A Tradeoff Between Safety and Liveness for Randomized Coordinated Attack Protocols

George Varghese* Nancy A. Lynch †

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

We study *randomized, synchronous* protocols for coordinated attack. Such protocols trade off the number of rounds (N), the worst case probability of disagreement (U), and the probability that all generals attack ($\mathcal{L}$). We prove a nearly tight bound on the tradeoff between $\mathcal{L}$ and U ($\mathcal{L}/U \leq N$) for a *strong* adversary that destroys any subset of messages. Our techniques may be useful for other problems that allow a non-zero probability of disagreement.

1 Introduction

Suppose two computers are trying to perform a database transaction over an unreliable telephone line. If the line goes dead at some crucial point, standard database protocols mark the transaction status as "uncertain" and wait until communication is restored to update its status. The protocol will ensure that the two computers *eventually* agree if communication is eventually restored.

On the other hand, suppose that the transaction has a real time constraint (e.g., a decision to commit or reject the transaction must be reached in 10 minutes) and the cost of disagreement is high. Then standard commit protocols do not work. If communication can fail for up to ten minutes it is always possible for the two computers to disagree. Is there a protocol that prevents disagreement in all cases?

The answer is no. The question was first formalized in [G] as the coordinated attack problem. In this problem, there are two generals who communicate only using unreliable messengers. The generals are initially passive; however, at any instant either general may get an input signal that instructs him to try to attack a distant fort. The generals have a common clock. The problem is to synchronize attack attempts subject to the conditions:

- **Validity**: If no input signal arrives, neither general attacks.[1]
- **Agreement**: Either both generals attack or they both do not attack.
- **Nontriviality**: There is at least one execution of the protocol in which both generals attack.

It is shown in ([G], [HM]) that there is no *deterministic* algorithm that meets all three conditions. In this paper, we consider a generalization to an arbitrary number of generals connected by a graph of unreliable links. Clearly the impossibility result applies here as well.

Coordinated attack (CA) looks suspiciously like Byzantine agreement (BA) [LPS]. The major differences are: first, in BA, *generals* exhibit arbitrary failures while in CA only *links* fail by destroying messages; second, in BA only *some fraction* of the generals are assumed to be faulty while in CA *all* links can be faulty. Thus there does not appear to be any way to reduce CA to BA or vice versa.

There is a well-known history of *randomization* providing a cure for a *deterministic* impossibility result

*Supported by DEC Graduate Education Program.
†Supported by NSF Grant 8915206-CR, DARPA Grant N0014-89-J1988, and ONR Grant N0014-91-J1046.

[1]Another validity condition that is often used is that if no messages are delivered, then no general attacks. We prefer our definition because it focuses on input-output behavior. However, our results can be modified to fit the other validity condition.

(e.g. [RL], [B]). Thus we we turn to *randomized CA*. We hope to trade a small probability of disagreement when links fail for a high probability of agreement (on a positive outcome) when links do not fail.

We modify the correctness conditions for deterministic CA to fit randomized CA. We retain the validity condition. We modify the agreement condition by requiring that the worst case probability of disagreement (denoted by U) be smaller than ϵ, a parameter. We replace the nontriviality condition by a measure $\mathcal{L}(R)$ (for liveness) that measures the probability all generals attack after an input signal, given that messages are delivered according to a given pattern R.[2] We measure the goodness of a CA protocol by seeing how high $\mathcal{L}(R)$ can be for a given R and ϵ.

Coordinated attack captures the fundamental difficulty of real-time synchronization over unreliable message channels. This paper investigates whether randomization can help coordinated attack. Our answer is basically no for nontrivial adversaries, and a qualified yes for much weaker adversaries. Our paper concentrates on a *strong adversary* that can deliver messages according to any possible pattern R but has no access to message bits.[3]

The rest of this paper is organized as follows. Section 2 contains our model, Section 3 describes a simple but inefficient protocol, and Section 4 introduces some useful concepts. Section 5 contains a basic lower bound, Section 6 describes an optimal protocol against a strong adversary, and Section 7 contains a second, more refined, lower bound. Section 8 contains our conclusions and the appendix contains a proof of the second lower bound.

2 Model

The generals are represented by processes i that are at the vertices of a undirected graph $G(E, V)$ with $V = \{1, \ldots, m\}$, $m \geq 2$. We consider synchronous protocols that work in $N + 2$ rounds, numbered $-1, 0, \ldots, N$, $N \geq 1$. We model the input as a message sent at the end of a fictitious Round -1 and arriving at the end of Round 0 from a fictitious "environment" node v_0. We assume $v_0 \notin V$. Informally, if a process i receives a message in Round 0 from v_0, it has received a signal to try to attack. Each process i also receives a sequence of J random bits called α_i. J is an upper bound on the total number of random bits used by any general.

A *protocol* F consists of a number of *local protocols* F_i. Each $F_i, i \in V$, is a state machine executed by process i. F_i has two possible start states s_i^0 and s_i^1, a state transition function δ_i, and a message generation function σ_i. Let S_i^r be the set of messages *received* by i from its neighbors in round r. Let q_i^r be the state of i at the end of round r. Then $q_i^r = \delta_i(q_i^{r-1}, r, S_i^r, \alpha_i)$. We assume without loss of generality that processes send messages to each neighbor in rounds $1 \ldots N$ since we can always simulate algorithms in which this is not true by sending null messages that are ignored by the receiver. Let m_{ij}^r be the message *sent* by i to neighbor j in round r. Then $m_{ij}^r = \sigma_i(q_i^{r-1}, j)$. At the end of N rounds, i outputs a bit O_i based on q_i^N. $O_i = 1$ iff i decides to attack.

