors in the system. They proved that this threshold is tight, that is, one cannot guarantee a unique agreement if the number of faulty processors is a third or more. Lamport, Shostak, and Pease [LSP80] found another algorithm for a special type of networks that they call a p-regular network, but in this case the result is no longer tight. In a later paper Lamport [L80] studied the complete network with respect to another type of agreement called "The Weak Byzantine Generals Problem." In recent work ([D81]) the threshold is determined for two types of agreements and for specific assumptions about the networks.

In this paper we consider two extreme cases: networks in which all the routes of communication are predetermined, and networks in which neither the routes nor the topology are known to the processor. For every such network, G, we will prove:

> The Byzantine agreement can be achieved in a network G if and only if:
>
> 1. t is less than one half of the connectivity of G; and
> 2. t is less than one third of the total number of processors in G.

This result implies that the threshold for t holds for other assumptions about networks between these two extremes. By connectivity we mean a vertex-connectivity, that is, the minimal number of vertices the removing of which disconnects the network to at least two parts. The above threshold can be relaxed a little bit if we assume that all the processors are supposed to have the same value.

§2 Basic Notions and Assumptions

A network is composed of processors connected by links. The processors communicate by sending information via messages. Assume that every processor in the network knows its direct neighbors and can identify which of them forwarded which message.

For a clear representation of the relationships among the processors the following notions are used. A sender of messages is called the *transmitter* and the messages it sends carry its *value*. The transmitter sends its values to its *receivers* either directly or through other processors called *relays*. A processor can be a transmitter, a receiver or a relay according to its function in the network with respect to a given message. A processor is *reliable* if it transfers the messages it has received without altering or eavesdropping on them. A *reliable transmitter* is a reliable processor that sends the same value to all its receivers. A *faulty processor* is a relay or a transmitter that is not reliable.

Let us concentrate on the case of a single processor that sends its value to the rest of the network. Solving this case successfully will enable us to handle the general case in which every processor sends its own value to every other processor.

Messages should contain some information about the route through which they were relayed. Otherwise, a receiver is not able to distinguish among messages and therefore is unable to discriminate between good and faulty processors. Let the information about the route

be the list of the processors which actually relayed the message.

We can assume that every relay appends its name to the message. However, a faulty processor can compose an imaginary route and exchange it with the real route appended to the message. Moreover, it can eliminate its own name from the route; and thus is able to thwart any effort to be identified as a faulty processor.

This argument shows that we cannot trust individual processors in composing the route. To ensure the inclusion of a faulty processor's name, assume that after a relay receives a message to relay, it first appends to the message the name of its forwarding neighbor. Only then does it relay the message. But now, if a faulty relay produces many false copies of the message it is supposed to relay, and sends them through various routes, then its name appears on every one of the routes.

Throughout all the above discussion one issue was omitted: *synchronization*. Consider an asynchronous distributed system. In a reliable asynchronous system there should be an upper bound on the amount of time between sending and receiving of a message. If the transmitter sends several messages to the same receiver, then, in a reliable system, the above upper bound determines the maximum length of the time-interval during which all the messages should arrive. In an unreliable system every message that has not arrived on time can be ignored, because it was relayed (delayed) by a faulty processor.

The ability of a faulty processor to disrupt may appear unlimited, but the situation is not so bad. Due to the Menger Theorem [H72], if the connectivity of the graph is k, then there exist at least k disjoint paths between every pair of processors. This shows that the transmitter has the ability to transmit k copies of its value through k disjoint routes to every receiver.

Summary of assumptions:

P1. The processors are arranged in a k connected bidirectional network.

P2. Every processor knows the members of the network.

P3. Processors communicate only by sending messages along links.

P4. Every processor can identify the neighbor from which it receives each message.

P5. A message includes the route through which it was delivered.

P6. A reliable processor relays a message only after it appends the name of the processor from which it received the message to the message's route.

P7. A reliable processor relays messages without both altering them and eavesdropping on their values.

P8. There exists an upper bound on the delay of relaying a message by a reliable processor.

P9. There exists an upper bound, t, on the number of faulty processors in the system.

