Degradable Byzantine Agreement*

Nitin H. Vaidya **Dhiraj K. Pradhan**
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112

March 1993

Technical Report[1] 93-015

*A preliminary version appears in the 13th Intl. Conf. on Distr. Comp. Systems, May 1993, Pittsburgh.
[1]This report supercedes report #92-020.

Abstract

Consider a system consisting of a sender that wants to send a value to certain receivers. Byzantine agreement protocols [2, 7, 8] have been proposed to achieve this in the presence of arbitrary failures. The imposed requirement typically is that the fault-free receivers must all agree on the same value [7, 8]. (Dolev [2] analyzes a somewhat weaker form of agreement). It has been shown that such an agreement is impossible if a third or more of the nodes are faulty [2, 7, 8].

We propose an agreement protocol that achieves Lamport's Byzantine agreement [8] up to a certain number of faults and a degraded form of agreement with a higher number of faults. Essentially, the degraded form of agreement allows the fault-free receivers to agree on at most two different values one of which is necessarily the default value. The default value is distinguishable from all other values. The proposed approach is named "degradable Byzantine agreement" or simply "degradable agreement". Specifically, m/u-degradable agreement is defined using two parameters, m and u, and the following four conditions. (The term *node* refers to the sender and the receivers).

(1) If the sender is fault-free and at most m nodes are faulty, then all the fault-free nodes must agree on the sender's value.

(2) If the sender is faulty, and the number of faulty nodes is at most m, then all the fault-free nodes must agree on an identical value.

(3) If the sender is fault-free, and the number of faulty nodes is more than m but at most u, then the fault-free nodes may be partitioned into at most two classes. The fault-free nodes in one of the classes must agree on the sender's value, and the fault-free nodes in the other class must all agree on the default value.

(4) If the sender is faulty, and the number of faulty nodes is more than m but at most u, then the fault-free nodes may be partitioned into at most two classes. The fault-free nodes in one of the classes must agree on the default value, and the fault-free nodes in the other class must all agree on an identical value.

It is shown that at least $2m + u + 1$ nodes are necessary to achieve m/u-degradable agreement. An m/u-degradable agreement algorithm is presented for more than $2m+u$ nodes. Also, network connectivity of $m + u + 1$ is shown to be necessary and sufficient to achieve m/u-degradable agreement. For a system containing more than $2m + u$ node, conditions (3) and (4) imply that, up to u faults, at least $m + 1$ fault-free nodes are guaranteed to agree on the same value.

1 Introduction

Consider a system consisting of a sender that wants to send a value to certain receivers. Byzantine agreement (weak [7] or otherwise [8]) and Crusader agreement [2] protocols have been proposed to achieve this in the presence of arbitrary (possibly malicious) failures. The requirement is typically that the fault-free receivers must all agree on the same value [8, 7]. (Dolev [2] analyzes a somewhat weaker form of agreement.) Prior work has shown that such agreements are impossible if a third of the nodes (or more) are faulty. This paper also assumes the arbitrary failure model which is also known as the Byzantine failure model.

We propose an agreement protocol that achieves Lamport's Byzantine agreement[2] [8] up to a certain number of failures and a degraded form of agreement with a higher number of faults. Essentially, the degraded form of agreement allows the fault-free receivers to agree on at most two different values one of which is necessarily the default value.[3] This is a degraded form as compared to Byzantine agreement [8] which requires all the fault-free receivers to agree on a single value. The proposed approach is called "degradable Byzantine agreement" or simply "degradable agreement" for brevity. The next section presents a definition of degradable agreement.

This paper shows that degradable agreement is of interest in practice. It is shown that degradable agreement provides an ability to achieve forward recovery as well as backward recovery when the number of failures is large (more than a third of the nodes may be faulty).

For the sake of simplicity, this paper draws on the concepts presented in two well-known papers by Lamport et al. [8] and Dolev [2]. Section 2 defines the proposed degradable agreement approach. Section 3 motivates the proposed approach and discusses an application. An algorithm for achieving degradable agreement is presented in Section 4. Bounds on the number of nodes and connectivity for the proposed form of agreement are presented in Section 5. The problem addressed in this paper suggests a degradable clock synchronization approach. Section 6 discusses the issue of clock synchronization. Section 7 summarizes the

[2]Byzantine agreement was presented by Lamport, Shostak and Pease. However, for brevity we refer to it as Lamport's Byzantine agreement.

[3]Default value, denoted V_d, is distinguishable from all other values.

results.

2 Degradable Agreement Protocol

The system model can be described as follows. The system consists of a sender and some receivers. The sender wants to send its value to the receivers. In the following, the term *node* may refer to the sender or a receiver. A faulty node (sender or receiver) may demonstrate arbitrary behavior. V_d denotes the default value. The default value V_d is assumed to be distinguishable from all other relevant values.

Degradable agreement is defined using two parameters, m and u, where $u \geq m$. Degradable agreement defined by parameters m and u is hereafter called m/u-degradable agreement. An m/u-degradable agreement protocol satisfies the following conditions, where f is the number of faulty nodes.

m/u-**Degradable Agreement:**

- if $f \leq m$, then conditions D.1 and D.2 below must be satisfied.
- if $m < f \leq u$, then conditions D.3 and D.4 below must be satisfied.

(D.1) If the sender is fault-free, then all the fault-free receivers must agree on the sender's value.

(D.2) If the sender is faulty, then the fault-free receivers must agree on an identical value.

(D.3) If the sender is fault-free, then the fault-free receivers may be partitioned into at most two classes. The fault-free receivers in one class must agree on the sender's value, and the fault-free receivers in the other class must all agree on the default value.

(D.4) If the sender is faulty, then the fault-free receivers may be partitioned into at most two classes. The fault-free receivers in one class must agree on the default value, and the fault-free receivers in the other class must all agree on an identical value.

Conditions D.1 and D.2 are identical to those satisfied by Lamport's Byzantine agreement [8]. Conditions D.3 and D.4 define degraded agreement and are applied in fault situations with more than m but at most u faults. Thus, when $m = u$, degradable agreement is equivalent to Lamport's Byzantine agreement.

Let N be the number of nodes in the system. Observe that, if $N > 2m + u$ then m/u-degradable agreement ensures (by conditions D.3 and D.4) that at least $m + 1$ fault-free nodes (including the sender) agree on an identical value, even when the number of faults is more than m (but at most u). Thus, graceful degradation can be achieved. Note that graceful degradation is possible up to u faults even when $u \geq N/3$ only if we insist on achieving Byzantine agreement only up to m faults for some $m < \left\lfloor \frac{N-1}{3} \right\rfloor$. In other words, the capability to achieve Byzantine agreement can be traded with the capability to achieve degraded agreement up to a larger number of faults. (Note that the above does not contradict the impossibility result in [4].)