An execution of F is described in terms of a vector of local executions. A *local execution* E_i consists of q_i^0, (q_i^r, S_i^r, m_{ij}^r) for $1 \leq r \leq N$, and O_i. To generate an execution of F we need to define a run that represents the inputs as well as which messages get through in rounds $1 \ldots N$ of the protocol. Formally, a *run* $R = I(R) \cup M(R)$. $I(R)$, the *input* for run R, is an arbitrary subset of $\{(v_0, i, 0) : i \in V\}$. $M(R)$, the *messages delivered* in run R, is an arbitrary subset of $\{(i, j, r) : (i, j) \in E, 1 \leq r \leq N\}$. For example, in the run $\{(v_0, 3, 0), (1, 2, 6), (3, 2, 7)\}$ only F_3 receives a signal to attack. Also only the message sent in Round 6 from F_1 to F_2 and any message sent in Round 7 from F_3 to F_2 are delivered: all other sent messages are lost.

We will use the notation (A_i) to denote a vector A consisting of a component A_i for each $i \in V$. An execution for a fixed F is uniquely specified by random input $\alpha = (\alpha_i)$, and a run R. We define $Ex(R, \alpha) = (E_i)$ as the *execution generated* by R and α for a fixed protocol F. Each E_i is a local execution such that:

- If $(v_0, i, 0) \notin R$ then $q_i^0 = s_i^0$. If $(v_0, i, 0) \in R$ then $q_i^0 = s_i^1$ (i.e., the initial state of the local execution encodes the input).
- For all $r, 1 \leq r \leq N$: $m_{ij}^r = \sigma_i(q_i^{r-1}, j)$.
- For all $r, 1 \leq r \leq N$: $m_{ji}^r \in S_i^r$ iff $(j, i, r) \in R$.
- For all $r, 1 \leq r \leq N$: $q_i^r = \delta_i(q_i^{r-1}, r, S_i^r, \alpha_i)$.

The *output* of execution E is the vector $(O_i(q_i^N))$. We say two executions E and $\tilde{E}$ are *identical* to j if $E_j = \tilde{E}_j$.

[2] It may seem strange that unsafety is measured as the worst case across all runs while liveness is measured separately for each run. However, the situation is similar to Data Link protocols in which the prefix property (safety) is always preserved but liveness is guaranteed only if the channel is delivering messages.

[3] Since our lower bounds are pessimistic, there is no point in considering a stronger adversary that can read message bits. Also, some form of encryption could be used to make this assumption reasonable.

We consider sets of executions of a particular protocol. If X and Y are sets of executions, then XY denotes $X \cap Y$, and $X + Y$ denotes $X \cup Y$. D_i denotes the set of executions in which $O_i(q_i^N) = 1$, and $\overline{D_i}$ the set of executions in which $O_i(q_i^N) = 0$. Similarly, $(D_i|R)$ denotes the set of executions that have run R and in which $O_i(q_i^N) = 1$.

TA (*total attack*) denotes the set of executions $D_1 D_2 \ldots D_m$. NA (*no attack*) denotes the set of executions $\overline{D_1}\ \overline{D_2} \ldots \overline{D_m}$. PA (*partial attack*) denotes the complement of $NA \cup TA$. Thus, TA is the set of executions in which all processes agree on an output of 1, NA is the set of executions in which all processes agree on an output of 0, and PA is the set of executions in which some pair of processes disagree.

Each α_i is drawn from $\{0,1\}^J$ using the uniform probability distribution. This probability distribution on inputs α induces a probability distribution on executions for each possible run R, in the natural way. For each set X of executions and each run R, we use the notation $Pr[X|R]$ to denote the probability of event X according to this distribution of executions.

Now consider two runs $R = \{(i,j,1)\}$ and $\tilde{R} = \emptyset$. The only difference in the runs is that i sends a message that is delivered in R. Thus, given the same random input, i will decide the same regardless of whether an execution follows run R or run $\tilde{R}$. This leads to a key notion of indistinguishable runs. We say that two runs R and $\tilde{R}$ are *indistinguishable* to i if for all α, $Ex(R,\alpha)$ and $Ex(\tilde{R},\alpha)$ are identical to i. We use $R \overset{i}{\equiv} \tilde{R}$ to denote that R and $\tilde{R}$ are indistinguishable to i. A natural consequence is:

Lemma 2.1 If $R \overset{i}{\equiv} \tilde{R}$ then $Pr[D_i|R] = Pr[D_i|\tilde{R}]$.

An *adversary* $\mathcal{A}$ is a set of runs. We will only deal in this paper with a *strong adversary*, $\mathcal{A}_s$, where $\mathcal{A}_s$ is the set of all possible runs.

Next, we describe the correctness conditions and the liveness measure. Validity requires that no process attacks if there is no input. Agreement requires that the worst-case probability of partial attack be no more than ϵ, a parameter. Finally the liveness measure for a run R is the probability of total attack on run R.

- **Validity** : A protocol satisfies validity if for all vectors α, for all R such that $I(R) = \emptyset$, and for all i: $O_i = 0$ in $Ex(R,\alpha)$.
- **Agreement**: We define $U_{\mathcal{A}}(F)$, the *unsafety* of protocol F against adversary $\mathcal{A}$, as: $U_{\mathcal{A}}(F) = Max_{R \in \mathcal{A}} Pr[PA|R]$. Then F satisfies agreement with parameter ϵ if $U_{\mathcal{A}}(F) \leq \epsilon$.
- **Liveness**: We define *liveness* $\mathcal{L}(F,R)$ of protocol F on run R by: $\mathcal{L}(F,R) = Pr[TA|R]$.

Our goal is to find an "optimal" algorithm F that meets the validity and agreement conditions, and such that $\mathcal{L}(F,R)$ is as large as possible for any run R. We end this section with two elementary lemmas on which our lower bounds are based. The first states that the unsafety is at least as large as the difference in attack probabilities of any two processes. The second states that the liveness is no more than the attack probability of any process. The two inequalities given below do not seem very tight, and so it is perhaps surprising that the lower bounds based on these inequalities are as tight as they are.

Lemma 2.2 For all $i,j \in V$, $Pr[D_i|R] - Pr[D_j|R] \leq U_s(F)$.