In the rest of the paper we will consider two extreme models: First model, a **Routed–Network** in which every receiver knows beforehand the routes through which the transmitter will send the messages; Second model, a **Unknown–Network** in which a processor knows neither the topology of the network nor the routes through which it will receive messages.

It is clear that the freedom that a faulty processor has in an Unknown–Network is much more than that in a Routed–Network. Therefore, it should be harder to reach unanimity in an Unknown–Network. We will present algorithms to achieve unanimity in both models. The surprising fact is that the conditions on the topology of the network for the existance of unanimity is the same in both cases. This means that for a very wide range of network types the threshold under which no unanimity can be achieved is the same.

§3 Reliable Transmitter

Consider first a reliable transmitter, that is, a transmitter which sends the same value to every receiver. In this case it is easy to achieve unanimity. Notice that in the general case the receiver does not know if the transmitter is reliable or not. This special case is needed for the general algorithm.

3.1 Reliable Transmitter in a Routed–Network

In a Routed–Network a transmiter can send messages along disjoint routes. This increases the probability of the receivers getting the correct values. The receiver can compare the actual routes, along which it received the messages, to the planned ones; thus it can

eliminate some of the problems that a faulty processor might cause.

A receiver may receive many messages from the transmitter carrying several values. The following algorithm provides the receiver with a method to select the correct values out of all the copies it has received. Define $\emptyset$ to be the empty value this means that either the transmitter did not send any value or that there is no way to find a unique value out of the set of copies. This last case arises when the transmitter is a faulty processor that has sent too many conflicting values.

Algorithm *Find* $(t; a_1, \ldots, a_r; x)$

1. Delete every message a_i whose route does not agree with the planned route. Call the messages that are left the *good messages*.
2. If there are $t+1$ good messages that carry the same value, then set the *found* value to be their value.
3. Otherwise, set the found value to be $\emptyset$.

The following theorem proves that the Find Algorithm enables a reliable transmitter to send its value faithfully. Throughout the rest of the paper G denote a network (either routed or unknown) of connectivity at least $2t+1$, where t is the upper bound on the number of faulty processors.

Theorem 1. *Let G be a Routed–Network. If a reliable transmitter sends $2t+1$ copies of its value to every receiver through disjoint paths, then every reliable receiver can obtain the transmitter's value.*

Proof: Every message contains a route that is supposed to be the route through which it was delivered. If the route is entirely composed of reliable relays, then its route will be the planned route and will be the actual route along which the message was delivered. If the planned route contains faulty processors then several things might happen. Assumption P6 implies that at least the name of the last faulty processor which relayed the message will appear on the message's route. The algorithm ensures that all messages with non matching routes are ignored.

Thus, the number of good messages a receiver has to consider is bounded by $2t+1$. If the reliable transmitter sends $2t+1$ copies of its value through disjoint routes and the number of faulty processors is less than $t+1$, then the number of good messages is at least $t+1$; and by P8 they will be received by the receiver. Moreover, there are at most t wrong values and at least $t+1$ correct values. This completes the proof. ∎

Notice that in a Routed–Network no "false alarm" can happen; that is, if the reliable transmitter did not send anything, then the Find Algorithm would not provide any value different from $\emptyset$ to any reliable processor. Assumption P8 is not essential for the proof of Theorem 1; it is sufficient to assume that eventually every message will arrive. Assumption P8 will be used in later sections to enable the receivers to know when to start the next step of their algorithms.

3.2 Reliable transmitter in an Unknown–Network

An Unknown–Network requires much more effort for ensuring a correct reception. Neither the transmitter nor the receiver knows the topology of the network; therefore, no preplanning of routes can be done. One method to establish a faithful transmission using the connectivity of the network is broadcasting. The transmitter can send the message to its neighbors and they can relay it to their neighbors and so on. However, this method must be carefully handled in the presence of faulty processors.

Another issue that produces problems is the effect of a flood of faulty messages. The transmission should be a finite process. We cannot prevent a faulty processor from sending an infinite number of messages, but we have to provide the reliable processors with a method to ignore this behavior and to be able to determine the correct value after a finite number of steps.

The following algorithm recursively describes the broadcasting method. Processor x forwards a message that carries the value a and the route ρ, to processor y.