It is later proved that to achieve m/u-degradable agreement the system must consist of at least $2m + u + 1$ nodes (including the sender), and also that $2m + u + 1$ nodes are sufficient. Therefore, given a system consisting of 7 nodes, one may achieve any one of the following:

- 2/2-degradable agreement, or
- 1/4-degradable agreement, or
- 0/6-degradable agreement.

This illustrates the trade-off between Byzantine agreement and degraded agreement. The following table lists the minimum number of nodes necessary for different values of m and u.

u / m	1	2	3	4	5
1	4	5	6	7	8
2	–	7	8	9	10
3	–	–	10	11	12

It is known that if a third (or more) of the clocks are faulty, it is not possible to achieve clock synchronization [3, 6]. Clock synchronization is necessary to be able to correctly detect the presence or absence of messages. Section 6 discusses this issue.

3 Motivation

Consider a fault tolerant system consisting of multiple computation channels. Figure 1(a) illustrates a system with three channels. Byzantine agreement is useful in such systems to distribute information from a single sender (for example, a sensor) to all the channels [11]. The three channels in Figure 1(a) obtain their input from the sensor and then perform

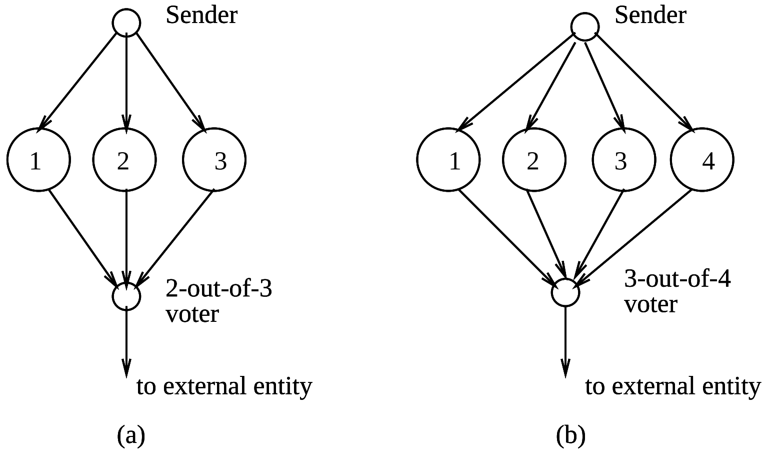

Figure 1: Multiple channel systems

computations on that input. Eventually, the output of the three channels must be sent to an external entity (for example, to a controller). The external entity takes a majority vote on the output of the three channels and determines the correct value. It is clear that if the sender is itself faulty, the external entity may not be able to obtain the correct value. Thus, in such a system, Lamport's Byzantine agreement [8] ensures the following conditions:

(B.1) Given is a system with $3m$ channels and 1 sender. If the sender is fault-free and at most m channels are faulty, then the external entity obtains the correct value using majority vote.

(B.2) All the fault-free channels are in an identical state, up to m faults.

Although the proposed approach is useful when multiple senders measure the same quantity and send its value to the channels, the discussion in this paper is limited to a single sender. The three-channel system in the above example may fail, if two of the channels obtained the same incorrect value from the sender. This could happen if two nodes out of four (three channels and one sender) are faulty, as Byzantine agreement with four nodes only tolerates one fault. In general, if more than m faults occur, Byzantine agreement may result in the external entity using an incorrect output, even if the sender is fault-free. However, Byzantine agreement tolerates up to m faults, meaning that forward recovery [10] can be performed in the presence of up to m faults.

The concept of a default value is pertinent to the discussion below. If the external entity obtains a default value from the multiple channel system, it can take a "default" action which usually results in a safe operation. Thus, degradable agreement improves the safety of the system. Another possibility is to re-do the computation (i.e. perform backward recovery [10]) whenever the external entity receives a default value. Backward recovery is effective against transient failures.

Degradable agreement, thus, improves the ability to survive more than m faults. Obviously, achieving this requires more resources, but we show that the increase in resource requirements is minimal. Consider a four channel system shown in Figure 1(b). For this system,[4] $m = 1$ and $u = 2$. Using the proposed degradable agreement approach the following conditions can be ensured as compared to those listed above for Byzantine agreement.

(C.1) Given is a system with $(2m+u)$ channels and 1 sender $(m \leq u)$. If the sender is fault-free and at most m channels are faulty, then the external entity obtains the correct value using $(m+u)$-out-of-$(2m+u)$ vote[5] on the outputs of the $2m+u$ channels.

(C.2) If the sender is fault-free and more than m but at most u channels are faulty, then the external entity obtains either the correct value or the default value.

[4]This system contains five nodes: four channels and one sender.

[5]$(m+u)$-out-of-$(2m+u)$ vote of $2m+u$ values is α if $\geq (m+u)$ values are α, default value otherwise.

(C.3) The fault-free channels are all in an identical state if number of faults is at most m. Also, up to u faults, the fault-free channels are divided into at most two classes; the channels in one class are in a "default" state (i.e. a safe state).

It is clear that in many situations, it is safer to use the default value as compared to an incorrect value. For instance, if a controller in a fly-by-wire system receives a default value from the computer, as a safety precaution it can inform the pilot about the problem.

Condition C.2 results in a correct or default output even when more than a third of the channels may be faulty. Condition C.1 is essentially the same as B.1. Thus, by condition C.2, the degradable agreement approach improves the ability to survive a larger number of faults. Also, conditions C.2 and C.3 ensure that the state of the fault-free channels diverges "gracefully". (Two fault-free channels that are not in the default state are always in an identical state.)

The above discussion motivates the proposed degradable agreement approach. The proposed approach, in general, improves the safety of the system and also improves the ability of the system to perform backward recovery in the presence of more than m faults. Similar to Byzantine agreement, degradable agreement can perform forward recovery up to m faults.

4 An Algorithm for m/u-Degradable Agreement

This section presents an algorithm to prove that m/u-degradable agreement can be achieved with more than $(2m + u)$ nodes. No attempt is made here to present an efficient algorithm.

It is assumed that V_d is a default value that is distinguishable from other values. Define VOTE(μ, ν) of ν values $w_1, w_2, \cdots w_\nu$ as α if at least μ of the ν values are equal to α, else VOTE(μ, ν) is defined to be the default value V_d. Also, in case of a tie, define VOTE(μ, ν) $= V_d$. For example, VOTE(2,4) of values 1, 2, 2, 3 is 2 and VOTE(2,4) of values 1, 2, 0, 3 is V_d. VOTE(2,4) of values 1, 2, 2, 1 is V_d because of the tie.

Algorithm BYZ presented below may be viewed as an extension of an algorithm in

[8]. BYZ assumes that the nodes are fully connected. Following assumptions are made regarding messages when proving correctness of algorithm BYZ:

(a) all messages are delivered correctly within a bounded delay,

(b) source of a received message can be identified,

(c) presence or absence of a message can be correctly detected when the number of faulty nodes is at most m, and

(d) when more than m faults exist, a fault-free node may incorrectly assume a message to be absent.