Lemma 2.3 For all $i \in V$, $\mathcal{L}(F,R) \leq Pr[D_i|R]$.

3 Example Protocol

We informally describe a simple protocol A for two processes 1 and 2 against a strong adversary. The limitations of this protocol will motivate both the lower bound in Section 5 and the optimal protocol of Section 6.

In order to conform to the model, we require that each process must send some message (at least a null message) in every round. For convenience, let us call a non-null message (i.e., a message that carries information) a packet. We assume implicitly that on every round a process sends either a packet or a null message.

Initially, at the start of round 0, process 1 chooses a random integer *rfire* that is uniformly distributed between 2 and N. Process 1 includes the value of *rfire* in any packet it sends. If process 2 receives any packet from process 1, process 2 will store the value of *rfire*.

In rounds 1 through N, the two processes send packets to each other in alternate rounds. Process 2 is allowed to send packets in odd rounds starting from round 1, while process 1 is allowed to send packets in even rounds. The protocol begins with process 2 sending a packet in round 1. However, in all later rounds, a process sends a packet in a round only if it has received a packet in the previous round, and it is allowed to send a packet in the round. Thus if the adversary destroys a packet sent in round r, all packet sending stops in rounds greater than r.

The main idea is that if all packets sent strictly before round number *rfire*, have been delivered, then the process that received the last packet (say i) will decide to attack. If the next packet sent by process i is delivered then the other process (say j) will also decide to attack. On the other hand, if any packet sent before round *rfire* is destroyed, then both processes stop sending packets and do not attack. Since the adversary that controls message delivery does not know the value of *rfire*, the adversary has only a chance of approximately $1/N$ of causing partial attack. This is because the adversary can cause partial attack *only if the first packet destroyed in the run is the packet sent in round rfire.* Thus $U_s(A) \approx 1/N$.

In addition, process 2 includes a bit that encodes its input in the packets it sends. Suppose at the end of Round 1, process 1 has not received a signal to attack and has not received a packet from process 2 saying that process 2 has received a signal to attack. Then process 1 does not send a packet in Round 2, and the protocol stops. Thus protocol A satisfies validity. Finally, let R_g be a "good" run in which all messages are delivered and the input is valid. Then on run R_g, both processes will always decide to attack. Hence $\mathcal{L}(A, R_g)$, the liveness of A on run R_g, is 1. However, this simple protocol raises two questions:

- $U_s(A) \approx 1/N$ and $\mathcal{L}(A, R_g) = 1$. Can we decrease $U_s(A)$ further while keeping $\mathcal{L}(A, R_g)$ unchanged? In other words, can we find a protocol a) whose probability of making a mistake is better than $1/N$, and b) whose probability of attacking on a good run is 1. It might seem that this can be done by running A several times. However, the answer is no, as we show in Section 5.

- Consider a run R in which the input is valid and all messages are delivered except the message sent by process 1 in Round 2. It is easy to see that $\mathcal{L}(A, R) = 0$. Intuitively, this is not satisfactory because in run R, all but one message is delivered, and yet the probability of attacking on run R is 0. Can we design a protocol whose liveness grows in some fashion with the number of messages delivered in a run? We will describe an "optimal" protocol S in Section 6.

4 Information Flow, Clipping, and Information Level

In this section, we describe three concepts that underlie both the lower bounds of Section 5 and the protocol in Section 6. We begin with a definition that captures the usual idea of information flow or possible causality [L] between process-round pairs in a run.

Consider any $i, k \in V \cup \{v_0\}$ and any $r, s \in \{-1, 0, \ldots, N\}$. We say that (i, r) directly flows to (k, s) in run R iff $s = r + 1$ and either $i = k$ or $(i, k, s) \in R$. We define the *flows to* relation between process-round pairs as the reflexive transitive closure of the directly flows-to relation. Thus:

Lemma 4.1 If (i, r) flows to (j, s) and (j, s) flows to (k, t) in run R, then (i, r) flows to (k, t) in run R.

We introduce a measure of the "knowledge" [HM] a process has in a run. We first define information "height" and use it to define the more useful idea of information "level". Intuitively, a process reaches height 1 when it hears the input. A process reaches height $h > 1$ when it has heard that *all* other processes have reached height $h - 1$. More formally, we say that j can reach height h by round r in run R iff h is a nonnegative integer subject to the following conditions:

- If $h = 0$, there are no conditions.
- If $h = 1$, $(v_0, -1)$ flows to (j, r) in R.
- If $h > 1$, then for all $i \neq j \in V$, there is some r_i such that (i, r_i) flows to (j, r) in R and i can reach height $h - 1$ by round r_i in R.

Next, we define $L_j^r(R)$, the *level* j reaches by round r of run R, to be the *maximum* height j can reach by round r. We use $L_j(R)$ to denote[4] $L_j^N(R)$ and $L(R)$ to denote $Min_{j \in V}(L_j(R))$.

Finally, we introduce a construction to "clip" a run with respect to a process i such that the constructed run preserves all information flow to i. This construction is the key to the lower bound proof. We define $Clip_i(R) = \{(j, k, r) \in R : (k, r) \text{ flows to } (i, N)\}$ in run R. It is not hard to see that clipping with respect to i preserves any information that i can gather in the run. Hence we have:

Lemma 4.2 Let $Clip_i(R) = \tilde{R}$. Then $L_i(R) = L_i(\tilde{R})$ and $R \overset{i}{\equiv} \tilde{R}$.

5 Lower Bound for Strong Adversary

The first lemma captures the intuitive idea that a change in level can only come about by receiving a message.

[4]Recall that N is the maximum round number

Lemma 5.1 For any run R and any $k \in V$, if $L_k(R) = l > 0$ then there must be some tuple $(j, k, r) \in R$ such that $L_k^r(R) = l$.