Algorithm *Send* $(x; a : \rho; y)$

1. IF ρ contains only processors which are members of the system and every processor appears at most once, THEN
   a. $\rho \leftarrow \rho x$;
   b. ADD $a : \rho$ to the messages received from $head(\rho)$ by y;
   c. FOR every neighbor z of y which is not in ρ DO Send$(y; a : \rho; z)$ OD ;.
2. ELSE ignore the message.

Notice that the transmitter's name is the first name in ρ, and if ρ is empty, then it is x.

The following lemma describes the basic properties of the Send Algorithm. Define two mesages to be *independent* if the sets of processors which relayed the messages are disjoint. A set of messages is independent if every pair of elements of that set are independent.

Lemma 2. *Let G be an Unknown-Network, the connectivity of which is at least $2t+1$. If a reliable transmitter v sends to every neighbor y the value a, then after a finite time every reliable processors will receive at least $t+1$ independent copies of a message carrying the value a.*

Proof: Transmitter v is reliable and it sends the value a by Send$(v; a : \emptyset; y)$ to each one of its neighbors. Assumption P2 implies that every processor knows the set of the members of the system. There are at least $t+1$ independent routes between x_0 and every reliable receiver which contain only reliable relays. Assumption P8 implies that after a finite time every reliable relay will receive the message and will relay it to the rest of the processors. From these arguments it is easy to conclude the proof. ∎

The Send Algorithm and Assumption P8 imply that there exists a time interval T during which all messages sent and relayed by reliable processors should arrive. In the final algorithm we will use this time interval dynamically to sample arriving messages to determine when a reliable transmitter sends a value and what the value is.

The next issue is the method by which a reliable processor is able to choose the right value out of all the messages it receives.

Algorithm *Purifying* $(t; a_1, \ldots, a_r; x)$

1. If there exists a subset of $\{a_1; \ldots; a_r\}$ containing $t+1$ independent messages all of which carry the same value, then set the *purified value* to be that value;
2. Otherwise, set the purified value to be $\emptyset$.

Notice that if more than one set of $t+1$ independent messages exists, then there may be many purified values. But because of the way the algorithm will be used, a plurality of possible values will not be a problem; any value can be choosen in this case.

The basic property of the Purifying Algorithm is elimination of fictive values, as proved by the following lemma.

Lemma 3. *Assume that the number of faulty processors is bounded by t. The Purifying Algorithm yields a nonempty value only if such a value was actually sent by the transmitter.*

Proof: If the maximal number of independent messages is not greater than t, then the Purifying Algorithm provides an empty value. If the transmitter does not send anything, then the only messages received should be produced by faulty processors. The messages may be relayed by reliable ones, but observe that the technique used in the Send Algorithm of adding the names of the processors along the route to the message, enables x to differentiate among the values. Every message which passed through faulty processors contains at least one name of a faulty processor; more precisely, every list of processors added to a message contains at least the name of the last faulty processor that relayed it. Thus, the faulty processors cannot produce more than t independent messages, which completes the proof. ∎

The following theorem proves that in an Unknown-Network with sufficient connectivity all the reliable receivers obtain the same value, if the transmitter is reliable.

Theorem 4. *Let G be an unknown-network of processors containing at most t faulty processors, and the connectivity of which is at least $2t+1$. If a reliable transmitter broadcasts its value to all its neighbors by the Send Algorithm, then, by use of the Purifying Algorithm, every reliable receiver can obtain the transmitter's value.*

Proof: The reliable transmitter uses the Send Algorithm to broadcast its value. It sends the same value to all receivers. Let $\{a_1, \ldots, a_r\}$ be the set of all the copies of the transmitter's value that receiver x receives. There are at most t faulty processors. By Lemma 2, at least $t+1$ of the received copies are independent and carry the original value. Note that if the transmitter were a faulty processor, then the above reasoning would fail to hold.