Whenever a node detects a message to be absent, it assumes that the message contains value V_d. Assumptions (a), (b) and (c) above are similar to those made by others [8]. The motivation behind assumption (d) will be clear in Section 6.

Algorithm BYZ is recursive. The algorithm for $m = 0$ is omitted here. Algorithm BYZ(m, m) achieves m/u-degradable agreement given at least $2m + u + 1$ nodes. Now we present BYZ$(1, m)$ and BYZ(t, m). In these algorithms, the following notation is used. In BYZ$(1, m)$, n_1 is the number of nodes to which algorithm BYZ$(1, m)$ is being applied. Similarly, in BYZ(t, m), n_t is the number of nodes to which algorithm BYZ(t, m) is being applied. N is the total number of nodes in the system. m and u are the two parameters that define the m/u-degradable agreement that we want to achieve in this system of N nodes. For BYZ(m, m), $n_m = N$. It is assumed that $N > 2m + u$ and $u \geq m > 0$. It can be seen that for BYZ(t, m), $n_t = N - m + t$.

Algorithm BYZ$(1, m)$

1. The sender sends its value to all the $(n_1 - 1)$ receivers.

2. Each receiver broadcasts the value it received from the sender to $(n_1 - 2)$ other receivers. As there are $(n_1 - 1)$ receivers, each receiver now has $(n_1 - 1)$ values.

3. Each receiver uses VOTE$(n_1 - 1 - m, n_1 - 1)$ of these $(n_1 - 1)$ values.

BYZ$(1, m)$ is *not* recursive. Lemma 2 in Section 4.1 proves some properties of algorithm BYZ$(1, m)$.

Algorithm BYZ(t, m), $1 < t \leq m$

1. The sender sends its value to all the $(n_t - 1)$ receivers.
2. For each i, let v_i be the value receiver i received from the sender in step 1. Receiver i acts as the sender in algorithm BYZ$(t-1, m)$ to send the value v_i to each of the $(n_t - 2)$ other receivers.
3. For receiver i, let $w_i = v_i$ and for each $j \neq i$, let w_j be the value receiver i received from receiver j in step 2 (using algorithm BYZ$(t-1, m)$). Thus, receiver i now has $n_t - 1$ values $w_1, w_2, \cdots, w_{n_t - 1}$. Receiver i uses VOTE$(n_t - 1 - m, n_t - 1)$ of these $(n_t - 1)$ values.

Algorithm BYZ(m, m) achieves m/u-degradable agreement if $N > 2m + u$, as proved below. Note that as the recursion unfolds in BYZ(m, m), the values of n_t and t change at each level of the recursion, however, the value of m remains fixed.

4.1 Proof of Correctness: Algorithm BYZ

The correctness of algorithm BYZ is being proved under assumptions (a) through (d) listed earlier in this section. For future reference note that assumption (d) is applicable only in fault situations where more than m nodes are faulty.

When a message is detected to be absent by a node, that node considers the absent message to contain default value V_d. Therefore, the following assumes that each node always sends a message when it is supposed to; however, a faulty node may send an incorrect message (possibly with value V_d). (Also see Section 6 for a related discussion). Assume that $N > 2m + u$ and $u \geq m > 0$.

Lemma 1 *When BYZ(t, m) is called with $t \geq 1, m \geq 1$, the following conditions hold:* (i) $n_t > t + u + m$ *and* (ii) $u < n_t - 1 - m$.

Proof: The proof is by induction on t. Initially when BYZ(m, m) is executed, $t = m$, $n_t = n_m = N$, and $t + u + m = 2m + u$. As $N > 2m + u$, we have $n_t > t + u + m$. Therefore, condition (i) holds for $t = m$.

Now we assume that (i) holds for some $t \leq m$ and show that it holds for $t - 1$. As (i) holds for t, we have $n_t > t + u + m$. BYZ($t - 1, m$) is called in step 2 of BYZ(t, m) with $n_t - 1$ nodes. As $n_t > t + u + m$, we have $(n_t - 1) > (t - 1) + u + m$. As $n_{t-1} = n_t - 1$, condition (i) holds for $t - 1$.

Thus, condition (i) of the lemma is proved. Condition (i) implies that $u < n_t - t - m$. As $t \geq 1$, this implies that $u < n_t - 1 - m$. Thus, condition (ii) is also proved. $\square$

Lemma 2 *Let f be the number of faulty nodes in the system. If $n_1 > 1 + u + m$, then*

1. *BYZ($1, m$) satisfies condition D.1 if $f \leq m$.*
2. *BYZ($1, m$) satisfies condition D.2 if $f = 1$.*
3. *BYZ($1, m$) satisfies condition D.3 if $m < f \leq u$.*
4. *BYZ($1, m$) satisfies condition D.4 if $m = 1$ and $1 < f \leq u$.*

Proof: Let $n_1 > 1 + u + m$. f is the number of faulty nodes in the system.

Case 1: $f \leq m$ and the sender is fault-free.

Assume that the fault-free sender sends value α to the receivers in step 1 of BYZ($1, m$). In step 2, each fault-free receiver broadcasts α (the value received from the sender) to the other $(n_1 - 2)$ receivers. When the broadcasts in step 2 of BYZ($1, m$) are complete, each receiver will have $(n_1 - 1)$ values, of which at least $(n_1 - 1 - m)$ must be α (because at least

$n_1 - 1 - m$ of the receivers are fault-free). Also, as $n_1 > 1 + u + m$ and $u \geq m$, we have $n_1 - 1 - m > m$. Therefore, no value other than α is received from any $n_1 - 1 - m$ nodes. Therefore, each fault-free node obtains α in step 3 of BYZ$(1, m)$. Thus, item 1 in the lemma is proved.

Case 2: $f = 1$ and the sender is faulty.

In this case all the receivers are fault-free. Therefore, in step 2 of BYZ$(1, m)$, each receiver must obtain the same set of $n_1 - 1$ values. This implies that in step 3, each receiver will obtain the same value using VOTE$(n_1 - 1 - m, n_1 - 1)$. Thus, item 2 in the lemma is proved.

Case 3: $m < f \leq u$ and the sender is fault-free.

As more than m nodes are faulty, by assumption (d), a fault-free node may incorrectly declare a message from another fault-free node as absent. Recollect that absent messages are assumed to contain default value V_d.

Assume that the sender sends value α to the receivers in step 1 of BYZ$(1, m)$. Each fault-free receiver broadcasts the value received from the sender to the other $n_1 - 2$ receivers. When the broadcasts in step 2 of BYZ$(1, m)$ are complete, each fault-free receiver will have $n_1 - 1$ values of which at most u values may be different from α and V_d (because at most u receivers are faulty). By Lemma 1, $n_1 - 1 - m > u$. Therefore, in step 3, a fault-free receiver cannot obtain VOTE$(n_1 - 1 - m, n_1 - 1)$ equal to any value other than α and V_d. In other words, each fault-free receiver must obtain VOTE$(n_1 - 1 - m, n_1 - 1)$ equal to either α or V_d. Thus, item 3 in the lemma is proved.