Proof: From the definition of level, we see that if there is no j, s such that $(j, k, s) \in R$ then $L_k^{s-1}(R) = L_k^s(R)$. Thus if $L_k^N(R) = l$ we can work backwards from round number N until we find the r required for the lemma. If we fail then there is no $(*, k, *)$ tuple in R, which would imply that $l = 0$, a contradiction. Thus we cannot fail. $\square$

The next lemma describes the key property of clipped runs and information levels that we use to prove our lower bound. It says that if i reaches information level l at the end of run R then at the end of $Clip_i(R)$ there must be some process k whose information level is no more than $l-1$. In essence, this is why i cannot go to a higher information level than l by the end of R.

Lemma 5.2 Consider a run R such that $L_i(R) = l > 0$ and $Clip_i(R) = \tilde{R}$. Then there is some $k \in V$ such that $L_k(\tilde{R}) \leq l - 1$.

Proof: By contradiction. Thus for all $k \in V$, we assume that $L_k(\tilde{R}) \geq l$.

Consider any $k \neq i$. By Lemma 5.1 and the fact that $l > 0$, there must be some tuple $(j, k, r) \in \tilde{R}$ such that $L_k^r(\tilde{R}) \geq l$. Since $(j, k, r) \in \tilde{R}$ then (by definition of clipping), (k, r) flows to (i, N) in R. Hence, we can show that (k, r) flows to (i, N) in $\tilde{R}$. We also know that $L_k^r(\tilde{R}) \geq l$. Since this is true for all $k \neq i$ we must have (see the definition of level) $L_i(\tilde{R}) \geq l + 1$. But by Lemma 4.2, this implies that $L_i(R) \geq l + 1$, a contradiction. $\square$

Lemma 5.3 For all protocols F, all runs R, and any process index $i \in V$, $Pr[D_i|R] \leq U_s(F)L_i(R)$.

Proof: By induction on l in the following inductive hypothesis.

Inductive hypothesis: For all i and all runs R with $L_i(R) = l$, $Pr[D_i|R] \leq U_s(F)l$.

Base case, $l = 0$: Thus $L_i(R) = 0$. Let $\tilde{R} = Clip_i(R)$. We first claim that $I(\tilde{R}) = \{\}$. Suppose not for contradiction. Then there is some j such that $(v_0, j, 0) \in \tilde{R}$; hence, since $\tilde{R} \subseteq R$, $(v_0, j, 0) \in R$. Also by the definition of clipping, $(j, 0)$ flows to (i, N) in R. But in that case, $L_i(R) \geq 1$, a contradiction. Thus we must have $I(\tilde{R}) = \{\}$. Also by Lemma 4.2, $R \overset{i}{\equiv} \tilde{R}$. Hence $Pr[D_i|R] = Pr[D_i|\tilde{R}] = 0$, by Lemma 2.1 and the validity requirement. Thus $Pr[D_i|R] = U_s(F)L_i(R)$.

Inductive Step, $l > 0$: Consider any l and R such that $L_i(R) = l$. Let $\tilde{R} = Clip_i(R)$. By Lemma 5.2, there exists some k such that $L_k(\tilde{R}) \leq L_i(R) - 1$. Hence, by the inductive hypothesis, $Pr[D_k|\tilde{R}] \leq U_s(F)(l-1)$. But by our bound on unsafety, Lemma 2.2, $Pr[D_i|\tilde{R}] - Pr[D_k|\tilde{R}] \leq U_s(F)$. Hence $Pr[D_i|\tilde{R}] \leq U_s(F)l$. But by the fact that R and $\tilde{R}$ are indistinguishable to i and by Lemma 2.1, it follows that $Pr[D_i|R] \leq U_s(F)l$. $\square$

Theorem 5.4 For any F, $\mathcal{L}(F, R) \leq U_s(F)L(R) \leq \epsilon L(R)$.

From Lemma 5.3, for any $i \in V$, $Pr[D_i|R] \leq U_s(F)L_i(R)$. Thus from Lemma 2.3, $\mathcal{L}(F, R) \leq U_s(F)L_i(R)$ for any $i \in V$. Thus from the definition of $L(R)$, $\mathcal{L}(F, R) \leq U_s(F)L(R)$. The theorem now follows from the agreement condition. $\square$

6 Optimal Protocol Against a Strong Adversary

In Protocol S which we describe below, we will arbitrarily designate process 1 to choose a random number *rfire*. In order to attack, we will require that any other process i hear the value of *rfire* from process 1 in addition to hearing the input. This motivates a second measure on a run R that we call the *modified level* measure. It is defined in a parallel fashion to the original level measure by first defining a modified height or m-height. Formally, we say that process j can *reach m-height* h by round r in run R iff h is a nonnegative integer subject to the following conditions:

- If $h = 0$, there are no conditions.
- If $h = 1$, $(v_0, -1)$ and $(1, 0)$ flow to (j, r) in R.
- If $h > 1$, then for all $i \neq j \in V$, there is some r_i such that (i, r_i) flows to (j, r) in R and i can reach m-height $h - 1$ by round r_i in R.

Thus the only difference between the m-height and height definitions is in the condition required to reach m-height 1. In the case of m-height we not only require that j has heard the input but also that j has heard from process 1. We also define $ML_i^r(R)$, $ML_i(R)$, $ML(R)$ analogously to the previous definitions for L_i.

Because of the small difference in the definitions, it is easy to show that the modified level measure differs by at most one from the level measure. Also the modified level measured by any two processes can differ by at most one.

Lemma 6.1 For all R and $i \in V$, $L_i(R) - 1 \leq ML_i(R) \leq L_i(R)$.

Lemma 6.2 For all R and $i, j \in V$, $ML_j(R) \geq ML_i(R) - 1$.

We will design a protocol based closely on the lower bound arguments of the previous section. Recall that we had shown that for any F, $\mathcal{L}(F, R) \leq \epsilon L(R)$. We have also seen that the modified level measure differs by at most one from the level measure. Thus in order to come close to meeting the lower bound, we will design a protocol in which:

- Each process i will calculate $ML_i(R)$, the value of the modified level at the end of the current run R.
- Each process will decide to attack with a probability proportional to $ML_i(R)$. This causes the liveness of the protocol to grow with $ML_i(R)$.

