Brief Announcement: Null Messages, Information and Coordination

Raïssa Nataf
Technion, Haifa, Israel

Guy Goren
Technion, Haifa, Israel

Yoram Moses
Technion, Haifa, Israel

Abstract

This paper investigates how null messages can transfer information in fault-prone synchronous systems. The notion of an *f-resilient message block* is defined and is shown to capture the fundamental communication pattern for knowledge transfer. In general, this pattern combines both null messages and explicit messages. It thus provides a fault-tolerant extension of the classic notion of a message-chain. Based on the above, we provide tight necessary and sufficient characterizations of the generalized communication patterns that can serve to solve the distributed tasks of (nice-run) Signalling and Ordered Response.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Computing methodologies → Reasoning about belief and knowledge

Keywords and phrases null messages, fault tolerance, coordination, information flow

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.49

Related Version *Full Version*: https://arxiv.org/abs/2208.10866

Funding *Guy Goren*: Guy Goren was partly supported by a grant from the Technion Hiroshi Fujiwara cyber security research center and the Israel cyber bureau, as well as by a Jacobs fellowship. *Yoram Moses*: Yoram Moses is the Israel Pollak academic chair at the Technion. Both his work and that of Raïssa Nataf were supported in part by the Israel Science Foundation under grant 2061/19.

1 Introduction

In synchronous models with a global clock, it may be possible to transmit information by *not* sending a message, which Lamport termed *sending a null message* in [7]. While null messages have been successfully employed to optimize communication in useful protocols (see, e.g., [1, 6] for early examples), the question of how null messages convey information, and what information they convey, has only been partly addressed. In addition, when failures can occur, the use of null messages can become rather challenging. If i does not receive a message from j in such a setting, i might not be able to distinguish whether this is because j purposely refrained from sending, or because j failed. Nevertheless, recent work [4] has shown that when the number of failures is bounded (by f, say), it is still possible to use null messages to transmit information. Very roughly speaking, their "Silent Choir" theorem implies that in a failure-free execution, the only way that a process j can learn i's value without receiving an explicit message chain from i is for there to be a set of $f + 1$ processes that received such a chain from i do not send a message to j. However, this is far from being sufficient. Our purpose in this paper is to initiate a systematic analysis of the role of null messages, and obtain sharper characterizations of their use when processes can fail.

We consider the standard synchronous message-passing model with crash failures. We assume a finite set $\mathbb{P}$ of processes that are connected via a communication network, and all start at time 0. Moreover, messages are reliably delivered in one time step. We call one of the processes the "source," and denote it by s. For simplicity, we restrict our attention to the case in which s has a binary initial value $v_s \in \{0, 1\}$, while every process $i \neq s$ has a unique initial state (with a fixed initial value, say 0). We assume a bound of f on the number of processes that can crash in any given run. Finally, we focus on deterministic protocols, so a ***protocol*** Q describes what messages a process sends and what decisions it takes, as a function of its local state. In particular, a protocol Q has a single run in which $v_s = 1$ and no failures occur. We call this run Q's *nice run*, and denote it by $\hat{r}(Q)$, or simply by $\hat{r}$ when Q is clear from context. A process is said to be ***active*** at time m if it has not crashed by time $m - 1$ and it correctly follows its protocol at time m. For more about our formal model, see [8]. Our analysis makes use of a formal theory of knowledge to capture how null messages affect what processes do or do not learn. See [3] for more details and a general introduction to the topic.

When the model is synchronous, it is common to consider the event that i did not send its neighbor j a message at time t in a given run as if i sent j a null message there. Of course, j will be able to observe at time $t + 1$ that no message was received. Notice, however, that if i *never* sends j a message at time t then this will not provide j any information whatsoever. We say that i sends j a *genuine* null message at time t in a run r if, in addition, there is some run $r' \neq r$ in which i *does* send a message to j at time t. From here on, all null messages will assumed to be genuine. To capture the information conveyed by a null message, we use the following:

▶ **Definition 1** (Null message sent in case φ)**.** *We say that in protocol Q process i sends a* null message *to j at time t* in case φ *if for every run r of Q in which i is active at time t, it* does not *send a message to j at time t in r if and only if φ holds at time t in r.*

Clearly, in a failure-free system (i.e., if $f = 0$) if i sends j a null message at time t then, at time $t + 1$ process j comes to know that φ was true. In the presence of failures, however, j is not guaranteed to know this, and a more subtle analysis is required. We begin by considering the problem of transmitting information about the initial value v_s of the source process to another process $j \neq s$. More precisely, we define a problem called *nice-run signalling* (NS) in the following manner. Following [2], we use the notation $\langle i, t\rangle$, which we call a *process-time node* to stand for process i at time t. A protocol Q is said to *solve* nice-run signalling (NS) between $\langle s, 0\rangle$ and $\langle j, m\rangle$ if $K_j(v_s = 1)$ holds at time m in Q's nice run $\hat{r}(Q)$. Instances of NS often appear when optimizing the communication costs of protocols that solve other distributed tasks (e.g., optimizing the good-case costs of Consensus [5]).