Case 4: $m = 1,\ 1 < f \leq u$ and the sender is faulty.

Assumption (d) is applicable in this case.

As $m = 1$, we have $n_1 > 2 + u$ and $n_1 - 1 - m = n_1 - 2$. As the sender is faulty, at least $n_1 - u$ receivers are fault-free.

At the end of step 2 of BYZ$(1, m)$, each fault-free receiver obtains a set of $n_1 - 1$ values. Assume that, in step 3, a receiver, say A, obtains VOTE$(n_1 - 2, n_1 - 1) = \beta$, $\beta \neq V_d$. This means that at least $n_1 - 2$ of the $n_1 - 1$ values obtained by A in step 2 are β. This in turn implies that at least $(n_1 - 2) - (u - 1) = n_1 - 1 - u$ fault-free receivers received β from the sender in step 1 of BYZ$(1, m)$. Then, any fault-free receiver other than A, say receiver B, can have (at the end of step 2) at most u values that are different from β and V_d. As $u < n_1 - 2$, receiver B cannot obtain VOTE$(n_1 - 2, n_1 - 1)$ equal to any value other than β and V_d. In other words, if receiver A obtains value $\beta \neq V_d$ in step 3, then every other receiver must obtain either β or V_d in step 3. Thus, item 4 in the lemma is proved. □

Lemma 1 presented earlier implies that the condition $n_1 > 1 + u + m$ required for Lemma 2 to be true is satisfied by BYZ$(1, m)$.

Lemmas 3 through 6 below prove that BYZ(m, m) achieves m/u-degradable agreement. The proofs of these lemmas assume that: (i) $m > 1$, (ii) $N > 2m + u$ and (iii) $u \geq m$.

Lemma 3 *For $1 \leq t \leq m$, algorithm BYZ(t, m) satisfies condition D.1 if $n_t > t + u + m$ and at most m nodes are faulty.*

Proof: By Lemma 1, $n_t > t + u + m$. Also, it is given that $u \geq m$.

The proof is by induction on t. The number of faulty nodes is at most m. Condition D.1 assumes that the sender is fault-free. Therefore, the lemma is true for $t = 1$ by Lemma 2.

We now assume that the lemma is true for BYZ$(t-1, m)$ where $2 \leq t \leq m$, and prove it for BYZ(t, m). In step 1 of BYZ(t, m), the fault-free sender sends a value, say α, to all the $(n_t - 1)$ receivers. In step 2, each fault-free receiver acts as a sender in BYZ$(t-1, m)$ to send value α (which it received from the sender) to the other $(n_t - 2)$ receivers. As $n_t > t + u + m$, $(n_t - 1) > (t - 1) + u + m$. Also, $n_{t-1} = n_t - 1$. Therefore, the induction hypothesis holds for BYZ$(t - 1, m)$. Thus, at the end of step 2, every fault-free receiver gets $w_j = \alpha$ for each fault-free receiver j. Since there are at most m faulty receivers, at least $n_t - 1 - m$ are fault-free. Therefore, each fault-free receiver i has $w_j = \alpha$ for at least $(n_t - 1 - m)$ of the

$n_t - 1$ receivers. As $m \leq u < n_t - 1 - m$ (by Lemma 1), this implies that each fault-free receiver obtains VOTE$(n_t - 1 - m, n_t - 1) = \alpha$. Thus, the lemma is proved for BYZ(t, m). □

Lemma 4 *For $1 \leq t \leq m$, algorithm BYZ(t, m) satisfies condition D.2 if $n_t > t + u + m$ and at most t nodes are faulty.*

Proof: The proof is by induction on t. By Lemma 2, BYZ$(1, m)$ satisfies condition D.2 if at most 1 node is faulty. We therefore assume that the lemma is true for BYZ$(t - 1, m)$ where $2 \leq t \leq m$, and prove it for BYZ(t, m).

The number of faulty nodes is at most t. Condition D.2 assumes that the sender is faulty. Therefore, at most $(t - 1)$ of the receivers are faulty. As $n_t > t + u + m$, $(n_t - 1) > (t - 1) + u + m$. In step 2, a receiver uses BYZ$(t - 1, m)$ to send the value it received from the sender to the other $n_t - 2$ receivers. As at most $t - 1$ of the $n_t - 1$ receivers are faulty, we can apply the induction hypothesis to conclude that BYZ$(t - 1, m)$ satisfies condition D.2 (note that $n_{t-1} = n_t - 1$). As $t \leq m$, by Lemma 3, it follows that BYZ$(t - 1, m)$ satisfies condition D.1 as well. Hence, at the end of step 2 of BYZ(t, m), any two fault-free receivers must obtain the same vector $w_1, w_2, \cdots, w_{n_t - 1}$. Therefore, all fault-free receivers must obtain the same value VOTE$(n_t - 1 - m, n_t - 1)$ in step 3 of BYZ(t, m). Thus, the lemma is proved for BYZ(t, m). □

Lemma 5 *For $1 \leq t \leq m$, algorithm BYZ(t, m) satisfies condition D.3 if $n_t > t + u + m$ and more than m but at most u nodes are faulty.*

Proof: The proof is by induction on t. Condition D.3 assumes that the sender is fault-free. By Lemma 2, BYZ$(1, m)$ satisfies condition D.3 if at most u nodes are faulty. We therefore assume that the lemma is true for BYZ$(t - 1, m)$ where $2 \leq t \leq m$, and prove it for BYZ(t, m).

In step 1 of BYZ(t, m), the fault-free sender sends a value, say α, to all the $(n_t - 1)$ receivers. Receiver j receives value v_j from the sender in step 1. As more than m nodes are faulty, value v_j received by a fault-free receiver j may be α or V_d (by assumption (d)).

In step 2, each fault-free receiver applies BYZ$(t-1, m)$ with $(n_t - 1)$ nodes. As $n_t > t+u+m$, $(n_t - 1) > (t-1)+u+m$. Also, $n_{t-1} = n_t - 1$. Therefore, we can apply the induction hypothesis to conclude that every fault-free receiver gets $w_j = v_j$ or V_d from each fault-free receiver j (using BYZ$(t-1, m)$). As v_j itself may be α or V_d, this implies that every fault-free receiver gets $w_j = \alpha$ or V_d from each fault-free receiver j. Since at most u receivers are faulty, at least $n_t - 1 - u$ of the $n_t - 1$ values received (in step 2 of BYZ(t, m)) by any fault-free receiver must be α or V_d. Now, $u < n_t - 1 - m$ by Lemma 1. Therefore, each fault-free receiver must obtain VOTE$(n_t - 1 - m, n_t - 1)$ equal to either α or V_d. Thus, the lemma is proved for BYZ(t, m). □

Lemma 6 *Algorithm BYZ(m, m) satisfies condition D.4 if more than m but at most u nodes are faulty and $n_m > 2m + u$.*

Proof: Condition D.4 assumes that the number of faulty nodes is at most u and the sender is faulty. Therefore, at most $(u-1)$ of the receivers are faulty. For BYZ(m, m), $n = N > 2m + u$.