To do so each process i in protocol S has a variable $count_i$ that counts the value of $ML_i^r(R)$. We say that i has begun counting if $count_i > 0$. We will see how i begins counting below. However, once i has begun counting, process i increases $count_i$ to s (for $s > 1$) when it has heard that all other processes have reached a count of $s-1$. It is easy to implement this if each message sent by a node i carries $count_i$. and a variable called $seen_i$, the set of nodes that i knows has reached $count_i$.

Protocol S must satisfy agreement with parameter ϵ. Let $t = 1/\epsilon$. Process 1 chooses a random number *rfire* uniformly distributed in the range $(0, t]$ and passes it on all messages. After N rounds, i decides to attack if i has heard the value of *rfire* from process 1 and $count_i \geq rfire$.

Process i starts counting (i.e., sets $count_i$ to 1) in round r as soon it finds out that $(v_0, -1)$ and $(1, 0)$ flows to (i, r). We have discussed the reason for the second condition. The first condition, of course, is imposed to ensure validity. To implement the first condition, we use a variable $valid_i$ at each process i that is set to *true* in the first round r such that $(v_0, -1)$ flows to (i, r). To implement the second condition, all processes other than process 1 initially set the value of $rfire_i$ to a special value *undefined* which is updated when a message is received with the value of *rfire*.

6.1 Protocol Code

Protocol S consists of local state machines, each of which has a set of states, an initial state, a state transition function, a message generation function, and an output decision function. We describe each component in turn:

Each process i has the following state variables:

- $count_i$: integer between 1 and N (counts the value of $ML_i^r(R)$ in the current run R.).
- $rfire_i$: either a default value of *undefined* or a real number in the range $(0, 1/\epsilon]$. We assume that the value of *undefined* is not in $(0, 1/\epsilon]$.
- $seen_i$: a subset of V (represents the processes that have reached $count_i$ that i knows about).
- $valid_i$: a boolean (that is true if i has heard from v_0.)

We also use three temporary variables at each process: $highcount_i$ (an integer), $highseen_i$ (a subset of V), and $highset_i$ (a set of messages, whose format we describe later.)

The initial states are as follows. Process 1 initially sets $rfire_1$ to a a random number uniformly distributed in the range $(0, 1/\epsilon]$. All processes i other than 1, set $rfire_i = undefined$. The $valid_i$ bit is only set if process i has received an input message from v_0 in Round 0. Finally process 1 sets $count_1 = 1$ iff $valid_1 = 1$. All other processes i initially set $count_i = 0$.

A message is denoted by m and has fields $m(rfire)$, $m(count)$, $m(seen)$, and $m(valid)$. The message generation function for i in every round sends a message $m(rfire, count, seen, valid)$ to all neighbors with $m(rfire) = rfire_i$, $m(count) = count_i$, $m(seen) = seen_i$, $m(valid) = valid_i$. Thus i sends a message with its current state to all neighbors in every round.

At the end of a round r, for $1 \leq r \leq N$, process i executes the procedure PROCESS-MESSAGE(S_i, i) where S_i is the set of messages process i has received in round r. PROCESS-MESSAGE(S_i, i) is shown in Figure 1. The first four lines are used to decide when a process starts counting; the remainder of the code does the actual counting.

Finally at the end of N rounds, $O_i = 1$ iff $rfire_i \neq undefined$ and $count_i \geq rfire_i$.

6.2 Proof of Properties of Protocol S

Notation: Consider any execution $Ex(R, \alpha)$. Let v^r denote the value of a variable at the end of round r. For example, $count_i^r$ denotes the value of $count_i$ at the end of r rounds. Define *rfire* to be the value of $rfire_1$ in the initial state.

Our first major step will be to establish that $count_i^r = ML_i^r(R)$. To allow a careful inductive proof,

```
PROCESS-MESSAGE($S_i, i$)

If ($rfire_i = undefined$) and ($\exists m \in S_i$ :
    $m(rfire) \neq undefined$) then $rfire_i := m(rfire)$
If ($valid_i = false$) and ($\exists m \in S_i : m(valid) = true$)
    then $valid_i := true$
If ($valid_i = true$) and ($rfire_i \neq undefined$)
    and ($count_i = 0$) then $count_i := 1$

If ($count_i \geq 1$) and ($S_i \neq \emptyset$) then
    $highcount := Max_{m \in S_i} m(count)$
    $highset := \{m \in S_i : m(count) = highcount\}$
    $highseen := \cup_{m \in highset} m(seen)$
    If $highcount = count_i$ then
        $seen_i := seen_i \cup highseen \cup \{i\}$
    Else
        If $highcount > count_i$ then
            $seen_i := highseen \cup \{i\}$;
            $count_i := highcount$;
    If $seen_i = V$ then
        $count_i := count_i + 1$;
        $seen_i := \{i\}$;
```

Figure 1: Procedure executed by process i at the end of a round in Protocol S

we will introduce invariants. The invariants should be intuitively clear from the previous discussion. The proofs of these invariants are deferred to the final paper.

Lemma 6.3 For any execution $Ex(R, \alpha)$ of Protocol S, the following assertions are true for $0 \leq r \leq N$ and for all $i, j \in V$:

1. $rfire_i^r$ is either equal to *rfire* or *undefined.*
2. $count_i^r \geq 1$ iff $rfire_i^r = rfire$ and $valid_i^r = true$.
3. $(1, 0)$ flows to (i, r) iff $rfire_i^r = rfire$.
4. $(v_0, -1)$ flows to (i, r) iff $valid_i^r = true$.
5. If (j, s) flows to (i, r) in R then either ($count_i^r > count_j^s$) or ($j \in seen_i^r$ and $count_i^r = count_j^s$) or ($count_i^r = count_j^s = 0$).
6. If ($j \in seen_i^r$) then there is some s such that ($count_j^s = count_i^r$) and (j, s) flows to (i, r) in R.
7. $seen_i^r \neq V$ and $seen_i^r \neq V - \{i\}$. Also, if $count_i^r \geq 1$ then $i \in seen_i^r$.
8. $ML_i^r \geq count_i^r$.