Usually the number of copies received is much more than $t+1$ and it may even be that the majority of them carry a faulty value. The task of receiver x is to find the correct value out of this mess. It does this

by applying the Purifying Algorithm. The Purifying Algorithm looks for a set of $t+1$ independent messages with the property that all the values carried by these messages are the same. The network contains at most t faulty processors, and the transmitter is not one of them. Therefore, as in Lemma 3, the faulty processors cannot produce more than t independent messages. Thus, the only possibility of finding such a set is to include a message carried only by reliable processors, which implies that the purified value will be the value of the reliable transmitter. This completes the proof of the theorem. ∎

In the case where the transmitter is faulty, Theorem 1 and Theorem 4 do not ensure the ability to reach a unique agreement on a value. This case is discussed in the rest of the paper.

Since the receivers do not know when the transmitter starts to broadcast its value, they are not able to know when to apply the Purifying Algorithm. To enable the general algorithm to proceed receivers should try to apply the Purifying Algorithm several times, until a nonempty value is obtained. The following corollary ensures that this sampling method will not produce a wrong value.

Corollary 5. *Under the conditions of Theorem 4, the application of the Purifying Algorithm to any subset of the messages sent by the reliable transmitter will produce either Ø or the transmitter's value.*

Proof: By Lemma 3, a nonempty value will be produced only if the transmitter actually sent one. The transmitter is reliable and therefore sends the same value to everybody. A set of $t+1$ independent messages must contain a message which was forwarded only by reliable processors, and therefore carries the transmitter's value. Since all the independent messages should carry the same value to produce a nonempty value, the only possibility is that each one of them carries the transmitter's value. ∎

§4 The Necessary Conditions

In terms of the communication vocabulary defined in Section 2, the Byzantine Agreement becomes:

Byz1) All reliable receivers agree on the same value.

Byz2) If the transmitter is reliable, then all reliable receivers agree on its value.

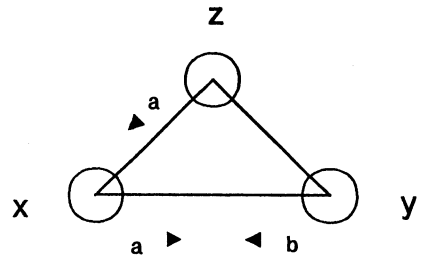

Figure 1: Three processors with one faulty

Thus, to prove the necessary condition what has to be shown is that if t is not less than half the connectivity or not less than the third of the total number of processors, then the Byzantine Agreement cannot in general be achieved. It is sufficient to show that the Byzantine Agreement cannot be achieved in a routed network, because this implies the proof for the unknown-network.

The following example clarifies the difficulties of reaching an agreement in the case $t \geq n/3$. Assume the network contains only three processors connected in a triangle and contains exactly one faulty processor. Let z be the transmitter and x be a processor which tries to follow the Byzantine Agreement. Figure 1 shows the information x gets.

Receiver x gets the value a directly from the transmitter and a different version of the transmitter's value, say b, from y. The receiver has to consider two possibilities:

1. the transmitter z is reliable and y is a faulty relay; and
2. the transmitter z is faulty and y is a reliable relay.

Now, x can decide: "faulty transmitter" (i.e. Ø) or it can choose a value. It does not know if the situation is 1 or 2. Therefore, if it decides "faulty transmitter" it might be that the situation is 1 which contradicts Byz2. This implies that it cannot decide "faulty transmitter" in this case. Processor x should choose the value a, otherwise it may fail to obey Byz2 in the case 1. Assume that the actual case is 2; therefore as a reliable processor, y faces the same problem: it receives b from z and a from x. The above arguments

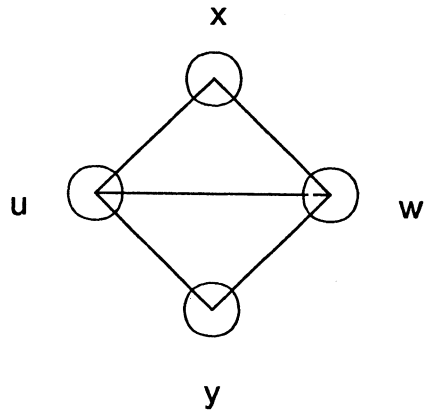

Figure 2: A two-connected network

imply that y will choose b to be the transmitter's value. These forced decisions contradict Byz1. This proves that no matter what decision the processors make, the Byzantine Agreement cannot be achieved.