In step 2 of BYZ(m, m), each fault-free receiver sends the value it received (from the sender) to the other $n_m - 2$ receivers using BYZ$(m-1, m)$. As $n_m > 2m + u$, $(n_m - 1) > (m-1)+u+m$. Also, $n_{m-1} = n_m - 1$. Therefore, by Lemma 5, we know that BYZ$(m-1, m)$ satisfies condition D.3. As at most $u-1$ receivers are faulty, at least $n_m - u$ are fault-free. Without loss of generality, assume that receivers 1 through $n_m - u$ are fault-free. v_j is the value fault-free receiver j received from the sender in step 1 of BYZ(m, m). Therefore, at the end of step 2, each fault-free receiver must obtain a vector of $n_m - 1$ values of the form $(v_1/V_d, v_2/V_d, \cdots, v_{n_m - u}/V_d, X, \cdots, X)$, where v_j/V_d indicates that the corresponding value is either v_j or V_d, and X denotes a value that is not relevant in our discussion here.

Now, suppose that in step 3 of BYZ(m, m), a fault-free receiver i obtains VOTE$(n_m - 1 - m, n_m - 1) = \beta$ where $\beta \neq V_d$. This implies that, for fault-free receiver i, at least $(n_m - 1 - m) - (u-1) = n_m - m - u$ of the first $n_m - u$ values in the vector of $n_m - 1$

values must be β. As $n_m = N$, $n_m - m - u \geq m + 1$. This implies that at least $m + 1$ fault-free receivers received β from the sender in step 1. Therefore, each fault-free receiver must have (at the end of step 2 of BYZ(m, m)) at least $m + 1$ values that are β or V_d. Thus, VOTE$(n_m - 1 - m, n_m - 1)$ cannot be any value other than β and V_d. Thus, the lemma is proved. □

Theorem 1 *BYZ(m,m) achieves m/u-degradable agreement if $N > 2m + u$.*

Proof: For $m = 1$, the proof follows from Lemma 2. For $m > 1$, the proof follows from Lemma 1 and from Lemmas 3 through 6 by choosing $t = m$. □

5 Bounds for Degradable Agreement

This section presents the lower bounds on the number of nodes and the network connectivity necessary to achieve m/u-degradable agreement. For future reference note that, by definition, a system that achieves m/u-degradable agreement also achieves Byzantine agreement [8] up to m faults.

Theorem 2 *Given N nodes, m/u-degradable agreement can be achieved only if $N > 2m+u$.*

Proof: The proof is in two parts. Part I proves that 1/2-degradable agreement is impossible with less than 5 nodes. Part II extends this result to prove the theorem in general.

Part I: It is clear that the number of nodes must be at least 4, else Byzantine agreement with 1 fault cannot be achieved. Therefore, assume that 1/2-degradable agreement can be achieved in a system of 4 nodes. Let the nodes be named S, A, B and C, where S is the sender node. Figures 2(a) through (c) illustrate three fault scenarios; the shaded nodes are faulty. In the following, let α and β be two different values distinct from V_d, i.e., $V_d \neq \alpha \neq \beta \neq V_d$.

In scenario (a), only node A is faulty. Also, in scenario (a), let the sender's value be β. Therefore, by condition D.1, fault-free nodes B and C must agree on β. The arcs in

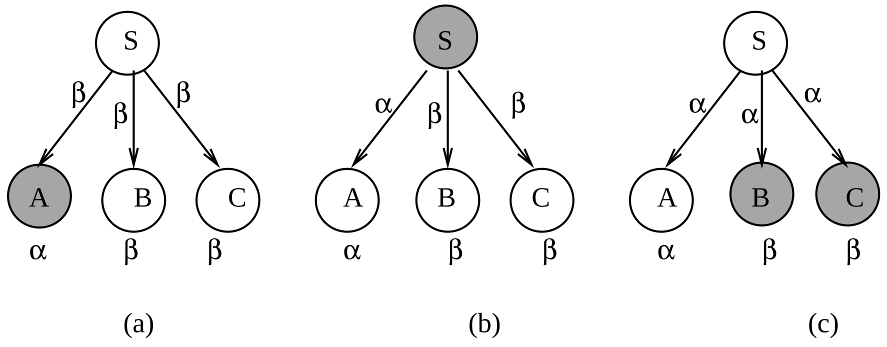

Figure 2: Proving lower bound on the number of nodes

Figure 2(a) indicate the values sent by S to the other nodes. Value β written below node B (C) in the figure indicates that node B (C) may tell the other receivers that the sender sent him β. Value α written below node A indicates that node A pretends to have received α from sender S. As node A is faulty, such a behavior is possible.

In scenario (b), only sender node S is faulty. Also, the faulty sender S sends α to node A and β to nodes B and C, as indicated in Figure 2(b). As node S is faulty, such a behavior is possible. Now, fault-free node B cannot distinguish between scenarios (a) and (b). As seen above, in scenario (a), node B agrees on β. Hence, in scenario (b) also node B must agree on β. Therefore, by condition D.2, nodes A and C too must agree on β.

In scenario (c), nodes B and C are faulty. Also, let the sender's value be α. Additionally, in scenario (c) assume that faulty nodes B and C pretend that they received value β from the sender, as illustrated in Figure 2(c). Now, fault-free node A cannot distinguish between scenarios (b) and (c). We have already concluded that node A agrees on β in scenario (b). Therefore, A must agree on β in scenario (c) as well. However, condition D.3 requires that fault-free node A should agree on either α or V_d in scenario (c). As $\alpha \neq \beta \neq V_d$, this is a contradiction.

Thus, 1/2-degradable agreement cannot be achieved with less than 5 nodes.

Part II: The proof in Part II is similar to a proof in [8]. It is clear that to achieve m/u-degradable agreement at least $3m + 1$ nodes are necessary (otherwise Byzantine agreement cannot be achieved for m faults). Therefore, consider a system consisting of N nodes such that $N = 3m + \mu$ where $1 \leq \mu \leq (u - m)$. Thus, $N \leq 2m + u$. Also, $m + \mu \leq u$. Assume that m/u-degradable agreement can be achieved in this system.