These invariants can now be used to establish that each process counts a value equal to its modified level measure. This should not be hard to believe since the code follows the definition of modified level.

Lemma 6.4 For all $i \in V$, any r such that $0 \leq r \leq N$, and any execution $Ex(R, \alpha)$ of Protocol S: $count_i^r = ML_i^r(R)$.

Proof: From the last invariant in Lemma 6.3, we see that $count_i^r \leq ML_i^r(R)$. So we show that $count_i^r \geq ML_i^r(R)$. We do so by induction on the value of $ML_i^r(R)$.

First if $ML_i^r(R) = 0$ we are done trivially since $count_i^r$ is always nonnegative. We use $ML_i^r(R) = 1$ as the base case. Then from the definition of $ML_i^r(R)$, we know that $(v_0, -1)$ and $(1, 0)$ flow to (i, r) in run R. Hence by the third and fourth invariants in Lemma 6.3, $rfire_i^r = rfire$ and $valid_i^r = true$. Hence by the second invariant in Lemma 6.3, $count_i^r \geq 1$.

Next, suppose $ML_i^r(R) = l > 1$. Then from the definition of $ML_i^r(R)$, we know that for all $j \neq i$ there exists $r_j < r$ such that (j, r_j) flows to (i, r) in run R and $ML_j^{r_j} = l - 1$. Hence by the fifth invariant in Lemma 6.3 and the inductive hypothesis, either $count_i^r > l - 1$ (in which case we are done) or for all $j \neq i$, $j \in seen_i^r$. But the second case contradicts the seventh invariant in Lemma 6.3, and so we are done. □

Next we sketch proofs of the validity, unsafety, and liveness properties of S.

Theorem 6.5 Protocol S satisfies validity.

Proof: Informally, in any execution in which no process receives an input signal, no process hears from v_0, and so $count_i^N = 0$ for all i. Thus by the output decision function, $O_i = 0$ for all i in this execution.

More formally, fix a run R such that $I(R) = \{\}$, a random vector α, and any process i. Consider the execution $Ex(R, \alpha)$. Thus $(v_0, -1)$ does not flow to (i, N) for any $i \in V$. Thus by Invariant 4 in Lemma 6.3, $valid_i^N = false$. Hence by Lemma 6.3, Invariant 2, $count_i^N < 1$. Hence $count_i^N = 0$. However, $rfire_i^N$ by Invariant 1, Lemma 6.3, is either equal to *rfire* (which is strictly greater than 0) or *undefined.* In either case, by the output decision function, $O_i = 0$ in $Ex(R, \alpha)$. □

To prove the unsafety and liveness properties of S we characterize when the total attack and no attack events occur. Let *Mincount* be the minimum across all processes i of the value of $count_i$ at the end of an execution. The next lemma states that all processes will attack if *Mincount* is no less than *rfire*, and no process will attack if *Mincount* is strictly less than *rfire* $- 1$;

Lemma 6.6 Fix an execution E of Protocol S. If *Mincount* $\geq$ *rfire* then $E \in TA$; but if *Mincount* $<$ *rfire* $- 1$ then $E \in NA$.

Proof: If *Mincount* $\geq$ *rfire* then for all processes i, $count_i^N \geq rfire$. But $rfire > 0$, hence for all i, $count_i^N \geq 1$. Hence (by Lemma 6.3, invariant 2), for all i, $rfire_i^N = rfire$. Hence for all i, $count_i^N \geq rfire_i^N$ and $rfire_i^N \neq undefined$. Hence for all i, (by the decision function), $O_i = 1$. Hence, $E \in TA$.

If *Mincount* $<$ *rfire* $- 1$, then using Lemma 6.4 and using the fact that the modified level measured at any two processes differs by at most 1 (Lemma 6.2), for all i, $count_i^N < rfire$. Now (by Lemma 6.3, Invariant 1), either $rfire_i^N = rfire$ or $rfire_i^N = undefined$. Hence, for all $i \in V$, either $count_i^N < rfire_i^N$ or $rfire_i^N = undefined$. Thus by the definition of the output decision function $O_i = 0$ for all i. Hence $E \in NA$. $\square$

Theorem 6.7 S satisfies agreement with parameter ϵ.

Proof: By definition $U_s(S)$ is the maximum across all runs R of $Pr[PA|R]$. Consider any execution $E = Ex(R, \alpha)$. Now partial attack PA is the complement of the no attack and total attack events, NA and TA. From Lemma 6.6, we know that either TA or NA will occur unless *Mincount* $<$ *rfire* $\leq$ *Mincount*$+1$. Hence $Pr[PA|R] \leq Pr[Mincount < rfire \leq Mincount+1|R]$. Now for a given R, *Mincount* is fixed while *rfire* is a uniformly distributed random number in the range $(0, 1/\epsilon]$. Thus $U_s(s) \leq \epsilon$. $\square$

Theorem 6.8 $\mathcal{L}(S, R) \geq Min(1, \epsilon ML(R))$.