By simulating the case of 3 processors one can prove the following lemma.

Lemma 6. *No unanimity can be achieved in an unknown-network of n processors if the number of faulty processors is greater than or equal to a third of the total number of processor.* ∎

Figure 2 shows a two connected network of four processors. Similarly to the triangle case (Figure 1), one can show that no unanimity can be achieved in this network. Moreover, even knowing that the faulty processor is either u or w would not help. The main reason for that is that the faulty processor can behave as a "filter" such that x and y always get different values. The fact that u and w separate x from y prevents them from deciding who is the faulty processor. Thus, they cannot decide which is the right value.

Lemma 7. *No unanimity can be achieved in an unknown-network of n processors if the number of faulty processors is not less than half of the connectivity of the network.*

Proof: Assume to the contrary that there exists a network G with n processors and connectivity k, and that there exists an algorithm to achieve the Byzantine Agreement in G for every distribution of up to t faulty processors in the network, where $k/2 < t \leq k$. Observe that if $t \geq k$, then the lemma obviously holds. The impossibility of this situation will be proved by constructing the Byzantine Agreement for the network of Figure 2.

Let $\{v_1, \ldots, v_k\}$ be a set of k processors that separates the network into two parts. Let X and Y be the sets of processors of these parts. Divide the sets of processors $\{v_1, \ldots, v_k\}$ into two parts, such that each set contains at most t processors. Denote by H the reduced network obtained from G by contracting all processors in each of the above four sets. The new network H is a network of 4 processors, in which processors U and W represent up to t processors of the original network G. Reaching the Byzantine Agreement in the presence of one faulty processor in H can be simulated by reaching the agreement in the network G, in the presence of up to t faulty processors. Similarly, the assumed algorithm for reaching the Byzantine Agreement in G can be simulated in order to reach the Byzantine Agreement in H. But this contradicts our previous arguments. The constructed contradiction proves the lemma. ∎

The necessary part of the main result is implied by the above lemmas. The rest of the paper is the proof of its sufficiency for the case of an Unknown–Network. We are only interested in existance and not in complexity; therefore, it is enough to observe that the solution for a Known-Network follows from that of an Unknown-Network. The first case can be solved directly using a much simpler algorithm such as the one for obtaining the Crusader Agreement in [Do81].

§5 The Byzantine Agreement for Unknown–Network

To obtain the Byzantine Agreement in an Unknown–Network the following recursive algorithm can be used. Throughout this section we use the following notation: Let G be a network of n processors with connectivity at least $2t + 1$; let $U \subseteq V$ be a subset of the set of processors of G; and let the current initiator of the algorithm, v, be a processor not in U. Let π be a simple path of processors in $G - U - \{v\}$. The path π describes the sequence of processors which initiated the algorithm before the current initiator.

To simplify the analysis of the algorithm's behavior we will limit it to the case in which every processor is eligible to initiate the algorithm at most once for each π. To ensure that we assume:

i. A reliable processor initiates the algorithm with $\pi = \emptyset$ at most once;

ii. A reliable processor intiates the algorithm with $\pi \neq \emptyset$ only according to step 3a of the algorithm.

The content of the messages exchanged during the algorithm is composed of the original value together with the current index m, with the set of receivers U, and with the path of initiators π. We say that a message is *erroneous* if $m > n$, or if $U \not\subseteq V$, or if π is not an *appropriate path*; that is, either π is not a simple path in $G - U - \{v\}$ or the transmitter v had previously initiated the algorithm with the same path π.

The algorithm uses a function MAJORITY, that each processor uses for deciding what the value is, given a set of received values. The function has to be such that it points out the majority value if there exists exactly one; otherwise it can get an arbitrary (but consistent) value.

Algorithm BG(G, v, U, π, t, m):
(the Byzantine Algorithm)

1. The transmitter v broadcasts a message containing its value together with the set U, the path π, and the index m through all its neighbors, using the Send Algorithm.

2. Denote by $A(v, m, U, \pi, u)$ the set of non-erroneous messages sent by the transmitter v, carrying the same U, m, and π, and arriving within time interval T, that receiver u has received. Each receiver applies the Purifying Algorithm to every set $A(v, m, U, \pi, u)$ it receives from the transmitter to find the value, a, that the transmitter intended it to receive.