Divide the $3m + \mu$ nodes into four groups such that three groups contain m nodes each and the fourth group contains μ nodes. (The sender is included in one of the first three groups). Name the three groups containing m nodes each as S_m, A_m and B_m, where S_m is the group that contains the sender node. The group containing μ nodes is named C_μ. Now, assume that the four node system in Figure 2 simulates the $3m + \mu$ node system; nodes S, A, B and C in Figure 2 simulate the nodes in S_m, A_m, B_m and C_μ, respectively. Node A agrees on value α only if all the nodes in A_m agree on value α (similarly for S, B and C). Also, if node A is faulty, then all the nodes in A_m are faulty (similarly for S, B and C). Now, as the $3m + \mu$ node system can achieve m/u-agreement, it should be able to reach correct agreement in all the three fault scenarios corresponding to those illustrated in Figure 2. But that is an impossibility as shown in Part I. □

As proved by the correctness of algorithm BYZ, $2m + u + 1$ nodes are not only necessary but also sufficient to achieve m/u-degradable agreement. The following theorem states the bound on network connectivity.

Theorem 3 *Given a system containing at least $2m+u+1$ nodes, m/u-degradable agreement can be achieved if and only if network connectivity is at least $m + u + 1$.*

Proof: Appendix A presents the proof. □

6 Message Passing and Clock Synchronization

An important implementation issue is that of clock synchronization. In order to ensure that presence or absence of a message can be detected, it is necessary to synchronize the logical clocks of all the fault-free nodes. However, it has been shown that clock synchronization cannot be achieved if a third (or more) clocks are faulty [3, 6]. When using m/u-degradable agreement, u may be larger than a third of the number of nodes. Thus, clock synchronization cannot be guaranteed *if* a node being faulty necessarily implies that its clock is faulty as well. There are three approaches to deal with this problem, as discussed in the following three subsections. The first two approaches use some form of clock synchronization, while the third relaxes the requirement of clock synchronization.

6.1 Degradable Clock Synchronization

Algorithm BYZ was earlier proved correct under assumptions (a) through (d) listed in Section 4. Assumptions (c) and (d) are re-stated below.

(c) When at most m nodes are faulty, each fault-free node correctly detects absence *or* presence of messages from other nodes. (This requires clock synchronization.)

(d) When more than m nodes are faulty, a fault-free node may incorrectly declare a message from another node to be absent. (This may happen due to time-outs.) In other words, absence of a message is always correctly detected, but presence of a message may not be correctly detected when more than m faults exist.

Condition (c) can be satisfied, provided that the logical clocks of fault-free nodes are synchronized up to m faults. This is achievable, as $\frac{N}{3} > \frac{2m+u}{3} \geq m$.

Condition (d) can always be satisfied. When the clocks are not synchronized, a fault-free node may time-out too early, thereby incorrectly declaring a message to be absent. A message that is actually absent will not be "created" due to lack of synchronization. (We assume that messages are numbered to facilitate detection of missing and duplicate messages.)

The above discussion and the proof of BYZ in Section 4 imply correctness of algorithm BYZ in the sense that m/u-degradable agreement is achieved *whenever* the algorithm terminates. However, in the absence of clock synchronization with more than m faults, the algorithm is not yet guaranteed to complete within "bounded" time. The term *bounded time* implies that the maximum time required is upper bounded by a known finite constant.

When the number of faults is at most m, all the fault-free nodes will have synchronized logical clocks. Therefore, it is possible to determine an upper bound, say t_{upper}, such that agreement algorithm BYZ must terminate on each node by the time its physical clock reads t_{upper}. As shown in Appendix B, algorithm termination within bounded time can be guaranteed with more than m faults if one of the following is true:

1. at least $m+1$ fault free nodes have their clocks synchronized and "approximate" real time, or

2. at least $m+1$ fault-free nodes detect (within bounded time) the presence of more than m faulty nodes.

Keeping the above conditions in mind, we introduce the following problem.

m/u-Degradable Clock Synchronization:

1. if at most m clocks are faulty, all the fault-free clocks must be synchronized and should "approximate" the real time.[6]

2. if more than m but at most u clocks are faulty then, either

   - at least $m+1$ fault-free clocks must be synchronized and should approximate the real time, or

   - at least $m+1$ fault-free clocks should detect the existence of more than m faulty clocks within bounded time.

We *conjecture* that m/u-degradable clock synchronization can be achieved if and only if there are more than $2m+u$ clocks. Motivation for this conjecture is the observation in Section 2 that, given $2m+u+1$ nodes, if at most u nodes are faulty, at least $m+1$ fault-free nodes agree on the same value using m/u-degradable agreement.

[6] *Approximate* means that the logical clock is within a known linear envelope around the real time. Such a logical clock is also said to be *accurate* [12].

While a solution to the above m/u-Degradable Clock Synchronization problem is sufficient to guarantee termination of BYZ within bounded time, it is not a necessary condition. Section 6.3 presents another solution that terminates BYZ within a bounded amount of time, up to u faults, using the physical clocks.

6.2 Traditional Clock Synchronization

One solution, applicable to systems such as FTMP, NETS [11] and FTP [5], is the use of hardware clock synchronization (as opposed to software algorithms). In typical systems, the complexity and cost of clock hardware is orders of magnitude lower as compared to the processor (or node) complexity. Therefore, the failure rates for clock hardware are likely to be significantly lower. Hence, one may assume that at most a third of the clocks may fail (although more than a third of the processors may fail). In such a case the term "node failure" in our discussion in this paper refers to the failure of the processor. Essentially, a processor being faulty does not necessarily imply that the associated clock hardware is faulty as well. For example, in Figure 1(b), we may assume that at most one clock may be faulty, while using 1/2-degradable agreement for the processors. Such an assumption is not likely to affect the overall system reliability or safety adversely as the clock failure rates are much lower than processor failure rates. Also, as clock complexity is lower, it is possible to use conservative designs to further lower the failure rates.

Another approach is to use more clocks than the number of processors. For example, for the system in Figure 1(b), one may use two more clocks in addition to those associated with each of the nodes, making the system capable of tolerating two clock failures. Addition of $3\mu + 1 - N$ clocks can be used to make the system capable of tolerating up to μ clock failures, for some μ where $m < \mu \leq u$. (If $N > 3\mu$, no additional clocks are necessary.) Note that the additional clocks form $(3\mu + 1 - N)$ new nodes, however, the new nodes contain only clocks (see Figure 3). This idea is analogous to the concept of witnesses proposed for maintaining consistency in replicated file systems [9].

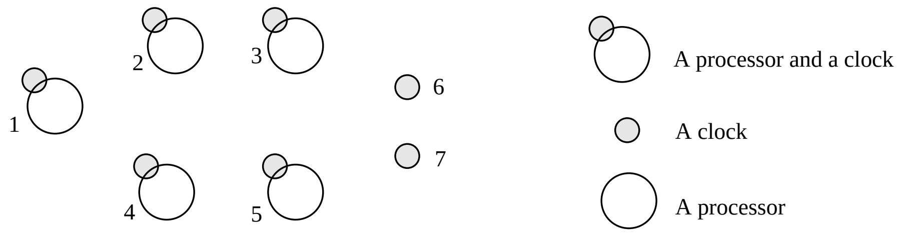

Figure 3: Example: A system with five original nodes (each containing a processor and a clock) and two new nodes (each containing only a clock)

6.3 Loosing Clock Synchronization With More Than m Faults

Each node has an associated physical clock and a logical clock. The logical clocks of the fault-free nodes are kept synchronized (whenever possible). As pointed out in Section 6.1, provided the logical clocks of fault-free nodes are synchronized, an upper bound t_{upper} can be determined such that each fault-free node completes executing algorithm BYZ by the time its physical clock reads t_{upper}.