Proof: Recall the definition of $\mathcal{L}(S, R)$ as the probability of total attack, $Pr[TA|R]$. We find a lower bound on $Pr[TA|R]$. Consider any execution E. From Lemma 6.6, $E \in TA$ if *Mincount* $\geq$ *rfire*. But by Lemma 6.4 and the definition of *Mincount*, *Mincount* $= ML(R)$. Hence, $E \in TA$ if $ML(R) \geq$ *rfire*. Thus for any run R, $Pr[TA|R]$ is no less than $Pr[ML(R) \geq rfire|R]$. Now for a given R, $ML(R)$ is fixed while *rfire* is a uniformly distributed random number in the range $(0, 1/\epsilon]$. Thus $Pr[TA|R]$ is no less than $Min(1, \epsilon ML(R))$. $\square$

7 Closing the Gap: A Second Lower Bound

Theorem 5.4 states that for every run R and every protocol F, the liveness $\mathcal{L}(F, R)$ of any protocol F is at most $Min(1, \epsilon L(R))$. We described a protocol S whose liveness is $Min(1, \epsilon ML(R))$. From Lemma 6.1, we know that $ML(R)$ differs from $L(R)$ by at most one. Thus we have a small but irritating gap of ϵ. Our second lower bound shows, under a reasonable set of conditions that we call the usual case assumption, that no protocol F can do better than $\epsilon ML(R)$ on all runs R. More precisely, if any protocol F has a run R such that $\mathcal{L}(F, R) > \epsilon ML(R)$ then there is some other run $\tilde{R}$ such that $\mathcal{L}(F, \tilde{R}) < \epsilon ML(\tilde{R})$. Thus together the two bounds show that Protocol S is indeed "optimal".

A precise description of the second lower bound is in the appendix. We note that the proof of the first lower bound is similar to the chain arguments used often in deterministic impossibility results (e.g., [FL]). However, in proving the second lower bound, we are led to some connections between causality, probabilistic independence, and probabilistic agreement that may be interesting in their own right.

8 Conclusions

A strong adversary can be used to model a situation where links can crash and restart at an arbitrary frequency. A solution to coordinated attack is important in situations where consensus must be reached across unreliable links and within a specified time constraint. For coordinated attack against a strong adversary, we have seen that no protocol can achieve a tradeoff between liveness and safety ($\mathcal{L}/U$) that is better than linear in the number of rounds. This is bad news. For example if we want to achieve liveness with probability 1 on some run, and yet limit the probability of error to be less than 0.001, then the protocol must run for at least 1000 rounds. Protocol S demonstrates that the lower bounds are tight, but its performance is far from adequate. While our results are stated in a synchronous model, it seems clear that they can be extended to an asynchronous model.

In practice, there are two approaches that may help us to overcome these limitations. One approach is to add redundant links and assume that failures can only affect some fraction of the links in the network; then solutions similar to Byzantine Agreement can be used. However, this approach is expensive. The other approach is to assume a weaker failure model than a strong adversary. One such adversary, which we call a *weak adversary*, is a *probabilistic* adversary which can destroy messages with a probability p that is not known in advance. We have preliminary results that show vastly improved performance against such an adversary.

9 Acknowledgements

We are grateful to Hagit Attiya, Baruch Awerbuch, Mihir Bellare, Cynthia Dwork, Ken Goldman, Steve Ponzio and Larry Stockmeyer for their help and suggestions.

References

[1] M. Ben-Or. Another advantage of free choice. *Proceedings 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30. August 1983

[2] J. Gray. Notes on Data Base Operating Systems. *Technical Report, IBM Report RJ2183(30001)*, IBM, February 1978, pp. 291–302, 1989. (Also in Operating Systems: An advanced course, Springer-Verlag Lecture Notes in Computer Science No. 60.)

[3] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Proceedings of the 3rd Annual Symposium on Principles of Distributed Computing*, pages 50-61, 1984.

[4] L. Lamport. Time Clocks and the ordering of events in a distributed System. *Communications of the ACM*, 21(7): 558-565, 1977.

[5] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3): 382-401, July 1981.

[6] M. Rabin and D. Lehmann. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. *Proceedings of 8th ACM Symposium on Principles of Programming Languages*, pages 133-138, 1981.

A Lower Bound Based on Independence

Our second lower bound needs the following assumption. We say that the *usual case assumption* holds if:

- The graph G is connected and the diameter of G is no more than the number of rounds N.
- $\epsilon < 0.5$.

It is easy to see that these two conditions capture the usual and interesting cases. If the first condition does not hold then it can be shown that $L_i(R) \leq 1$ for all i, R, F and so by Lemma 5.4, $\mathcal{L}(F, R) \leq \epsilon$. Similarly if the second condition does not hold, the protocol is allowed to fail more than half the time. Thus the conditions preclude parameter settings that force absurdly small values of liveness and allow absurdly large values of unsafety.

Theorem A.1 Under the usual case assumption, if any protocol F has a run R such that $\mathcal{L}(F,R) > \epsilon ML(R)$ then there is some other run $\tilde{R}$ such that $\mathcal{L}(F,\tilde{R}) < \epsilon ML(\tilde{R})$.

The proof exploits a simple connection between probabilistic independence and what we call causal independence. Intuitively, two processes are causally independent if there is no causal flow, possibly through another process, that can link the two processes. For any $i, j \in V$, we say that i and j are *causally independent* in run R if there is no $k \in V$ such that $(k, 0)$ flows to (i, N) and $(k, 0)$ flows to (j, N) in R. The connection is expressed by the intuitive lemma:

Lemma A.2 If i and j are causally independent in run R then the events $(D_i|R)$ and $(D_j|R)$ are independent events.

If i and j are causally independent in run R, then there must be some restrictions on their decision probabilities in R in order to preserve the agreement property. There are several ways in which these restriction can be phrased; we select one that is sufficient for the later development.

Lemma A.3 Consider a run R in which i and j are causally independent and such that $Pr[D_i|R] = \epsilon$. Then if $\epsilon < .5$, $Pr[D_j|R] = 0$.

Proof: Let $Pr[D_j|R] = \delta$. We know that $Pr[PA|R] \geq Pr[D_i\overline{D_j}|R] + Pr[D_j\overline{D_i}|R]$. But since i and j are causally independent in R we have by Lemma A.2

that the events $(D_i|R)$ and $(D_j|R)$ are independent. Hence, $Pr[PA|R] \geq \epsilon(1-\delta)+\delta(1-\epsilon)$ and so $Pr[PA|R] \geq \epsilon + \delta(1-2\epsilon)$. But since $\epsilon < 0.5$, $1-2\epsilon > 0$. Hence by agreement, $\delta = 0$. □

For the next lemma, recall the definition of $ML_i(R)$, the modified level of process i in run R. This lemma serves to set up the proof of the following lemma, Lemma A.5.