2 f-resilient message block

We now turn to study the communication patterns that protocols solving NS and related problems can use in their nice runs. We focus on "communication graphs," denoted by $CG_Q(r) = (\mathbb{V}, E_m, E_n, E_\ell)$, that account for the messages and the null messages that are sent in a run r of a given protocol Q. The set $\mathbb{V}$ of nodes of the graph consists of all process-time nodes $\theta = \langle i, t\rangle$, with $t \geq 0$. The set E_m consists of directed edges (θ, θ') such that a message is sent in r at θ and delivered to θ', while E_n consists of directed edges (θ, θ') such that a (genuine) null message is sent in r from θ to θ'. Finally, E_ℓ consists of all edges of the form $(\langle i, t\rangle, \langle i, t + 1\rangle)$, $i \in \mathbb{P}$ and $t \geq 0$, between consecutive nodes along the timeline of a process.

In general, a path in the communication graph $CG_Q(r)$ records a chain consisting of both actual messages and null messages. We therefore refer to it as a *weak* message chain. We are now ready to define a primitive that plays an essential role in solutions to NS.

▶ **Definition 2** (f-resilient message block). *Let $\theta, \theta' \in \mathbb{P} \times \mathbb{N}$ be two nodes. An* f-resilient message block from θ to θ' in $CG_Q(r)$ *is a set Γ of paths between θ and θ' such that for all sets $B \subset \mathbb{P}$ with $|B| \leq f$ there is a path in $CG_Q(r)$ that does not contain null messages sent by processes in B.*

Notice that a path that does not contain *null* messages sent by a process j can still contain messages sent by j. As a result, if the adversary crashes j, this path may still convey information. In a precise sense, f-resilient message blocks are both necessary and sufficient for solving nice-run signalling, and we can obtain a *tight* characterization of the communication patterns needed for solving NS:

▶ **Theorem 3.**

- (Necessity) *If a protocol Q solves* NS *from $\theta_s = \langle s, 0\rangle$ to $\theta_j = \langle j, m\rangle$, then there must be an f-resilient message block from θ_s to θ_j in $CG_Q(\hat{r})$. (Recall that $\hat{r}$ is Q's nice run.)*
- (Sufficiency) *If a communication graph CG contains an f-resilient message block between $\theta_s = \langle s, 0\rangle$ and $\theta_j = \langle j, m\rangle$, then there exists a protocol Q with $CG_Q(\hat{r}) = CG$. that solves* NS *between θ_s and θ_j.*

Beyond direct information transfer as captured by the NS problem, we proceed in [8] to consider a coordination problem called Ordered Response (OR) in which processes must perform actions in a linear temporal order as a reaction to a spontaneous event (See [2]). Namely, each process $i_h \in \{i_1, i_2, \ldots, i_k\}$ has a specific action a_h to perform, and these actions should be performed only if initially $v_s = 1$. Moreover, they need to be performed in temporal order. I.e., denoting by t_h the time at which i_h performs a_h, it is required that $t_1 \leq t_2 \leq \cdots \leq t_k$. Finally, we consider the variant in which all actions are performed in the nice run. One way to solve this while ensuring no OR violation in any run is, roughly speaking, to create f-resilient message blocks in $CG(\hat{r})$, i.e., solving NS between each consecutive pair of processes in the order, to inform i_{h+1} that i_h has acted. This would be governed by Theorem 3. However, information may also be transferred indirectly: for instance, if process i_3 knows that i_2 knows that $v_s = 1$ and that f processes – not including i_2 have failed – then it can infer that a_2 has been performed, and so i_3 can "safely" perform a_3. For characterizations of the communication patterns used in solutions to OR, see [8].

References

1. Eugene S. Amdur, Samuel M. Weber, and Vassos Hadzilacos. On the message complexity of binary byzantine agreement under crash failures. *Distributed Computing*, 5(4):175–186, 1992.
2. Ido Ben-Zvi and Yoram Moses. Beyond Lamport's happened-before: On time bounds and the ordering of events in distributed systems. *Journal of the ACM (JACM)*, 61(2):1–26, 2014.
3. Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning About Knowledge*. MIT Press, 1995. doi:10.7551/mitpress/5803.001.0001.
4. Guy Goren and Yoram Moses. Silence. *J. ACM*, 67:3:1–3:26, 2020. doi:10.1145/3377883.
5. Guy Goren and Yoram Moses. Optimistically tuning synchronous Byzantine consensus: another win for null messages. *Distributed Computing*, 34(5):395–410, 2021.
6. Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for byzantine agreement. *Mathematical Systems Theory*, 26(1):41–102, 1993.
7. Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Program. Lang. Syst.*, 6:254–280, 1984. doi:10.1145/2993.2994.
8. Raïssa Nataf, Guy Goren, and Yoram Moses. Null messages, information and coordination: Preliminary report. *CoRR*, abs/2208.10866, 2022. arXiv:2208.10866.