Consider the following steps to be performed by each fault-free node. We assume that logical clocks are synchronized using any algorithm (e.g. [12]) that tolerates m faults in N nodes.

- Perform algorithm BYZ.
- If physical clock reads t_{upper}, but BYZ has not terminated, then (i) terminate BYZ, and (ii) accept value V_d

When at most m nodes are faulty, the logical clocks of all fault-free nodes will be synchronized and each fault-free node would have completed BYZ when its physical clock reads t_{upper}. When the number of faults is larger than m, the logical clock of a fault-free node may no longer be synchronized. However, it will terminate algorithm BYZ when its physical clock reaches t_{upper} (if it has not already completed). Therefore, all the fault-free nodes will terminate execution of BYZ within bounded time. The above steps force a fault-free node to agree on value V_d, when the physical clock exceeds t_{upper} (and BYZ has not completed). As this situation can occur only when more than m faults exists, agreeing on V_d does not violate the requirements of degradable agreement. Additionally, the physical clock of a fault-free node will reach t_{upper} within bounded time. Therefore, the above steps result in completion of BYZ within bounded time.

7 Summary

m/u-degradable agreement protocol that achieves Lamport's Byzantine agreement [8] up to m faults, and a degraded form of agreement with more than m but at most u faults is proposed. Up to m faults, all the fault-free nodes agree on an identical value. For more than m faults (up to u faults), the degraded form of agreement allows the fault-free nodes to agree on at most two different values one of which is necessarily the default value; the other value is the sender's value if the sender is fault-free. It is shown that $2m + u + 1$ nodes are necessary and sufficient to achieve m/u-degradable agreement. An m/u-degradable agreement algorithm is presented for more than $2m + u$ nodes. Also, network connectivity of $m + u + 1$ is shown to be necessary and sufficient to perform m/u-degradable agreement.

The results presented in this paper suggest that degradable agreement is a cost-effective approach for tolerating a small number of Byzantine failures using forward recovery and a large number of failures using backward recovery. Further research is required to explore the applications of degradable agreement.

The paper also formulates the problem of degradable clock synchronization. This problem is a subject of further research.

A Network Connectivity of $m+u+1$ is Necessary and Sufficient

Let $N \geq 2m + u + 1$ be the number of nodes in the system under consideration, and let G represent the network of nodes. Let the connectivity of G be $\kappa \leq m + u$.

Proof of Necessity: This proof is similar to a proof in [2].

It is clear that connectivity κ must be at least $2m+1$, otherwise Byzantine agreement is not possible with m faults [4]. Therefore, let $2m < \kappa \leq m+u$. Assume that m/u-degradable agreement can be achieved in this system.

Consider the following scenario. Let $F = \{a_1, a_2, \cdots, a_\kappa\}$ be a set of nodes which can disconnect the network into two non-empty parts G_1 and G_2. Let $F_1 = \{a_1, \cdots, a_m\}$ and $F_2 = \{a_{m+1}, \cdots, a_\kappa\}$. Thus, $F_1 \cup F_2 = F$ and $F_1 \cap F_2 = \emptyset$. Also $|F_1| = m$ and $m < |F_2| \leq u$.

Without loss of generality, assume that the sender node is either in G_1 or in F_2. (Note that if the sender is in F, one can always define F_2 such that the sender belongs to F_2). We consider each case separately.

Case I: The sender is in G_1. Consider the following two fault scenarios, where $V_d \neq \beta \neq \alpha \neq V_d$.

(1) The sender is fault-free and its value is α. The nodes in F_1 are all faulty and all other nodes are fault-free. In this scenario, by condition D.1, all nodes in G_2 must agree on α.

(2) The sender is fault-free and its value is β. The nodes in F_2 are all faulty and all other nodes are fault-free. In this scenario, by condition D.3, all nodes in G_2 must agree on either β or V_d.

Let the present fault scenario be (1). Thus, all nodes in F_1 are faulty. All the messages between G_1 and G_2 must be relayed through the nodes in $F = F_1 \cup F_2$. Now, the faulty nodes in F_1 can always corrupt the messages to G_2 in such a way that nodes in G_2 cannot distinguish between fault scenarios (1) and (2). As scenarios (1) and (2) require the nodes in G_2 to agree on different values (recollect that $\beta \neq \alpha \neq V_d$), nodes in G_2 cannot satisfy the requirements of degradable agreement.

Case II: The sender is in F_2. Consider three fault scenarios below, where $V_d \neq \beta \neq \alpha \neq V_d$, and all the nodes in G_1 and G_2 are fault-free.

(3) The sender is fault-free and its value is β. The nodes in F_1 are all faulty and all other nodes are fault-free. In this scenario, by condition D.1, all nodes in G_1 must agree on β.

(4) All the nodes in F_2 including the sender are faulty. The nodes in F_1 are all fault-free. In this scenario, the nodes in G_1 and G_2 must satisfy condition D.4.

(5) The sender is fault-free and its value is α. The nodes in F_1 are all faulty and all other nodes are fault-free. In this scenario, by condition D.1, all nodes in G_2 must agree on α.

Let the present fault scenario be (4). Thus, all nodes in F_2 are faulty. All the messages between G_1 and G_2 must be relayed through the nodes in $F = F_1 \cup F_2$. Now, the faulty nodes in F_2 can always corrupt the messages to G_1 in such a way that nodes in G_1 cannot distinguish between fault scenarios (3) and (4). As scenario (3) requires nodes in G_1 to agree on β, in the present scenario also, the nodes in G_1 must agree on β. At the same time, the faulty nodes in F_2 can also corrupt the messages to G_2 in such a way that nodes in G_2 cannot distinguish between fault scenarios (4) and (5). As scenario (5) requires nodes in G_2 to agree on α, in the present scenario also, the nodes in G_2 must agree on α. Thus, in the present scenario, the nodes in G_1 and G_2 end up agreeing on α and β respectively, where $V_d \neq \beta \neq \alpha \neq V_d$. This violates condition D.4 of degradable agreement.

The above two cases together prove that network connectivity of $m+u+1$ is necessary to achieve degradable agreement.

Proof of Sufficiency: For a network of connectivity $\kappa \geq m+u+1$, algorithm BYZ presented in Section 4 can be used with the modification described here. (Instead of the following procedure, a procedure similar to *purifying* scheme presented by Dolev [2] may also be used.)