Lemma A.4 Suppose that for all runs R and for all $i \in V$, $Pr[D_i|R] = 0$ if $ML_i(R) = 0$. Then for all R and $i \in V$, $Pr[D_i|R] \leq ML_i(R)\epsilon$.

Proof: By induction on the value of $ML_i(R)$. Let $ML_i(R) = l$.

Base Case, $l = 0$: This is the assumption of the lemma.

Inductive Step, $l > 0$: Using a lemma similar to Lemma 5.2, we can show that if $\tilde{R} = Clip_i(R)$, then there is some k such that $ML_k(\tilde{R}) = l-1$. Hence by inductive assumption, $Pr[D_k|\tilde{R}] \leq \epsilon(l-1)$. Hence by Lemma 2.2, $Pr[D_i|\tilde{R}] \leq \epsilon l$. But by Lemma 4.2 and Lemma 2.1, $Pr[D_i|R] = Pr[D_i|\tilde{R}] \leq \epsilon l$. □

Now consider a run R_1 in which only process 1 receives an input message and no other message is delivered in the run. The next lemma states that if the probability of process 1 attacking in this run is exactly ϵ, then we can prove a tighter lower bound on the decision probabilities than the bound of Lemma 5.3. Recall that the bound in Lemma 5.3 was stated in terms of $L_i(R)$.

Lemma A.5 Suppose that $R_1 = \{(v_0, 1, 0)\}$, $Pr[D_1|R_1] = \epsilon$ and $\epsilon < 0.5$. Then for all runs R and all $i \in V$, $Pr[D_i|R] \leq ML_i(R)\epsilon$.

Proof: Consider any i and any R such that $ML_i(R) = 0$. Then we will claim that $Pr[D_i|R] = 0$. To do this we consider two cases, one of which must be true if $ML_i(R) = 0$.

- $(v_0, -1)$ does not flow to (i, N) in R. Then $L_i(R) = 0$ and hence by Lemma 5.3, $Pr[D_i|R] = 0$.

- $(1, 0)$ does not flow to (i, N) in R. Thus $i \neq 1$ as $(1, 0)$ flows to $(1, N)$. Consider the run $Clip_i(R)$. By the definition of clipping, there is no tuple $(*, 1, *)$ in $Clip_i(R)$, because if there was, $(1, 0)$ would flow to (i, N) in R. Consider the run $\tilde{R} = Clip_i(R) \cup \{(v_0, 1, 0)\}$. By construction, the only tuple of the form $(*, 1, *)$ in $\tilde{R}$ is $(v_0, 1, 0)$. Hence 1 and i are causally independent in $\tilde{R}$.

  Also, $R_1 = Clip_1(\tilde{R})$ and hence $R_1 \overset{1}{\equiv} \tilde{R}$. Thus $Pr[D_1|\tilde{R}] = \epsilon$. Hence by Lemma A.3, $Pr[D_i|\tilde{R}] = 0$. But $Clip_i(\tilde{R}) = Clip_i(R)$ and so by Lemma 4.2 and Lemma 2.1, $Pr[D_i|R] = Pr[D_i|\tilde{R}] = 0$.

Thus in either case, we have shown that for any i and R, $Pr[D_i|R] = 0$ if $ML_i(R) = 0$. The lemma now follows from Lemma A.4. □

Lemma A.6 Suppose the graph G is connected and has diameter no more than N. Then there is a run R such that $ML_1(R) = ML(R) = 1$, and the only tuple of the form $(*, 1, *)$ is $(v_0, 1, 0)$.

Proof: Let T be a spanning tree of G with 1 as the root. Such a tree exists because G is connected. Next we define R as follows.

- $I(R) = \{(v_0, 1, 0)\}$ (i.e., only process 1 receives input).

- For all $i, j \in V$ and $1 \leq r \leq N$, $(i, j, r) \in R$ iff i is the parent of j in the tree. (i.e., information only flows down the tree.)

It is not hard to see that since the height of the tree is no more than N, $ML_1(R) = 1$ and $ML_i(R) \geq 1$ for all $i \in V$. Thus $ML(R) = 1$. □

We now return to the proof of Theorem A.1.

Proof: Suppose there is some protocol F such that for all R, $\mathcal{L}(F, R) \geq \epsilon ML(R)$.

By Lemma A.6, there is a run R_1 such that $ML_1(R_1) = ML(R_1) = 1$ and the only tuple of the form $(*, 1, *)$ in R_1 is $(v_0, 1, 0)$. It is easy to verify that $L_1(R_1) = 1$.

Thus by assumption, $\mathcal{L}(S, R_1) \geq \epsilon ML(R_1) = \epsilon$. Thus by Lemma 2.3, $Pr[D_1|R_1] \geq \epsilon$. Also, by Lemma 5.3, since $L_1(R_1) = 1$, $Pr[D_1|R_1] \leq \epsilon$. Hence, $Pr[D_1|R_1] = \epsilon$.

Now consider the run $R_2 = Clip_1(R_1) = \{(v_0, 1, 0)\}$. Then by Lemma 4.2 $R_2 \overset{1}{\equiv} R_1$. Thus by Lemma 2.1, $Pr[D_1|R_2] = \epsilon$. Hence by Lemma A.5, for all i, R, $Pr[D_i|R] \leq \epsilon ML_i(R)$. Thus for all R, $Min_i Pr[D_i|R] \leq Min_i \epsilon ML_i(R)$. Thus from Lemma 2.3 and the definition of $ML(R)$, $\mathcal{L}(F, R) \leq \epsilon ML(R)$.

Thus we have shown that for any protocol F, if for all R, $\mathcal{L}(F, R) \geq \epsilon ML(R)$, then $\mathcal{L}(F, R) = \epsilon ML(R)$. This implies the theorem. □