3. IF $m > 0$ and $a \neq \emptyset$ THEN :

- **a.** For every $u \in U$, let a_u be the value receiver u has obtained in step 2 and π be the associated path. Receiver u acts as the transmitter in the Byzantine Algorithm BG$(G, u, U - u, \pi \cdot v, t, m - 1)$ to send the value a_u to every other receiver in $U - u$.
- **b.** For each $w \in U$ and each $u \neq w$ in U let $a_w(u)$ be the value receiver w receives from receiver u in step a. If no value is received, then set $a_w(u) = \emptyset$. Let $a_w(w)$ be the value receiver w has received from the transmitter v in step 2. Receiver w determines the value of v by MAJORITY$\{a_w(x) \mid x \in U\}$.

To prove the validity of the algorithm BG observe that

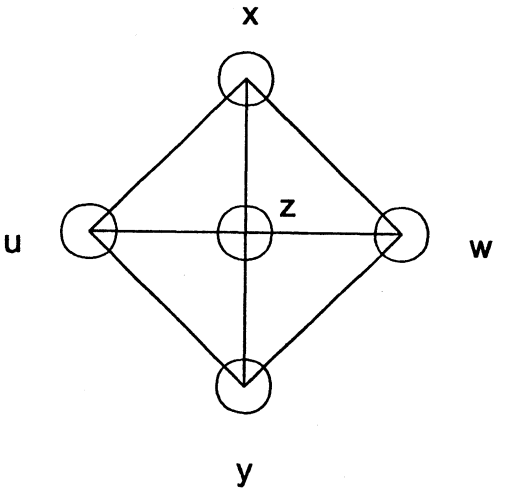

Figure 3: five processors and one faulty

the same processor can be used again and again as a transmitter of independent paths between pairs of processors, even if it were a relay. Moreover, even being a faulty processor does not matter for the simple reason that the total number of independent paths that would be affected by faulty processors will never exceed t.

Before proving the validity of the algorithm, consider the example in Figure 3. The network is 3-connected; it contains 5 processors and z is a faulty transmitter. At step 2 of the algorithm, processors x and y obtain the value a while processors u and w obtain b. At step 3a every processor transfers its version to the rest of the receivers. At step 3b every processor evaluates MAJORITY$\{a, a, b, b\}$, which produces the same value.

Lemma 8. *The Byzantine Algorithm terminates.*

Proof: Assumption P8 implies that the broadcasting process, at step 1 of the algorithm, is finite. That is after a finite time every processor will actually receive the messages and reliable processors will stop initiating new messages for the current broadcasting.

Step 2 of the algorithm might cause trouble. To consider the set A, a processor has to characterize messages according to the index m, the set U, and the path π. The three parameters have a finite range of possible values. The receiver can dynamically determine those messages that arrived within time interval T. By Corollary 5, after finding a nonempty value it can eliminate any further message carrying the same π sent by the same transmitter. Moreover, the sampling interval T advances in time and messages outside the

interval can be ignored. It will eventually get the correct value, since it samples consecutively all intervals of size T, and by Theorem 1 one of them will produce the value. Thus, the total number of possible sets A is finite. Moreover, with respect to every set the depth of the algorithm is also finite. This completes the proof of termination. ∎

The rest of the proof that the algorithm implies the Byzantine Agreement is similar to those in [LSP80] and [D81].

Lemma 9. *Let G be an Unknown-Network of processors; U be a subset of processors from G with cardinality at least $2r + m$, and let v be a processor not in U. If the set of processors U contains at most r faulty processors, then Algorithm $BG(G, v, U, \pi, t, m)$ satisfies condition Byz2 with respect to U.*

Proof: The condition Byz2 determines the behavior of the reliable processors in the case where the transmitter is reliable. A reliable processor faithfully follows the algorithm and therefore uses an appropriate π. The proof of the lemma is by induction on m. At the same time we will prove that if v is reliable and π is appropriate, then the receiver is not in π. This implies that π remains a simple path.