The original sender and the final destination of each message are included in the message (in *sender* and *destination* fields). For every message sent between two nodes during algorithm BYZ, say between nodes A and B, the following procedure is used. Node A sends $m+u+1$ copies of the message to node B on $m+u+1$ disjoint paths. At least $m+u+1$ disjoint paths exist between A and B, as the network connectivity is $\kappa \geq m+u+1$ [1]. Node B and other nodes are aware of the paths used by A to send messages to B.

Assume that node D receives a message from node C. Also assume that the *sender* and *destination* fields in the message indicate that A is the original sender of the message and B is the final destination.[7] Then, node D determines whether the link (C,D) lies on one of the $m+u+1$ paths used by A to send a message to B. If (C,D) is included in one of these paths, then D forwards the message to the next node on that path, else D deletes the message.

The final destination, node B, accepts exactly one copy of a message on each of the $m+u+1$ paths (and discards others if more are received). When node B receives these $m+u+1$ copies,[8] it assumes that the message sent by node A has content given by VOTE$(u+1, m+u+1)$ of these $m+u+1$ copies. Two cases are possible.

Case 1: At most m nodes are faulty. Then, at least $u+1$ of the message copies must travel paths that do not include any faulty nodes. Therefore, node B must correctly receive at least $u+1$ copies of the message. Thus, VOTE$(u+1, m+u+1)$ of the $m+u+1$ message copies must be the correct message. In this case, because all messages between fault-free nodes are received correctly, algorithm BYZ remains correct.

Case 2: More than m but at most u nodes are faulty. Then, at most u copies of the message copies received by node B may contain incorrect contents. Therefore, when node B takes VOTE$(u+1, m+u+1)$ of these $m+u+1$ copies, it may either obtain the correct message or it may obtain V_d. This is equivalent to saying that, when more than m but at most u nodes are faulty, a fault-free node may incorrectly declare a message as absent. (Recollect that absent message are assumed to contain value V_d). In Section 4, algorithm BYZ was proved correct under this condition (i.e. assumption (d)). Therefore, connectivity of $t+u+1$ is sufficient.

[7]If the message was relayed through a faulty node before it reached D, the *destination* and *sender* fields in the message may be corrupted. The procedure described here works in spite of such corruption.

[8]If node B detects that a messages copy is absent, it assumes the message copy's content to be V_d.

B Algorithm Termination

When the number of faults is at most m, all the fault-free nodes will have synchronized logical clocks. Therefore, it is possible to determine an upper bound, say t_{upper}, such that agreement algorithm BYZ must terminate on each node by the time its physical clock reads t_{upper}. As shown below, algorithm termination within bounded time can be guaranteed with more than m faults if one of the following is true:

1. at least $m+1$ fault free nodes have their clocks synchronized and "approximate" real time, or

2. at least $m+1$ fault-free nodes detect (within bounded time) the presence of more than m faulty nodes.

To guarantee termination within bounded time, each fault-free node performs the following steps along with BYZ. In the following, a *termination* message is assumed to be a special type of message that can be distinguished from all other messages.

(i) If more than m faults are detected and the agreement algorithm has not terminated, then: terminate BYZ, agree on value V_d, and broadcast a *termination* message,

(ii) If physical clock $\geq t_{upper}$ and algorithm has not terminated, then: terminate BYZ, agree on value V_d, and broadcast a *termination* message,

(iii) If algorithm BYZ has terminated and physical clock $\geq t_{upper}$ then: broadcast a *termination* message.

(iv) If *termination* messages received from $m+1$ distinct nodes and algorithm BYZ has not terminated then: terminate BYZ, agree on V_d, and broadcast a *termination* message.

Let S be largest set of fault-free nodes that have their logical clocks synchronized (and the clocks approximate real time). Consider the following two situations, where f is the number of faulty nodes.

Case 1: $f \leq m$**.** In this case, S must contain all the fault-free nodes. Therefore, when the physical clock of a fault-free node reaches t_{upper}, it must have completed agreement algorithm BYZ. Thus, in this case, all nodes obviously complete BYZ within bounded time.

Case 2: $m < f \leq u$**.** In this case, by the two conditions stated at the beginning of this section, there are two possibilities.

(2.1) $m+1$ nodes have their logical clocks synchronized (and the clocks approximate real time). These $m+1$ nodes complete BYZ within bounded time and broadcast *termination* messages to all the nodes (using step (ii) or (iii) above). As message transmission takes bounded time (the bound is known), these messages reach a fault-free node within bounded time. A fault-free node terminates BYZ when it receives $m+1$ termination messages (if its algorithm has not completed already), as per step (iv) above. Thus, all fault-free nodes terminate BYZ within a bounded time.

(2.2) $m+1$ fault-free nodes detect the presence of more than m faults. These nodes will broadcast a *termination* message to all the nodes and also terminate their own algorithm BYZ (as per step (i) above). Thus, each fault-free node must receive at least $m+1$ termination messages and then terminate its algorithm BYZ (if it has not already completed), as per step (iv). Similar to case (2.1) above, here also, all fault-free nodes terminate BYZ within a bounded time.

References

[1] N. Deo, *Graph Theory with Applications to Engineering and Computer Science.* Englewood Cliffs NJ: Prentice-Hall, 1984.

[2] D. Dolev, "The Byzantine generals strike again," *J. Algo.*, pp. 14–30, 1982.

[3] D. Dolev, J. Y. Halpern, and H. R. Strong, "On the possibility and impossibility of achieving clock synchronization," *J. Computer and System Sciences*, vol. 32, pp. 230–250, 1986.

[4] M. J. Fischer, N. A. Lynch, and M. Merritt, "Easy impossibility proofs for distributed consensus problems," in *Fourth ACM Conf. Distr. Comp.*, pp. 59–70, 1985.

[5] R. E. Harper, J. H. Lala, and J. J. Deyst, "Fault tolerant parallel processor architecture overview," in *Digest of papers: The* 18^{th} *Int. Symp. Fault-Tolerant Comp.*, pp. 252–257, 1988.

[6] C. M. Krishna and I. S. Bhandari, "On graceful degradation of phase locked clocks," in *IEEE Real-Time Systems Symposium*, pp. 202–211, 1988.

[7] L. Lamport, "The weak Byzantine generals problem," *J. ACM*, vol. 30, pp. 668–676, July 1983.

[8] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Prog. Lang. Syst.*, vol. 4, pp. 382–401, July 1982.

[9] J.-F. Paris, "Voting with witnesses: A consistency scheme for replicated files," in *International Conf. Distributed Computing Systems*, pp. 606–612, 1986.

[10] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design.* Digital Press, Bedford, MA, 1982.

[11] T. B. Smith III et al., *The Fault-Tolerant Multiprocessor Computer.* Park Ridge, NJ: Noyes Publications, 1986.

[12] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, pp. 626–645, July 1987.