Byz2 holds for $m = 0$, because the transmitter is reliable; it sends its value with $\pi = \emptyset$ for the first time, and by Theorem 4 and Corollary 5 every reliable receiver receives the same value.

Assume that the lemma holds for $m - 1 > 0$. The reliable transmitter v broadcasts its unique value a to every receiver in U by the Send algorithm, using π for the first time. According to Theorem 4 and Corollary 5, every receiver w in U has not yet obtained a nonempty value carrying π, and therefore obtains in step 2 the value $a_w = a$. In step 3a, each processor u applies the algorithm $BG(G, u, U - u, \pi \cdot v, t, m - 1)$ to send the value it obtained to all the other processors in $U - u$. Since U contains at least $2r + m$ processors, $U - u$ contains at least $2r + m - 1$ processors. The set U does not contain v or the processors along the path π. The transmitter v considers π to be appropriate, and therefore u is not in π. Moreover, since v has not sent π before, u does not suppose to send $\pi \cdot v$ before. Thus, if u is reliable, then $\pi \cdot v$ is appropriate. This implies that the induction hypothesis can be applied to reach the conclusion that every other reliable processor in U receives from every other reliable processor the same value a. The set U contains at least $2r + m$ processors. Since $m > 0$ and since there are at most r faulty processors in U the majority in step 3b, done by every individual receiver w, produces the same value a. This completes the proof of the lemma. ∎

The following theorem uses Lemma 9 to prove that algorithm $BG(G, v, U, \pi, t, m)$ induces the Byzantine Agreement among the processors in U.

Theorem 10. *Let G be an Unknown-Network of processors; U be a subset of processors from G with cardinality at least $3m$, and let v be a processor not in U. If the set of processors $U \bigcup \{v\}$ contains at most m faulty processors, then Algorithm $BG(G, v, U, \pi, t, m)$ satisfies conditions Byz1 and Byz2 with respect to U.*

Proof: The proof is by induction on m, the bound on the number of faulty processors in the set $U \bigcup \{v\}$. The case $m = 0$ is the case where there are no faulty processors in U and v is a reliable transmitter. In this case Theorem 4 and Corollary 5 imply that the algorithm trivially satisfies Byz1 and Byz2.

Assume that the theorem holds for $m-1 > 0$. Consider first the case in which the transmitter v is reliable. Lemma 9 holds for every distribution of faulty processors, especially for the case where $r = m$ faulty processors in $U \bigcup \{v\}$. Therefore, algorithm BG satisfies Byz2 which implies Byz1 for this case.

The only remaining case is the one where the transmitter is a faulty processor. There are at most m faulty processors in $U \bigcup \{v\}$ and the transmitter is one of them, therefore U itself contains at most $m - 1$ faulty processors. Since U contains at least $3m$ processors then $U - u$ contains at least $3m - 1 > 3(m - 1)$ processors. This implies that the inductive assumption can be applied to step 3a. The validity of Byz2 and the inductive assumption on Byz1 imply that in step 3b every two reliable processors get the same value for every processor in U. This leads to the fact that $\text{MAJORITY}\{a_w(x) \mid x \subseteq U\}$ produces the same value for every processor w in U, which proves Byz1. ∎

Theorem 10 implies that the Byzantine Agreement, for the whole graph G, can be obtained by taking $V - \{v\}$ to be U, choosing $t = m$, and $\pi = \emptyset$.

Corollary 11. *The algorithm $BG(G, v, V - v, \emptyset, t, t)$ achieves the Byzantine Agreement for every network of more than $3t$ processors with connectivity greater than $2t$.* ∎

Refrences

[D81] D. Dolev, "The Byzantine Generals Strike Again", Technical Report STAN-CS-81-846, Computer Science Department, Stanford University, Feb. 1981.

[H72] F. Harary, *Graph Theory,* Addison-Wesley, 1972.

[L80] L. Lamport, "The Weak Byzantine Generals Problem", Technical Report 58, Computer Science Laboratory, SRI International, November 1980.

[LSP80] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", Technical Report 54, Computer Science Laboratory, SRI International, March 1980.

[PSL80] M. Pease, R. Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults", Journal of the ACM 27 (1980), 228-234.